C. Språket för inbyggda system

- Maskinnära
- De flesta datorresurser åtkomliga för programmeraren
- Samma operatorer som Java (som ni redan kan)

Du har läst C i Datortekniken

http://www.ict.kth.se/courses/IS1200/

- **Hemlab 1**. Maskinnära programmering med C
- **Hemlab 3**. Trådar och semaforer

Koden till hemlab 3 är ett typexempel på hur man kommenterar och publicerar kod.
(Du kommer ju senare att behöva skriva en rapport …)

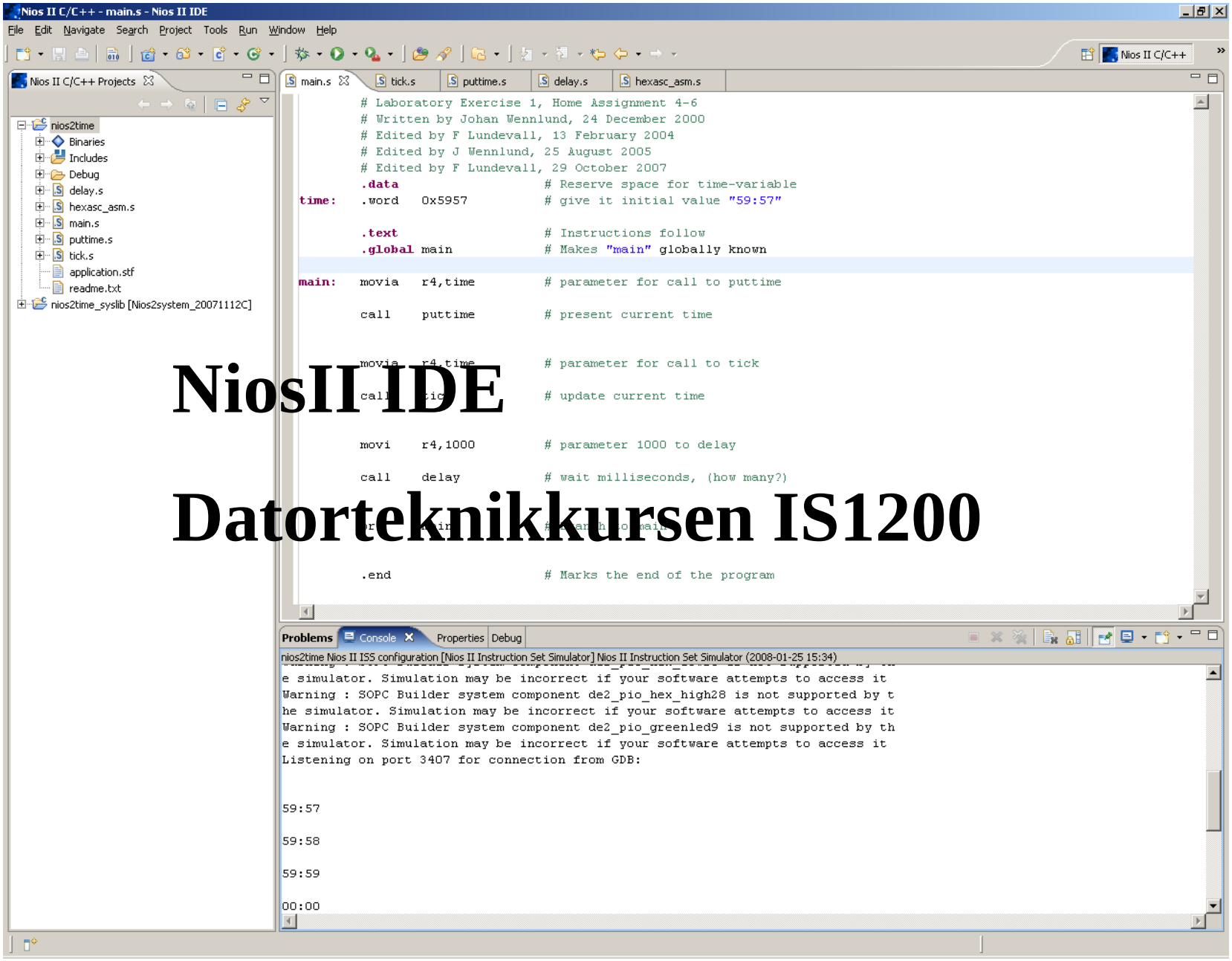
”Pretty-printed source code (PDF)”

Integrated Development Environment **IDE**

En **utvecklingsmiljö** (*Integrated Development Environment*, IDE), är ett datorprogram eller en programsvit som vanligtvis innehåller en textredigerare, kompilator, och debugger, tillsammans med ett antal andra funktioner avsedda att underlätta vid programmering.

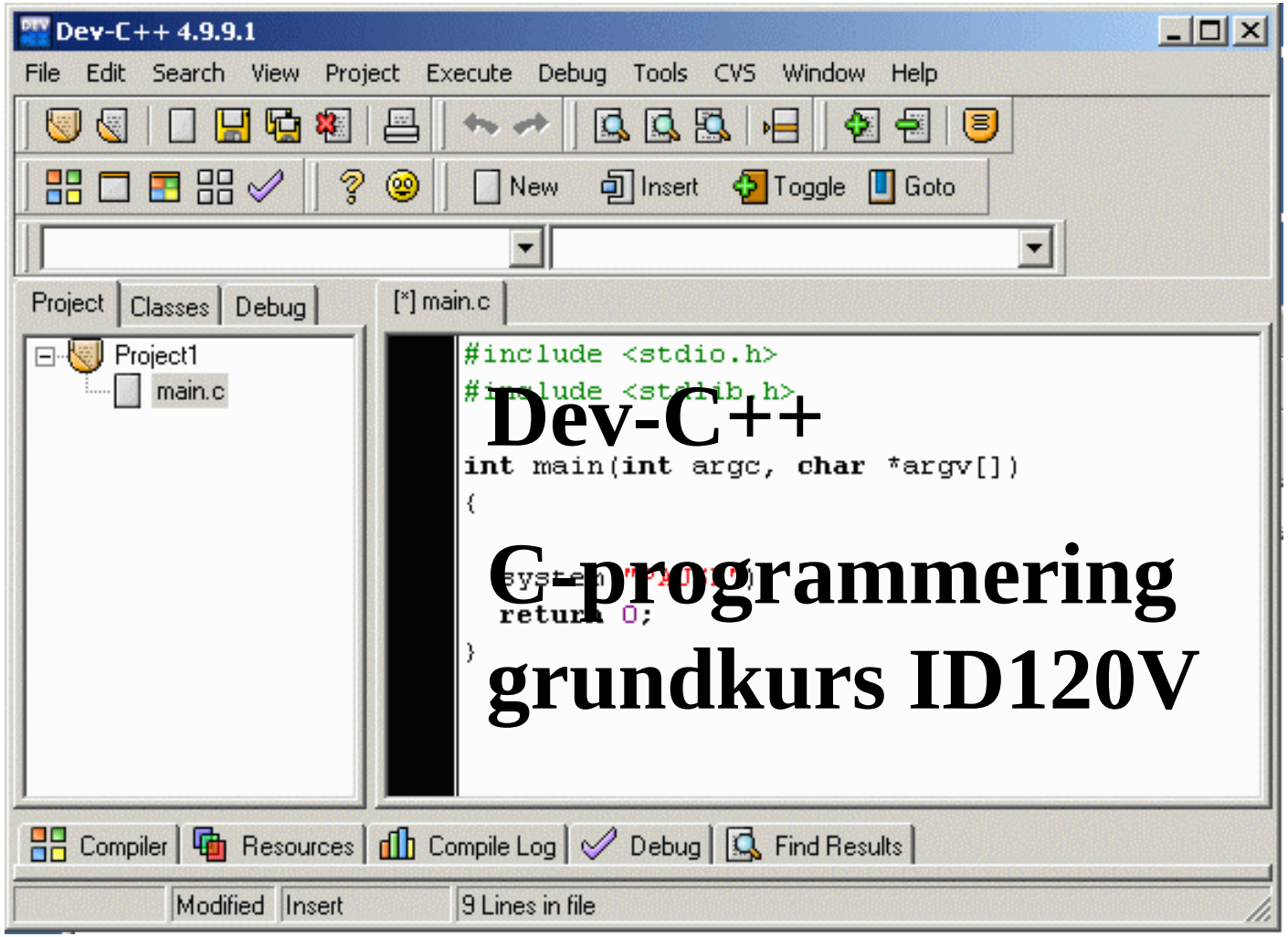
I skolan finns C-kompilatorer installerade i datorsalarna på plan 6:
NiosII IDE, **Dev-C++**, **Code::Blocks**

Till denna kurs IS1300 behöver Du **IAR Embedded Workbench** på din egen dator.



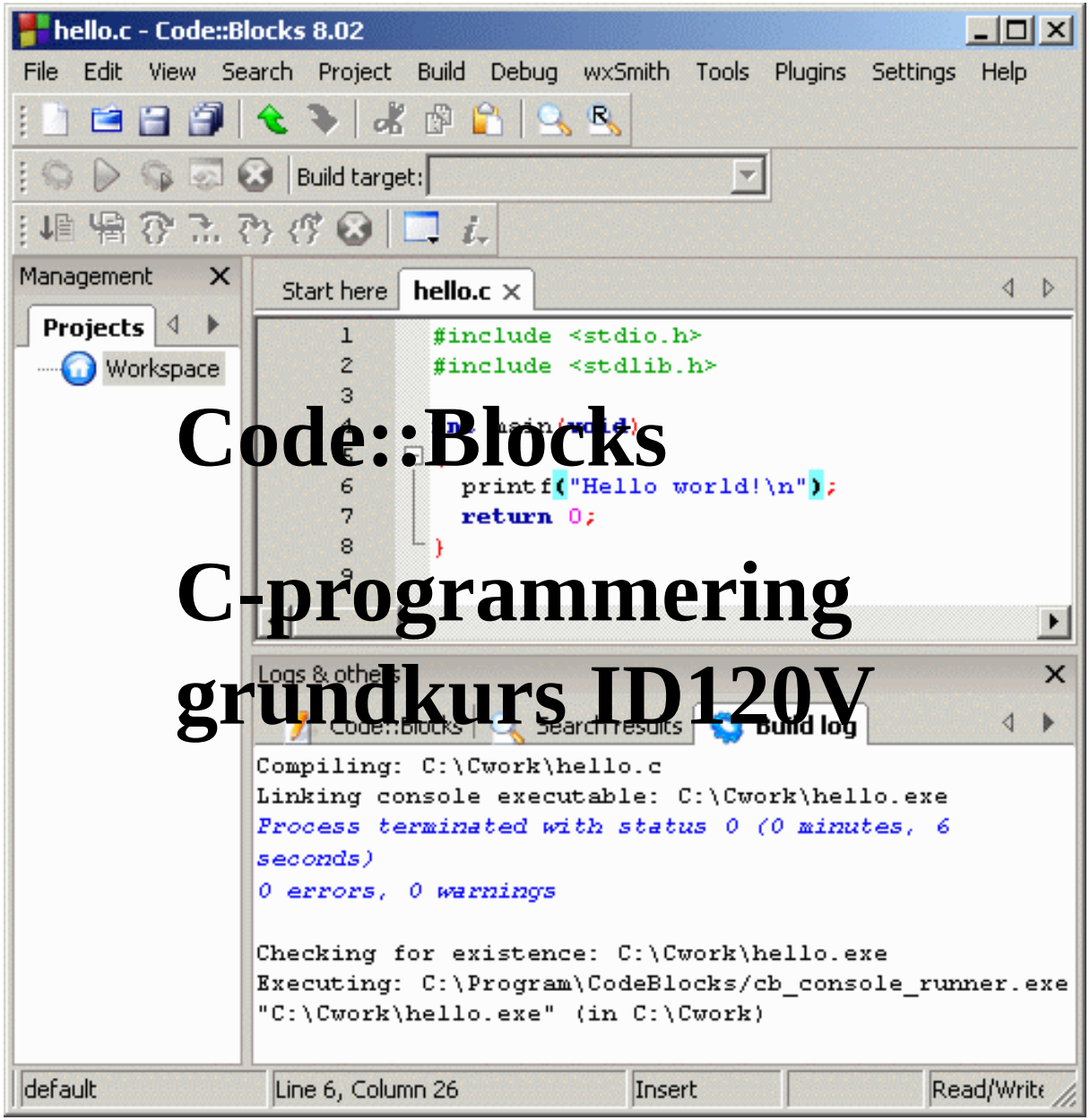
NiosII IDE

Datorteknikkursen IS1200



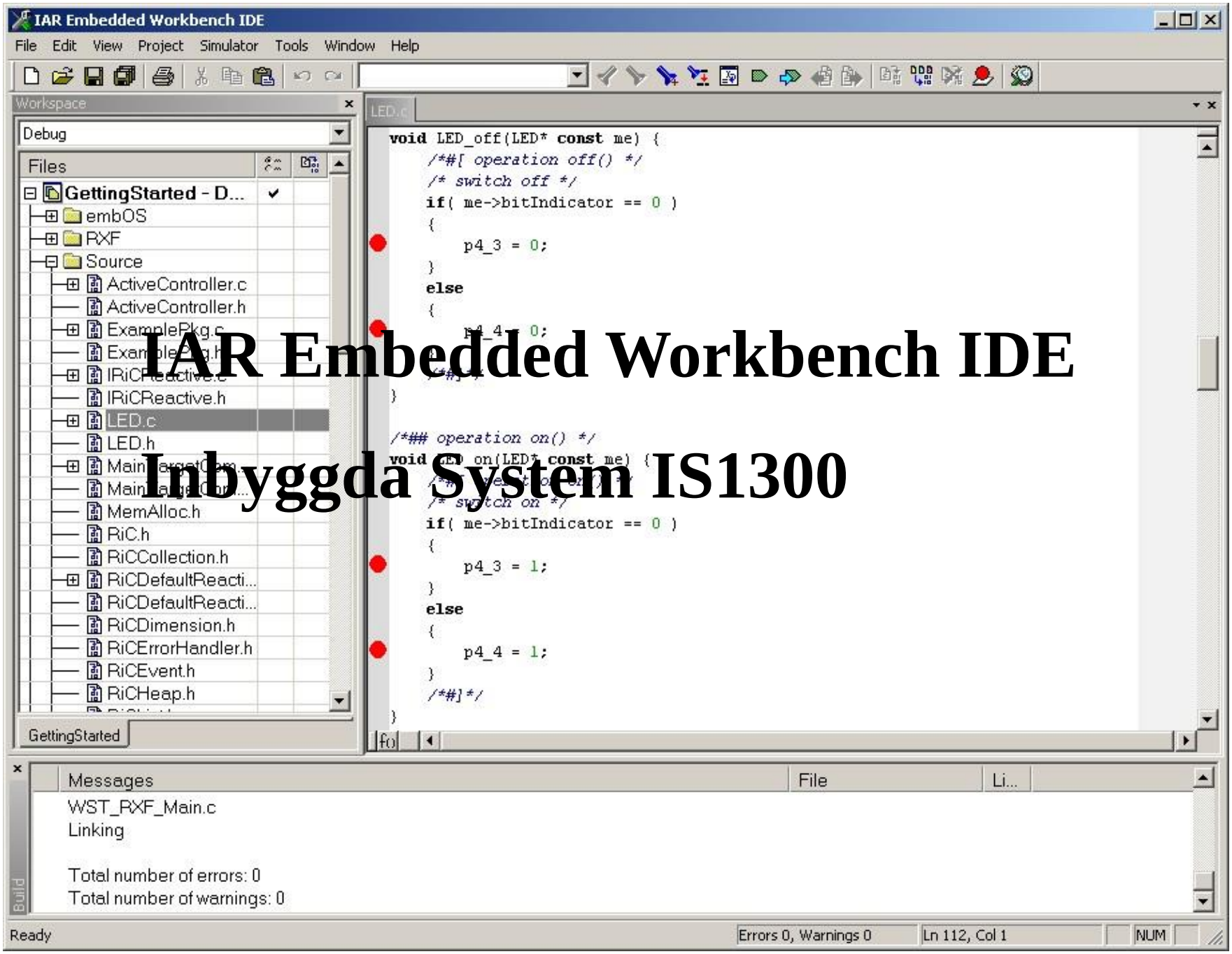
Dev-C++

C-programmering grundkurs ID120V



Code::Blocks

C-programmering grundkurs ID120V



IAR Embedded Workbench IDE

Inbyggda System IS1300

Prova för att få svar!

Det bästa sättet för att få svar på sina C-frågor är att skriva och köra korta C-program som skriver ut svaret med **`printf()`**.

Det går fortare än att fråga läraren, eller att ringa någon expert – och dessutom blir ju svaret direkt det rätta!

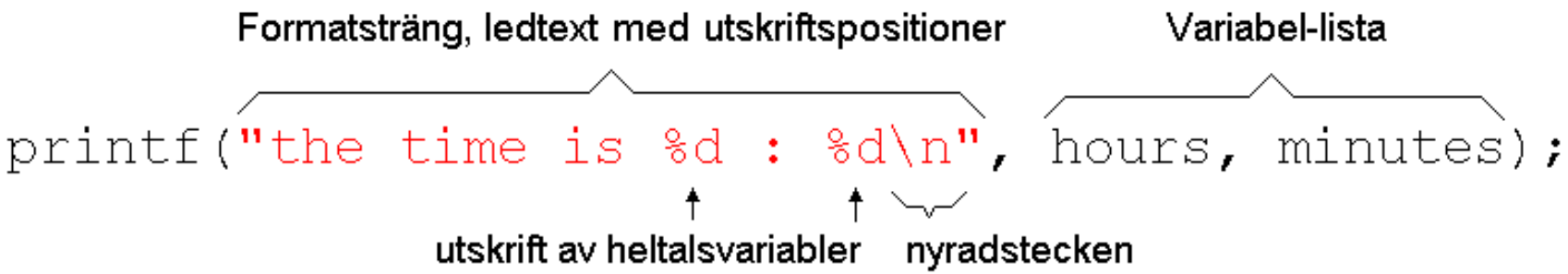
Får Du svaret från någon ”expert” måste Du ju ändå prova …

Men om nu målsystemet inte har någon bildskärm att skriva ut svaret på?

IAR Embedded Workbench har ett ”**Terminal Output**” fönster för felsökning, där man kan se vad **`printf()`** skriver!

Formaterad utskrift **`printf()`**

the time is 12 : 34



- **`%d`** utskrift av heltalsvariabel
- **`%x`** utskrift av heltalsvariabel men hexadecimalt
- **`%f`** utskrift av flyttalsvariabel
- **`\n`** nyradstecken **`\t`** tabulatortecken

För att använda `printf()`

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

printf() (scanf())

När man har frågor som gäller målsystemet måste man göra sina prov-program för det, men ofta gäller frågorna hur C-språket fungerar och då kan man ju använda **valfri C-kompilator** om man tycker det blir smidigare eller snabbare!

Funktionen **`scanf()`** används för inmatning av data. Den har en formatsträng liknande **`printf()`**.
Oftast kan man initiera variabler direkt i koden med önskade testvärden, utan att använda **`scanf()`**, och sedan ändra och kompilera om, varje gång man vill prova med andra värden.

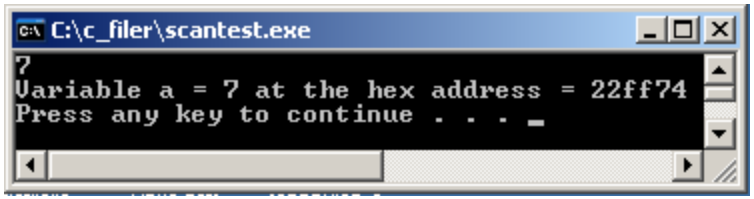
Inmatning med **scanf()**

```
int a;

scanf("%d", &a);
```

Inmatning av ett heltal till variabeln **a**. Formatsträngen fungerar här på samma sätt som för **printf()**. Avsluta inmatningen med retur.

Adressoperatorn &a betyder *adressen* till **a**. `scanf()` lägger det inmatade talet på **a**:s adress. (Dvs. i **a**).

```
printf("Variable a = %d at the hex address = %x\n", a, &a);
```



För att använda **scanf()**

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

William Sandqvist william@kth.se

C Variabelnamn

- siffror, _ , bokstäver (gemena, versaler) ej å ä ö, går bra
- *interna variabelnamn* är signifikanta för åtminstone de 31 tecknen
- skiljer på versaler och gemena, `Kapital` ≠ `kapital`
- följande tecken är *reserverade ord* (nyckelord) och får inte användas

C:s Reserverade ord (Nyckelord)

följande tecken är *reserverade ord* (nyckelord) och får inte användas:

auto	break	case	char	const
continue	default	do	double	else
enum	extern	float	for	goto
if	int	long	register	return
short	signed	sizeof	static	struct
switch	typedef	union	unsigned	void
volatile	while			

Vem skulle komma på idén att använda dessa?

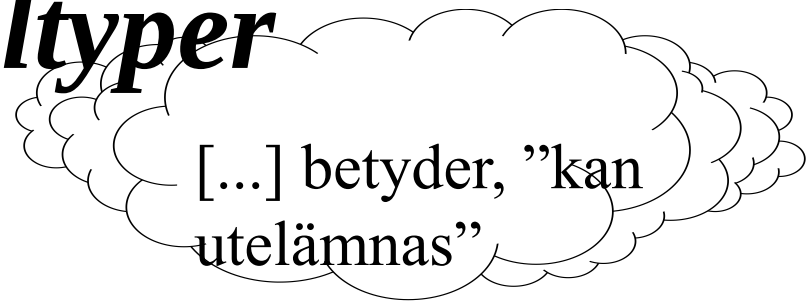
C:s Reserverade ord (Nyckelord)

följande tecken är *reserverade ord* (nyckelord) och får inte användas:

auto	break	case	char	const
continue	default	do	double	else
enum	extern	float	for	goto
if	int	long	register	return
short	signed	sizeof	static	struct
switch	typedef	union	unsigned	void
volatile	while			

Jag kallade en gång en variabel för switch, den variabeln var för en strömbrytare …

C:s Variabeltyper

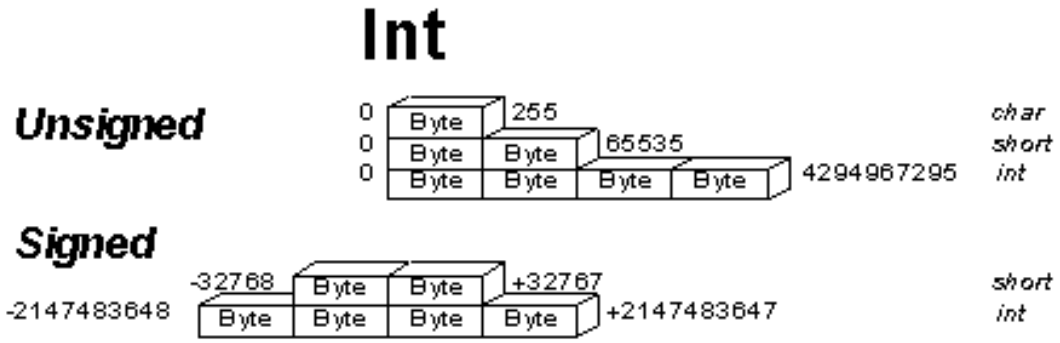


- **heltalstyper i ANSI-C, fördefinierade**

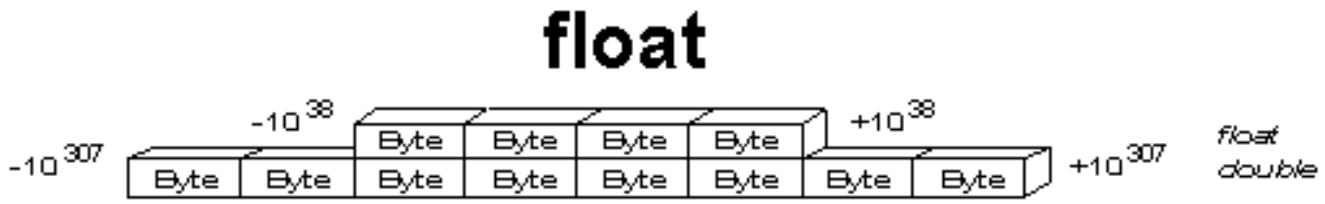
–`signed char`	minst 1 byte
–`unsigned char`	minst 1byte
–`[signed] short [int]`	minst 2 byte
–`unsigned short [int]`	minst 2 byte
–`[signed] int`	minst 2 byte
–`unsigned [int]`	minst 2 byte
–`[signed] long [int]`	minst 4 byte
–`unsigned long [int]`	minst 4 byte

C's Variabeltyper

Kanske så här? Heltalsvariabler: **char**, **short int**, **int**, **long signed** eller **unsigned**



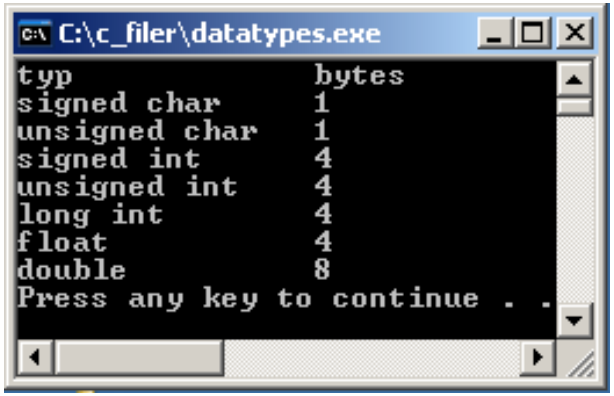
Flyttalsvariabler: **float**, **double**



Hur många Byte är en int?

Oklart! Det får man veta genom att köra ett testprogram på sitt målsystem:

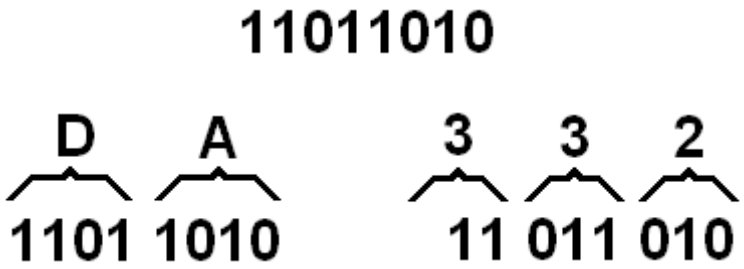
```
#include <stdio.h>
int main(void)
{
  printf("typ\t\tbytes\n");
  printf("signed char \t%d\n", sizeof(char));
  printf("unsigned char \t%d\n", sizeof(unsigned char));
  printf("signed int \t%d\n", sizeof(int));
  printf("unsigned int \t%d\n", sizeof(unsigned int));
  printf("long \t\t%d\n", sizeof(long int));
  printf("float \t\t%d\n", sizeof(float));
  printf("double \t\t%d\n", sizeof(double));

  system("PAUSE");
  return 0;
}
```



Dec – Bin – Hex – Okt

Visst kommer
Du ihåg . . .

Dec	Bin	Hex	Dec	Bin	Hex
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F



$218_{10} = 11011010_2 = DA_{16} = 332_8$

Decimalt, Hexadecimalt och Oktalt i **C**

Decimala konstanter skrivs som vanligt:

218

Hexadecimala konstanter skrivs:

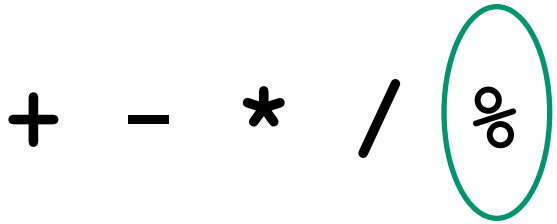
0xDA

Oktala konstanter skrivs:

0332 (OBS! Se upp med nollor framför ”decimaltal”)

Binära konstanter finns ej i **C** !

C:s Aritmetiska operatorer



Aritmetiska operatorer	
Operator	**Beskrivning**
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
%	Modulo, rest

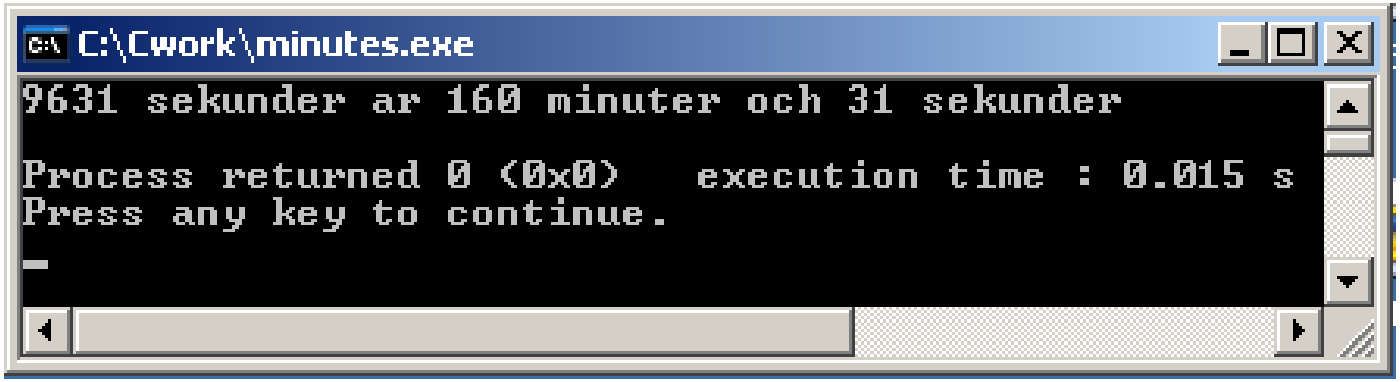
Modulo operatorn ("%") är förmodligen mindre känd än de övriga operatorerna. Den gör en heltals-division, och resultatet blir divisionens rest.

Heltalsdivision: 22/3 = 7
Modulo, resten: 22%3 = 1

Minuter och sekunder

Hur många minuter är 9631 sekunder?

```
int sek = 9631;

printf("%d sekunder är %d minuter och %d sekunder\n",
sek, sek/60, sek%60 );
```



Var tionde gång …

```
 . . .
if (n%10==0)
 {
    printf(”You are number ten!\n");
    n=n+1;
 }
 . . .
```

Operatorer till villkorsuttryck

```
< > <= >= == != && || !
```

Vilkorsoperatorer	
Operator	**Beskrivning**
<	Mindre än
>	Större än
< =	Mindre än eller lika med
> =	Större än eller lika med
= =	Lika med
! =	Inte lika med

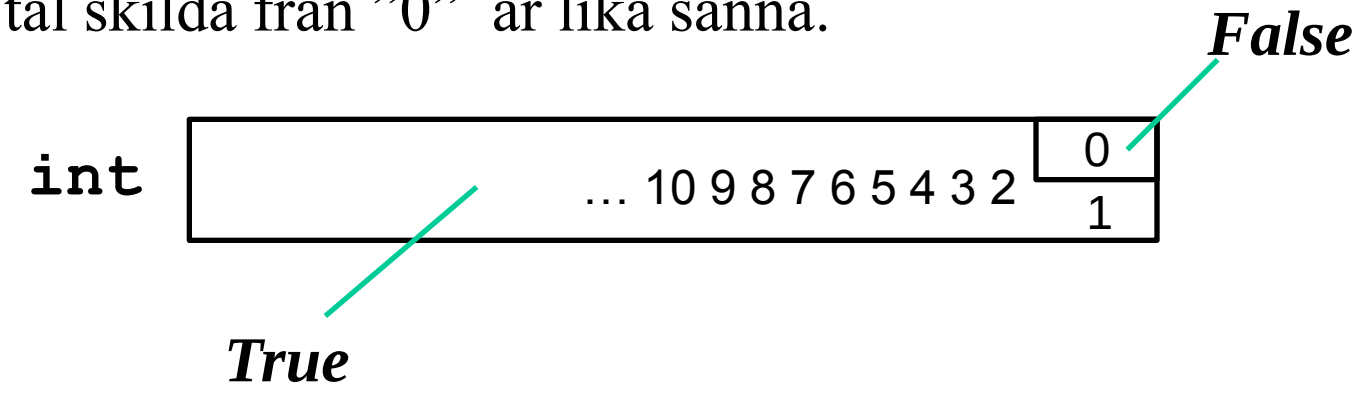
Logiska operatorer	
Operator	**Beskrivning**
&&	Logiskt AND
||	Logiskt OR
!	Logiskt NOT

C har ingen Boolsk variabeltyp SANT/FALSKT utan använder heltalsvariabler till detta.

För C är SANT *alla* heltal förutom 0 som är FALSKT!

Sant och Falskt?

Alla programspråk använder en heltalsvariabel för att lagra sant/falskt 1/0 (det finns i allmänhet inte mindre minne att tillgå). C gör likadant, men använder hela talområdet. ”0” är falskt, och alla andra heltal är ”sanna”.
Om C ska uttrycka att något är sant, väljes talet ”1”, men alla andra tal skilda från ”0” är lika sanna.



Sant och Falskt?

```
int main()
 {
     . . .
    return 0;
 }
```

Main returnerar ”0” till operativsystemet som startat programmet. Det betyder att allt gick bra.
1, 2 … 17 betyder att det gick dåligt på 1, 2 … 17 :e sättet (sant/falskt och dessutom felkod).

Sant och Falskt?

heltal

```
if (a = b) . . . ;
```

heltal kan vara sant/falskt

Kanske menade man (?):

```
if (a == b) . . . ;
```

Eller användbart. Så här läser man tecken ända tills man avbryts av strängslutstecknet (0):

```
if (a = getchar()) . . . ;
```

Villkorsoperatorn

`uttryck 1 ? uttryck 2 : uttryck 3`

Beräkningen går till så att *uttryck 1* beräknas först.
Om detta uttryck får ett värde som inte är noll (*sant*) så beräknas därefter *uttryck 2* och resultatet av hela villkorsuttrycket blir lika med värdet av *uttryck 2*.
Om å andra sidan *uttryck 1* får vardet 0 (*falskt*) beräknas i stället *uttryck 3* och resultatet av hela villkorsuttrycket blir lika med värdet av *uttryck 3*.

Exempel med villkorsoperator

```
/* absolutbeloppet av ett heltal x */
     (x) > 0 ? (x) : -(x);
```

?-operatorn ger kompakt men svårbegriplig kod, den används ofta i ”makron” – Du kommer nog att stöta på den …

Detta menas:

```
/* absolutbeloppet av ett heltal x */
     if (x > 0) x = x;
     else x = -(x);
```

since when did C programmers worry if anyone else could read their code?

”bit för bit” operatorer

& | ^ ~
<< >>

"bit för bit" operatorer	
Operator	**Beskrivning**
&	AND
|	OR
^	EXOR
~	Complement (NOT)
< <	Vänsterskift
> >	Högerskift

"bit för bit" operatorerna används för att "vaska ut" information om enskilda bitar i en variabel.

Dessa viktiga operatorer har Du redan mött och använt i Datorteknik-kursen!

William Sandqvist william@kth.se

Utskrift av binärtal?

```
void bin_prnt_byte(int x)
 {
   int n;
   for(n=0; n<8; n++)
    {
      if((x & 0x80) !=0)
        printf("1");
      else
        printf("0");
      x = x<<1;
    }
  }
```

C:s printf() har formatet hexadecimala tal, med skriver ***inte*** ut binära tal – har Du behov av att skriva ut binära konstanter så skapar Du själv en funktion för detta.

Tilldelning

Tilldelningsoperatorn ("=") låter en variabel bli lika med en annan. `x = y;`

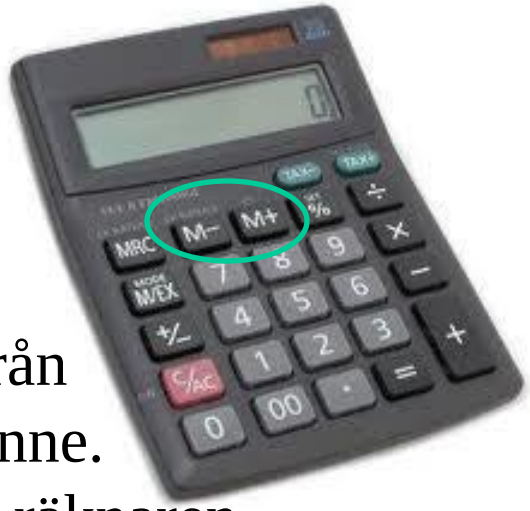
I C-språket kan man kombinera tilldelningsoperatorn med de aritmetiska operatorerna, eller "bit för bit" operatorerna, till **bekväma förkortningar**, för att erhålla speciella tilldelningsoperatorer.

Speciella tilldelningsoperatorer		
Uttryck	**Ekvivalent uttryck**	**Operation**
x += y	x = x + y	Addition
x -= y	x = x - y	Subtraktion
x *= y	x = x * y	Multiplikation
x /= y	x = x / y	Division
x %= y	x = x % y	Modulo
x &= y	x = x & y	AND
x |= y	x = x | y	OR
x ^= y	x = x ^ y	EXOR

```
M = M + y;
```

Special!

```
M += y;
```



Addera eller subtrahera till/från ackumulator minne.

M+ och M- på räknaren

William Sandqvist william@kth.se

Increment och Decrement

En vanlig uppgift i program är att öka eller minska en variabel med ett. Det går naturligtvis att göra med addition eller subtraktion, men de flesta processorer har även speciella, optimerade, instruktioner för detta. C-språket har därför Increment (öka med ett) och Decrement (minska med ett) -operatorer:

```
x++ ; /* add 1 to x */
x-- ; /* subtract 1 from x */
```

Operatorerna kan även skrivas **`++x`** och **`--x`**. Då utförs ökningen/minskningen av `x` *innan* variabeln används (annars *efter*). Med dessa fyra varianter kan programmeraren skräddarsy uttryck så att de blir så korta och effektiva som möjligt.

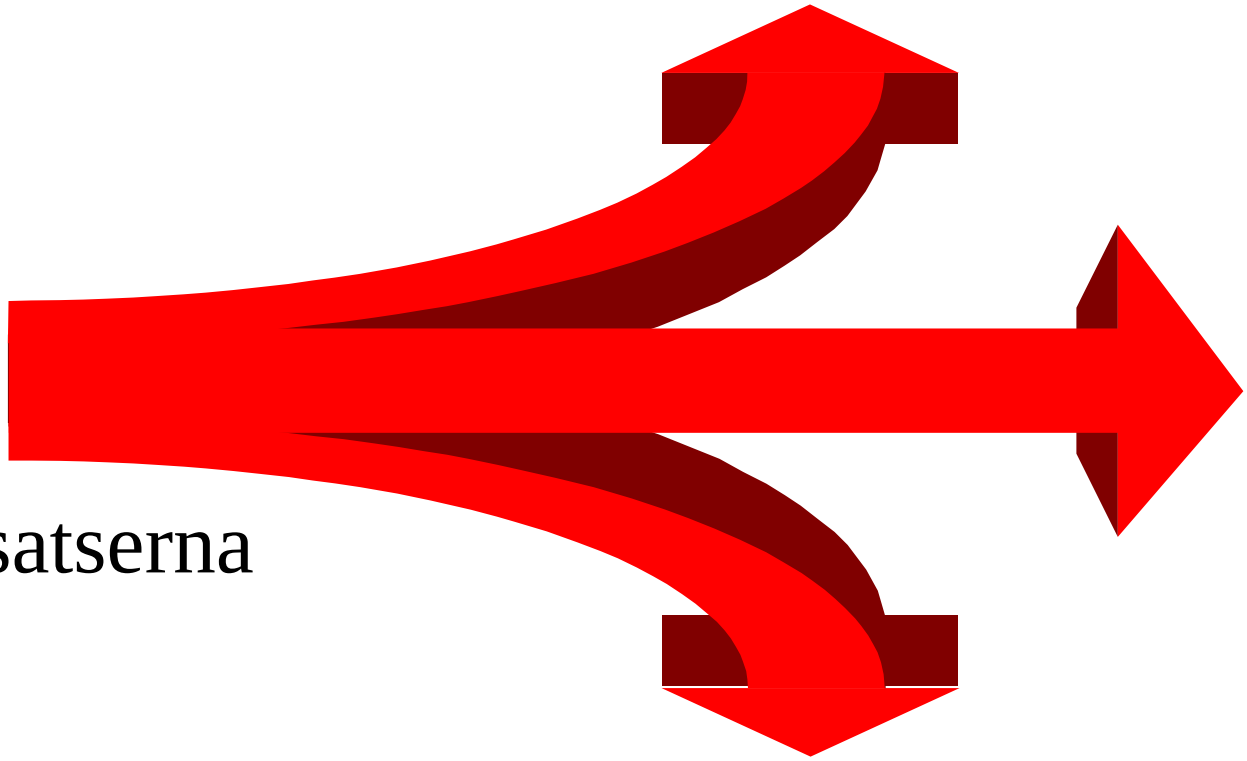
Selektion i C

”vägval”

`if`

och

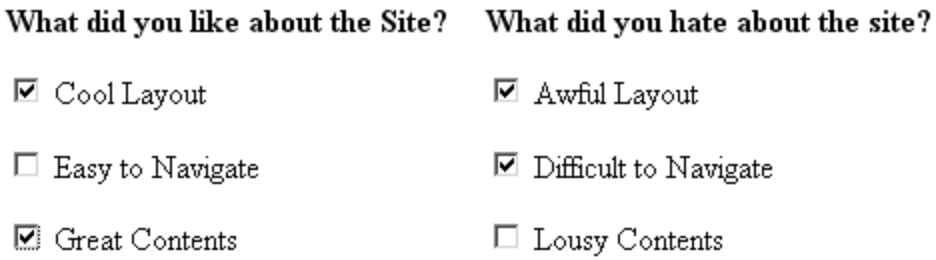
`switch` –satserna



Checkbox eller Radiobutton?

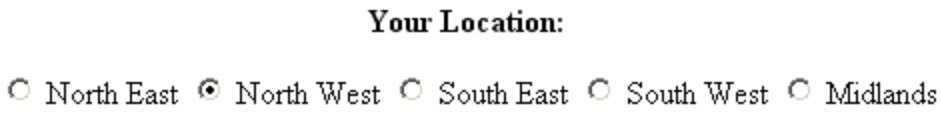
Checkbox:

```
if(a)b; if(c)d; if(e)f; . . .
```



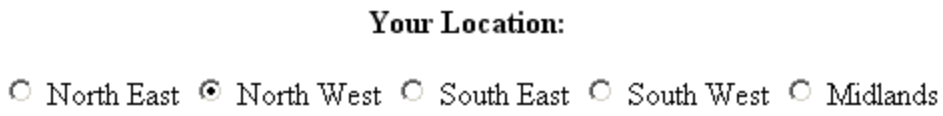
Radio Button:

```
if(a)b; else if(c)d; . . . else f;
```



Radiobutton …

Att välja endast **ett alternativ** bland flera …

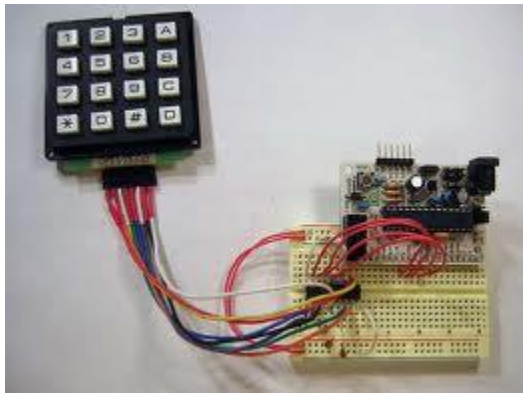



```
if(a) b;
else if(c) d;
else f;
```

William Sandqvist william@kth.se

C's `switch` – `case`

```
Switch( a )
{
   case 1:
     b = 5;
     break;
   case 2:
     b = 7;
     break;
   case 3:
     b = 10;
     break;
   default :
     b = 0;
}
```



Omkodning.
Tangentbordet avger en annan kod **a** än vad som är ingraverat på tangenten **b**!

a är en heltalsvariabel. Här sker omkodning mellan **a** och **b**.

Break behövs för att inte **alla efterföljande** alternativen **också** ska utföras.

Mycket användbar – vi återkommer till denna vid "tillstånds-diagram"

William Sandqvist william@kth.se

Smidig meny-hantering

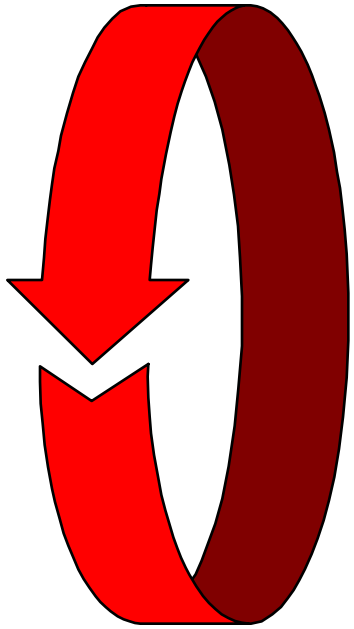
```
switch( choice )
    {
      case 'Y' : /* Yes */
      case 'y' : /* yes */
      case 'J' : /* Ja  */
      case 'j' : /* ja  */
        printf( "As you wish" );
        break;
      case 'N' : /* No Nej */
      case 'n' : /* no nej */
        printf( "Ok. You don't need to" );
        break;
      default  :
        printf(″Wrong answer, Y/y/J/j/N/n");
    }
```

Gruppera alternativen

Default, för alla ospecifierade alternativ

Iteration i C

”upprepa”



Upprepa med

`WHILE` -

och

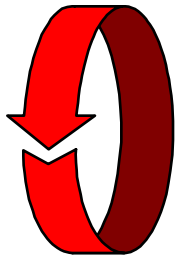
`FOR` - satserna

Iteration i C

”upprepa”

uttryck_1 ;

`while` (*uttryck_2*){



sats

uttryck_3 ;

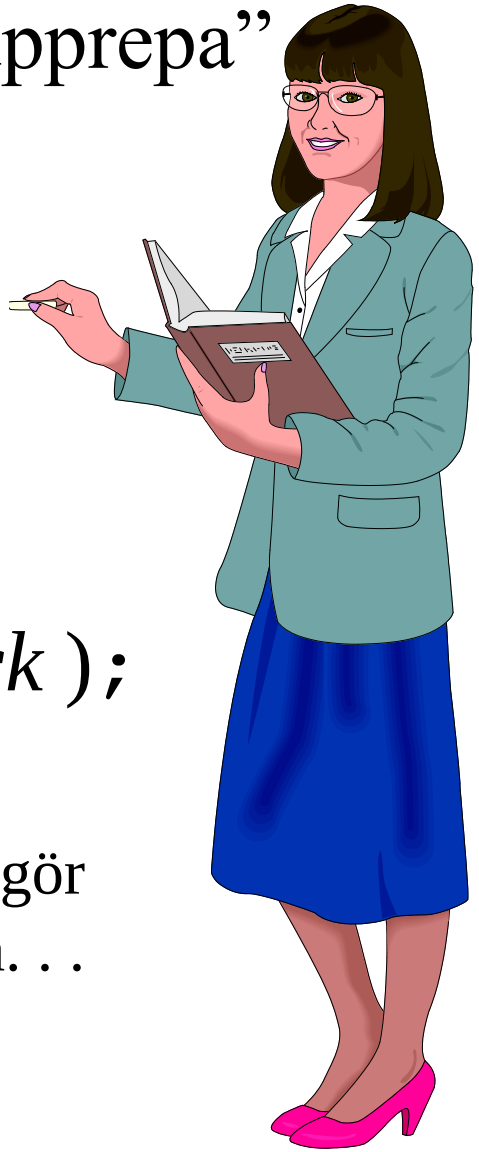
}

`for` (*uttryck_1 ; uttryck _2 ; uttryck_3*)

sats

Iteration i C

”upprepa”



Det finns ytterligare en iterations-möjlighet med följande syntax

```
do {
   sats
   sats
} while ( uttryck );
```

Utförs åtminstone en gång – gör först, tänk efter sen metoden. . .

C:s for-loop

Se **zenon.ppt** i kursen ID120V !

William Sandqvist william@kth.se

Zenons paradox och for-loopen

Den snabbfotade Achilles ska tävla mot en sköldpadda.

Zenon säger att Achilles aldrig kan hinna ifatt sköldpaddan om denna får **en meters** försprång …

William Sandqvist william@kth.se

Varför skulle Achilles *inte* kunna hinna ikapp sköldpaddan?

När Achilles hunnit ifatt försprånget på 1 m, har sköldpaddan redan hunnit ytterligare en bit, och när Achilles väl hunnit fram till den punkten så har ju sköldpaddan gått vidare …

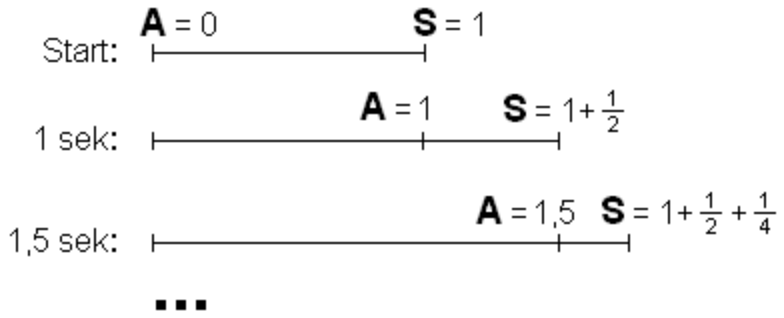
Resonemanget kan sedan fortsättas i det oändliga!

William Sandqvist william@kth.se

Matematikerns svar

Achilles, **A** hastighet 1 m/s

Sköldpaddan, **S** hastighet ½ m/s
försprång 1 m



ikapp efter ett oändligt antal iterationer vid sträckan :

$$S = 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \bullet\bullet\bullet$$

$$2 \cdot S = 2 + \underbrace{1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \bullet\bullet\bullet}_{= S}$$

$$2 \cdot S = 2 + S \quad \Rightarrow \quad S = 2$$ ikapp!

William Sandqvist william@kth.se

Vi tar hjälp av C:s for-loop

```
for(s=0, x=1, count=1; count<=limit; count++, x*=2)
 {
   s += 1.0/x ;
 }
```

$$S = 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \bullet\bullet\bullet$$

C:s **kommaoperator** gör att vi direkt kan initiera sträcka och iterationsnummer.

Vi kan även räkna fram 2-potensen och uppdatera iterationsnumret varje varv.

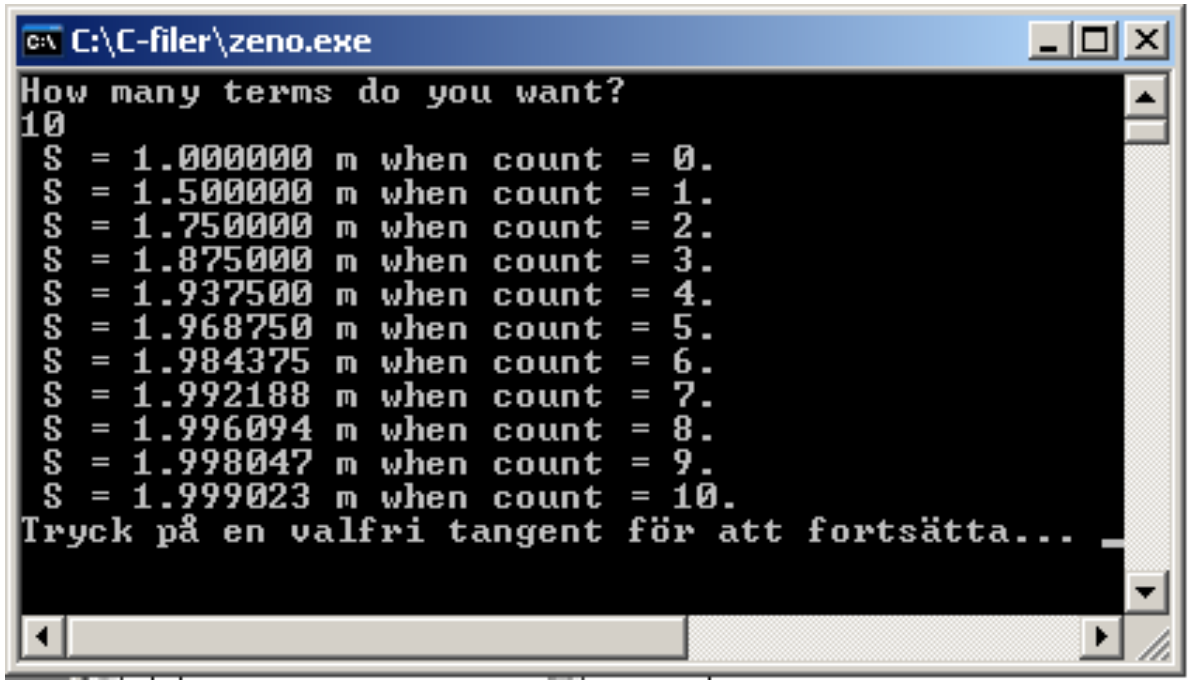
I for-loopens ”kropp” räknas den aritmetiska serien fram.

Programmet **zeno.c**

```
/* zeno.c -- series sum */
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
  int count; float s, x; int limit;
  printf("How many terms do you want?\n");
  scanf("%d", &limit);
  for( s=0, x=1, count=0; count<=limit; count++, x*=2)
   {
     s += 1.0/x;
     printf(”S = %f at count = %d.\n", s, count);
   }
  system("PAUSE");
  return 0;
}
```

Exemplet är från: **C Primer Plus**, *Stephen Prata* ISBN 1-57169-161-8

Programkörning



Programkörningen gör det troligt att Achilles hinner ikapp sköldpaddan inom 2 meter från startlinjen. Achilles har hastigheten 1 m/s, så detta kommer att ske efter *två* sekunder.

C:s for-loop är flexibel

En nedräkning … (annars kan man inte skjuta upp raketer)

```
for( secs = 5 ; secs > 0 ; secs--)
```

Öka i steg om 13 …

```
for( n = 2 ; n < 60 ; n +=13)
```

Räkna bokstäver i stället för siffror …

```
for( ch = 'A' ; ch <= 'Z' ; ch++)
```

William Sandqvist william@kth.se

C:s for-loop är flexibel

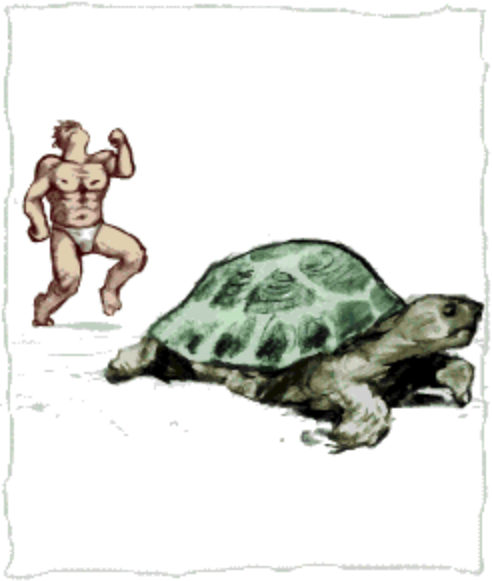
Procentuell ökning …

```
for(skuld=100.0;skuld<150.0;skuld*=1.1)
```

Evighets-slinga …

```
for( ; ; )
```

Achilles springer …



Fysik i Lund Achilles och Sköldpaddan

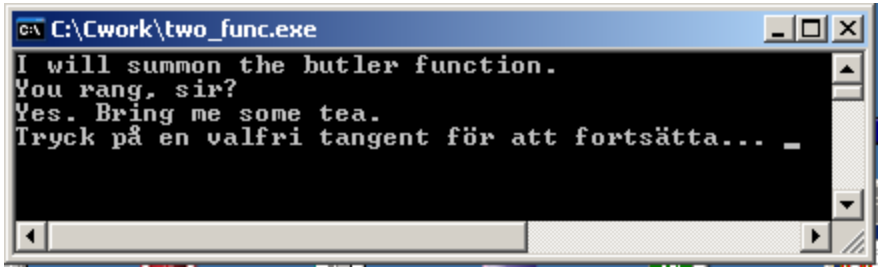
William Sandqvist william@kth.se

C är funktionsbaserat

Se **funktioner.ppt**
i kursen ID120V !

Enkelt program med funktion

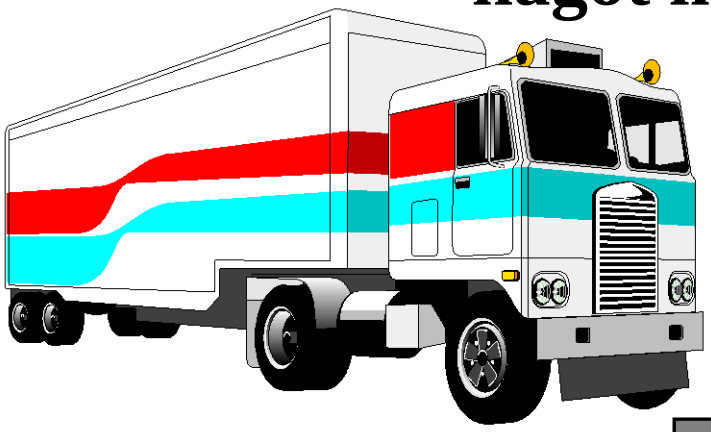
```
/* two_func.c - a program that uses two functions in one file */
/* from Stephen Prata C Primer Plus ISBN 1-57169-161-8        */

#include <stdio.h>
#include <stdlib.h>
void butler(void); /* ANSI C function prototype */

int main( void)
{
  printf("I will summon the butler function.\n");
  butler(); printf("Yes. Bring me some tea.\n");
  system("PAUSE");
  return 0;
}

void butler(void) /* start of function definition */
{
  printf("You rang, sir?\n");
}
```



Funktioner

Parametrar

något in och något annat ut



Funktion

Funktioner ger en klarare bild

• tyvärr så klarar inte människan att ***”hålla hur många bollar (hattar) i luften som helst”*** förmågan att se samband och konsekvenser mellan samtidiga skeénden är begränsad. Fundera t ex på hur många tal du kan memorera om någon räknar upp dem för dig.

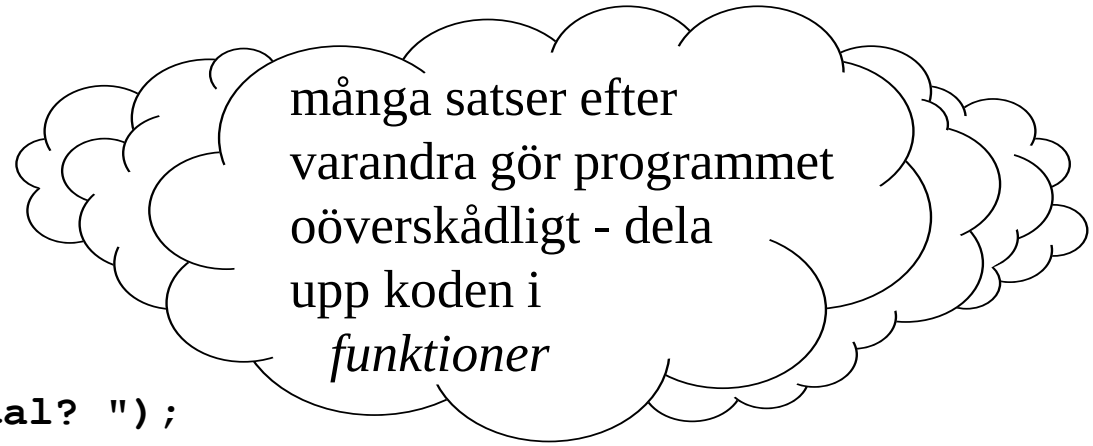
• kanske klarar du 7 st men ofta hamnar man på 3-4 st.

• med funktioner kan man dela upp sin källkod i del- (under-) program för att få överblick, se samband och konsekvens

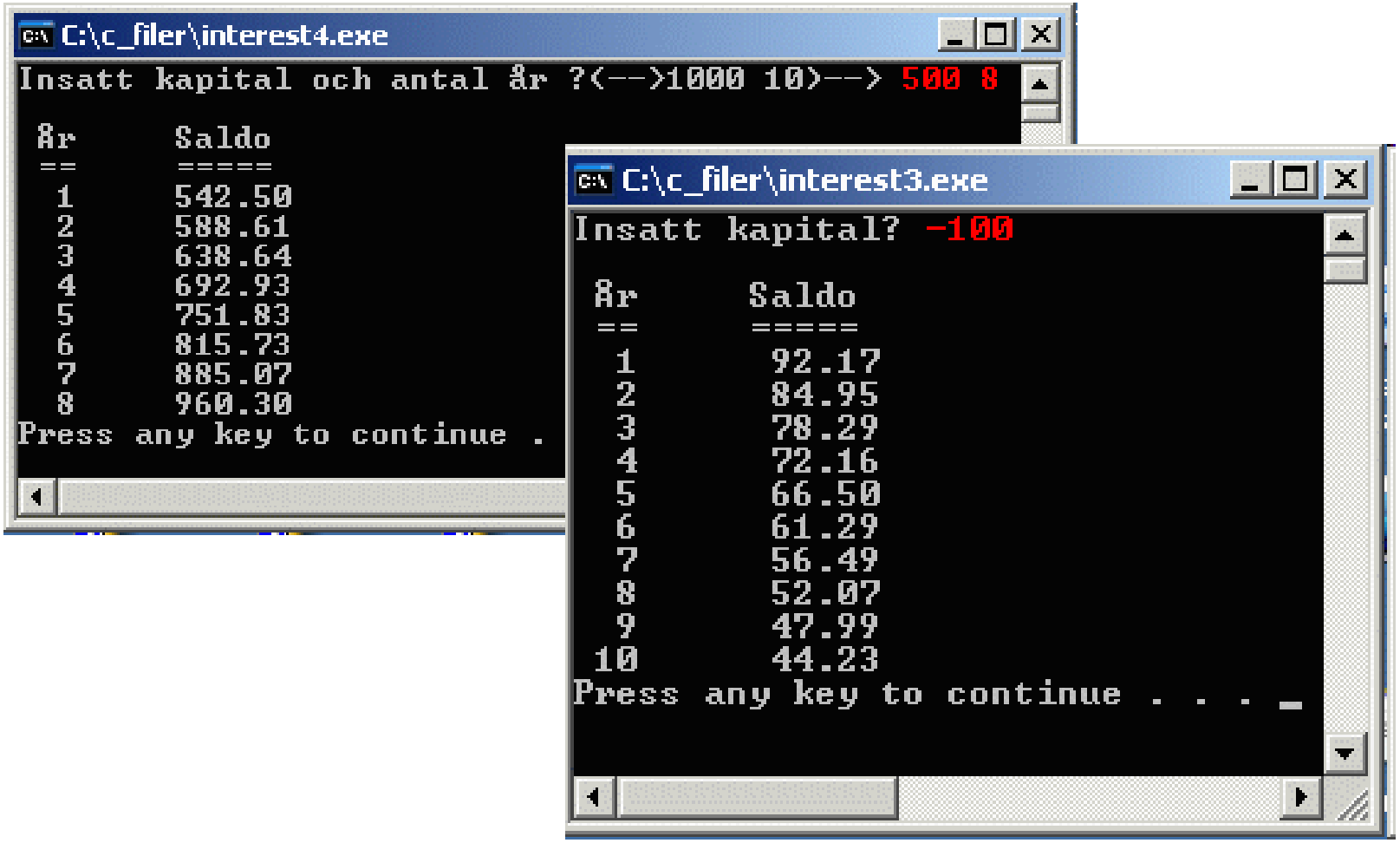
• varje delprogram (funktion) löser en avgränsad deluppgift

Ett ränteberäkningsprogram

```
/* Beräknar kapitaltillväxt på 10 år framåt eller bakåt */
#include <stdio.h>
#define RANTESATS 8.5
#define ANTAL_AR  10

int main ( void  )
{
        float   kapital ;
        int     ar ;
        printf("Insatt kapital? ");
        scanf("%f", &kapital);

        printf("\n År     Saldo\n ==     =====\n");
        for ( ar = 1; ar <= ANTAL_AR ; ar++ ) {
                if ( kapital > 0 )
                        kapital = kapital * ( 1 + RANTESATS/100 );
                else    kapital = kapital * 1/( 1 + RANTESATS/100 );
                printf("%3d%11.2f\n", ar,kapital>0 ? kapital:-kapital);
        }
        system("PAUSE");
        return 0;
}
```



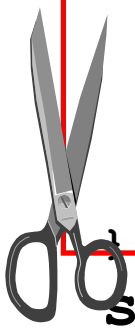
Ett ränteberäkningsprogram



Att konstruera en *funktion*

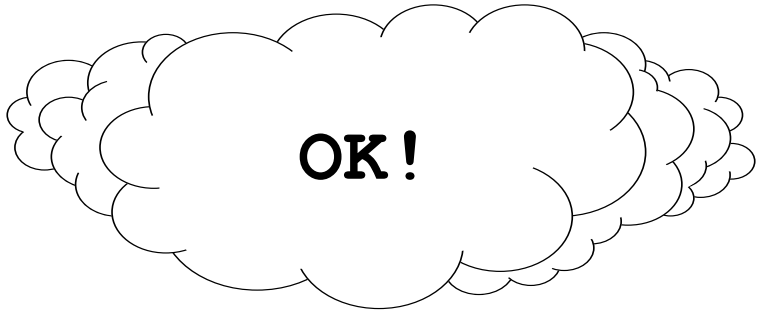
Klipp ut!

```
/* Beräknar kapitaltillväxt på 10 år framåt eller bakåt */
#include <stdio.h>
#define RANTESATS 8.5
#define ANTAL_AR  10

int main ( void  )
{
        float   kapital ;
        int     ar ;
        printf("Insatt kapital? ");
        scanf("%f", &kapital);

        printf("\n År     Saldo\n ==     =====\n");
        for ( ar = 1; ar <= ANTAL_AR ; ar++ ) {
                if ( kapital > 0 )
                        kapital = kapital * ( 1 + RANTESATS/100 );
                else    kapital = kapital * 1/( 1 + RANTESATS/100 );
                printf("%3d%11.2f\n", ar,kapital>0 ? kapital:-kapital);
        }
        system("PAUSE");
        return 0;
}
```

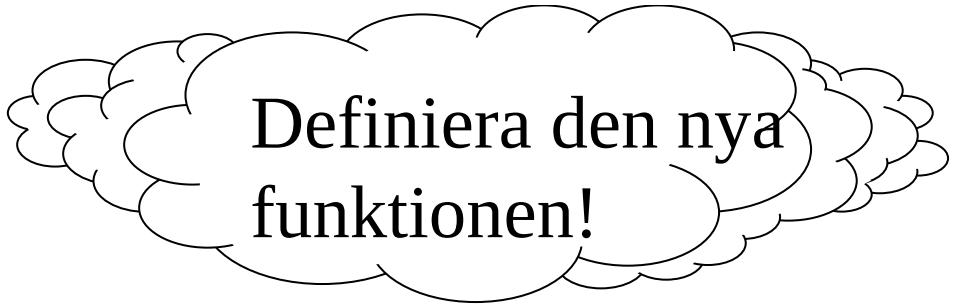


Kvar i main …

```
/* Beräknar kapitaltillväxt på 10 år framåt eller bakåt */
#include <stdio.h>
#define RANTESATS 8.5
#define ANTAL_AR  10

int main ( void  )
{
        float   kapital ;
        int     ar ;
        printf("Insatt kapital? ");
        scanf("%f", &kapital);

        system("PAUSE");
        return 0;
}
```



Klistra in!




```
printf("\n År     Saldo\n ==     =====\n");
for ( ar = 1; ar <= ANTAL_AR ; ar++ ) {
        if ( kapital > 0 )
                kapital = kapital * ( 1 + RANTESATS/100 );
        else    kapital = kapital * 1/( 1 + RANTESATS/100 );
        printf("%3d%11.2f\n", ar,kapital>0 ? kapital:-kapital);
}
```

Definiera den nya funktionen.


```
void TabellPaSkarmen( int antalAr , float kapital )
{
        int ar ;

        printf("\n År     Saldo\n ==     =====\n");
        for ( ar = 1; ar <= antalAr; ar++ ) {
                if ( kapital > 0 )
                        kapital = kapital * ( 1 + RANTESATS/100 );
                else    kapital = kapital * 1/( 1 + RANTESATS/100 );
                printf("%3d%11.2f\n", ar,kapital>0 ? kapital:-kapital);
        }

        return;
}
```

Funktionens delar

funktionens returtyp - utdatatyp

Definiera den nya funktionen!

funktionsprototyp, funktionshuvud

```
void TabellPaSkarmen( int antalAr, float kapital )
{
    int ar;
    . . .
    return;
}
```

funktionens namn

funktionens formella parametrar (argument) - indata

Funktionens delar

funktionens egen variabel, okänd utanför funktions-kroppen

Definiera den nya funktionen!

```
void TabellPaSkarmen( int antalAr , float kapital )
{
        int ar ;

        printf("\n År     Saldo\n ==     =====\n");
        for ( ar = 1; ar <= antalAr; ar++ ) {
                if ( kapital > 0 )
                        kapital = kapital * ( 1 + RANTESATS/100 );
                else    kapital = kapital * 1/( 1 + RANTESATS/100 );
                printf("%3d%11.2f\n", ar,kapital>0 ? kapital:-kapital);
        }

        return;
}
```

indatavariablerna, som bara är kända i funktionen används i funktionen

Funktionsdeklaration och funktionsanrop

```
/* Beräknar kapitaltillväxt på 10 år framåt eller bakåt */
#include <stdio.h>
#define RANTESATS 8.5
#define ANTAL_AR  10

void TabellPaSkarmen( int , float );

int main ( void  )
{
        float   kapital ;
        int     ar ;
        printf("Insatt kapital? ");
        scanf("%f", &kapital);
        TabellPaSkarmen( ANTAL_AR, kapital );

        return 0;
}
```

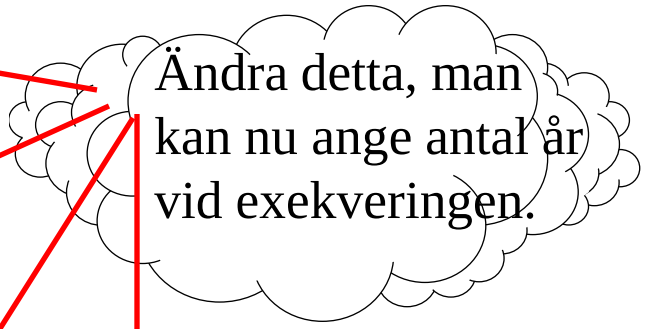
i stommen,
`main()` - funktionen

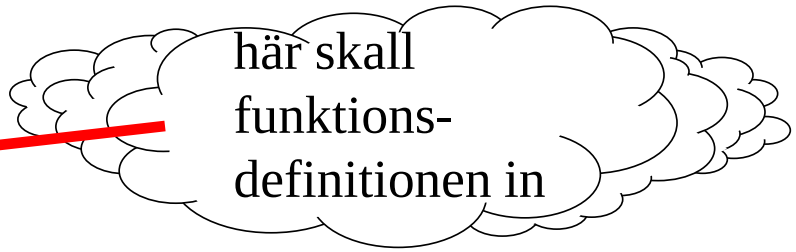
lägg till funktions-*deklaration* och *funktions-anrop*

här skall funktions-*definitionen* in

Parameter i stället för konstant

```
/* Beräknar kapitaltillväxt på x år framåt eller bakåt */
#include <stdio.h>
#define RANTESATS 8.5
#define ANTAL_AR  10

void TabellPaSkarmen( int , float );

int main ( void  )
{
        float   kapital ;
        int     ar , antalAr;
        printf("Insatt kapital och antal år ?(-->1000 10)--> ");
        scanf("%f%d", &kapital, &antalAr);
        TabellPaSkarmen( antalAr, kapital );

        return 0;
}
```





Översikt – hela programmet

```
/* Beräknar kapitaltillväxt på 10 år framåt eller bakåt */
#include <stdio.h>
#define RANTESATS 8.5

void TabellPaSkarmen( int , float );
```

stommen

```
int main ( void  )
{
            float       kapital ;
            int         antalAr;
            printf("Insatt kapital och antal år ?(-->1000 10)--> ");
            scanf("%f%d", &kapital, &antalAr);
            TabellPaSkarmen( antalAr, kapital );

            return 0;
}
```

en översiktsbild av hela programmet

```
void TabellPaSkarmen( int antalAr , float kapital )
{
            int ar ;

            printf("\n År      Saldo\n ==      =====\n");
            for ( ar = 1; ar <= antalAr; ar++ ) {
                        if ( kapital > 0 )
                                    kapital = kapital * ( 1 + RANTESATS/100 );
                        else        kapital = kapital * 1/( 1 + RANTESATS/100 );
                        printf("%3d%11.2f\n", ar,kapital>0 ? kapital:-kapital);
            }
            system(”PAUSE”);
            return;
}
```

funktionen

Översikt, funktionsdeklaration och funktionsd

```
/* Beräknar kapitaltillväxt på 10 år framåt eller bakåt */
#include <stdio.h>
#define RANTESATS 8.5

void TabellPaSkarmen( int , float );

int main ( void  )
{
          float      kapital ;
          int        antalAr;
          printf("Insatt kapital och antal år
          scanf("%f%d", &kapital, &antalAr);
          TabellPaSkarmen( antalAr, kapital );

          return 0;
}
```

en översiktsbild av hela programmet

funktionsdeklaration, för att kompilatorn skall kunna kontrollera att man använder funktionen rätt

```
void TabellPaSkarmen( int antalAr , float kapital )
{
          int ar ;

          printf("\n År      Saldo\n ==      =====\n");
          for ( ar = 1; ar <= antalAr; ar++ ) {
                    if ( kapital > 0 )
                              kapital = kapital * ( 1 + RANTESATS/100 );
                    else      kapital = kapital * 1/( 1 + RANTESATS/100 );
                    printf("%3d%11.2f\n", ar,kapital>0 ? kapital:-kapital);
          }

          return;
}
```

funktionsdefinition

En hatt i taget …

Å va bra!
Nu tar jag en hatt åt gången!

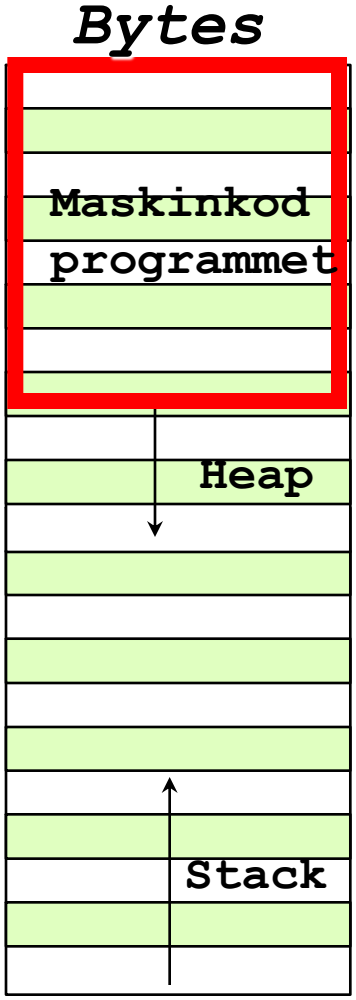
```
/* Beräknar kapitaltillväxt på 10 år framåt eller bakåt */
#include <stdio.h>
#define RANTESATS 8.5

void TabellPaSkarmen( int , float );

int main ( void  )
{
            float       kapital ;
            int         antalAr;
            printf("Insatt kapital och antal år ?(-->1000 10)--> ");
            scanf("%f%d", &kapital, &antalAr);
            TabellPaSkarmen( antalAr, kapital );

            return 0;
}

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

            printf("\n År      Saldo\r
            for ( ar = 1; ar <= antal
                        if ( kapital >
                                    kap
                        else        kap
                        printf("%3d%11
            }

            return;
}
```

```
C:\c_filer\interest4.exe
Insatt kapital och antal år ?(-->1000 10)--> 500 8

 År      Saldo
 ==      =====
  1      542.50
  2      588.61
  3      638.64
  4      692.93
  5      751.83
  6      815.73
  7      885.07
  8      960.30
Press any key to continue . . .
```

Funktionsanrop

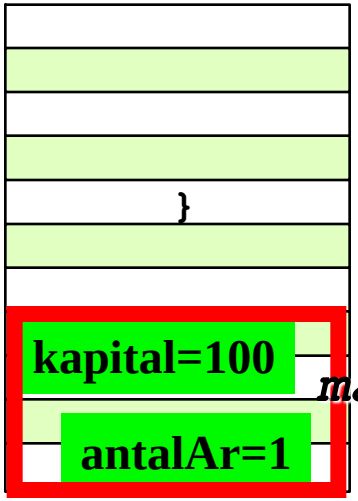
- Hur anropas en funktion?
- Vad händer då en funktion anropas och exekveras?

Main's aktiveringspost på "stacken"




```
int main ( void  )
{
   float   kapital ;
   int     antalAr;
   printf("Insatt kapital och antal år ? ");
   printf("(-->1000 10)--> ");
   scanf("%f%d", &kapital, &antalAr);
   TabellPaSkarmen( antalAr, kapital );

   return 0;
}
```

main() exekveras




```
(Inactive C:\OS2\MDOS\WINOS2\TEMP\"PKG001F.EXE)
Insatt kapital och antal år ?(-->1000 10)--> 100 1

År      Saldo
==      =====
 1      108.50
```

main's "egna" variabler skapas på stacken - ***aktiveringspost***

tid

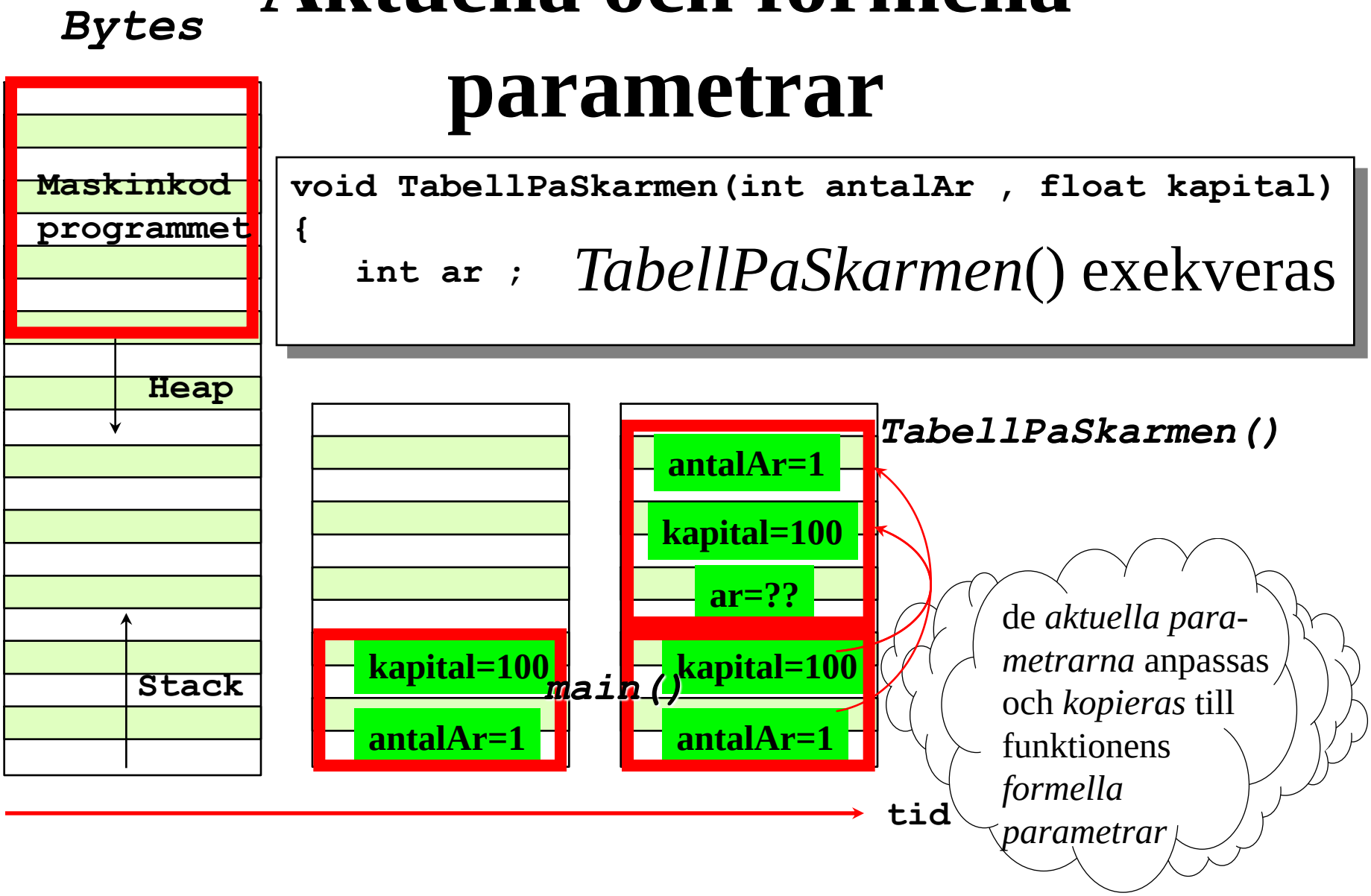
TabellPaSkarmen()

```
void TabellPaSkarmen( int antalAr , float kapital )
{
    int ar ;

    printf("\n År
    for ( ar = 1; ar
    if ( kapital > (
      kapital = kapi
    else kapital = k
      printf("%3d%11
    }

    return;
}
```

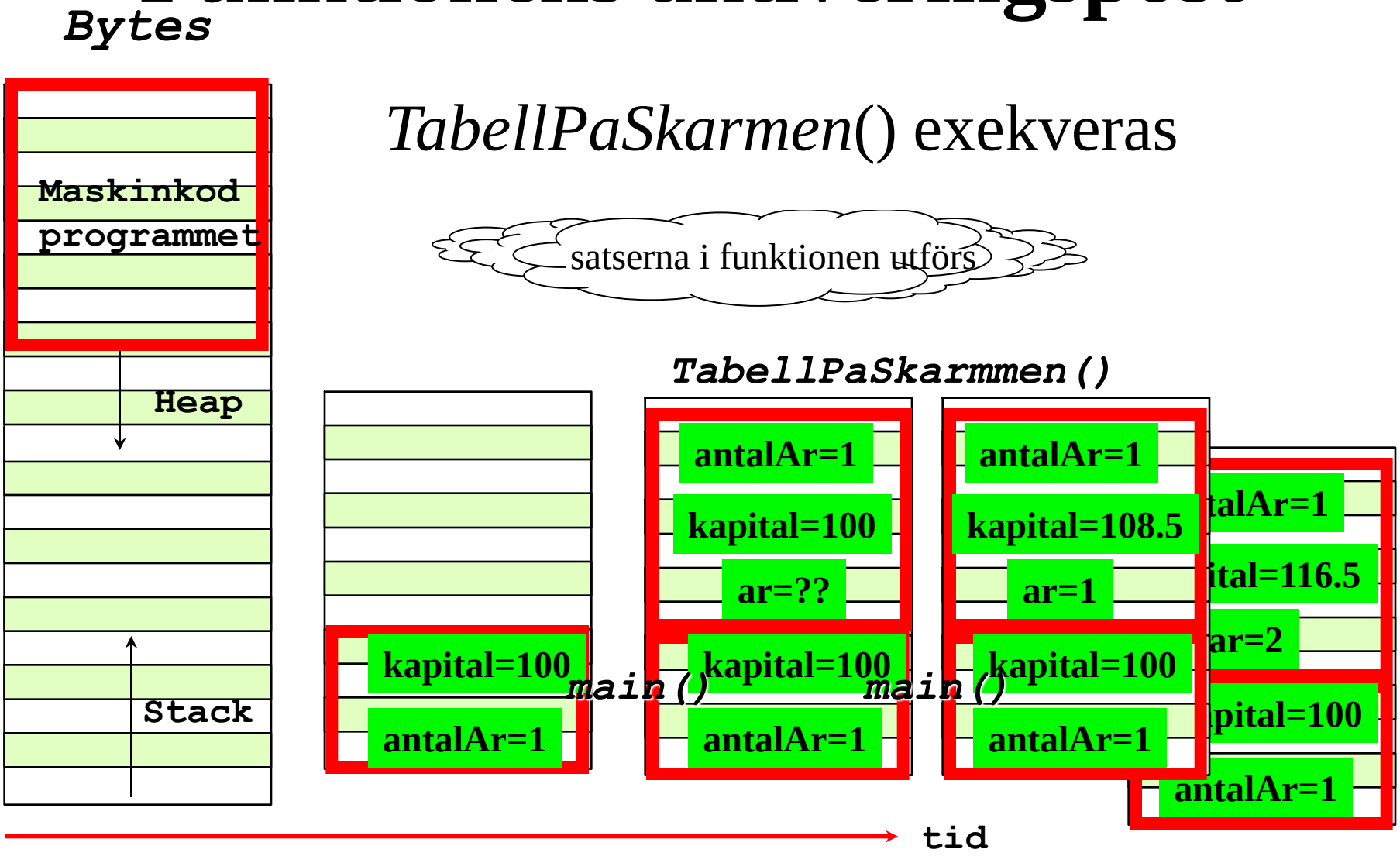
Kan heta samma, men behöver inte heta samma!

```
int main ( void  )
{
   float  kapital ;
   int  antalAr;
   printf("Insatt kapital och antal år ? ");
   printf("(-->1000 10)--> ");
   scanf("%f%d", &kapital, &antalAr);
   TabellPaSkarmen( antalAr, kapital );

   return 0;
}
```

Aktuella och formella parametrar



Funktionens aktiveringspost

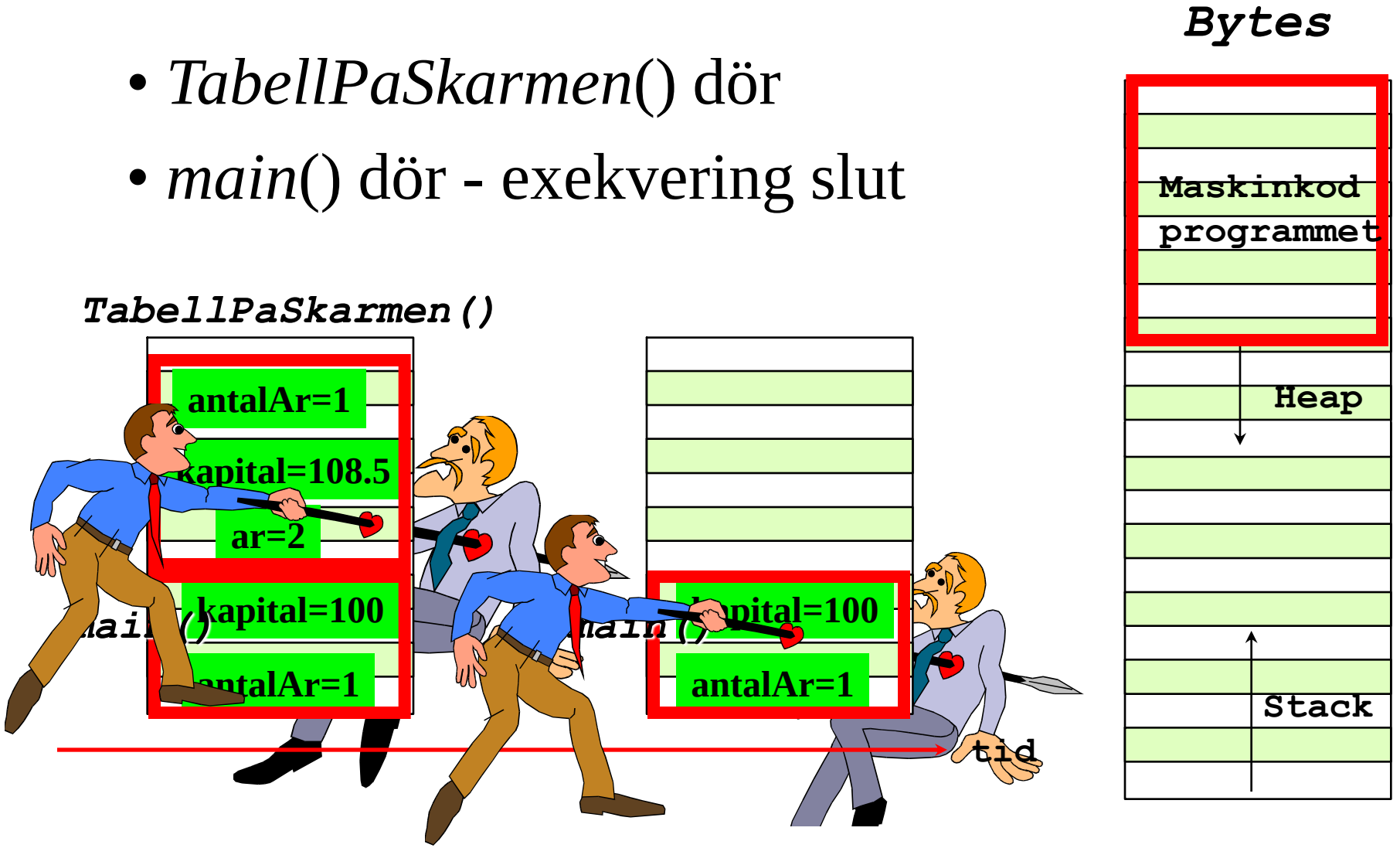


TabellPaSkarmen() exekveras

satserna i funktionen utförs

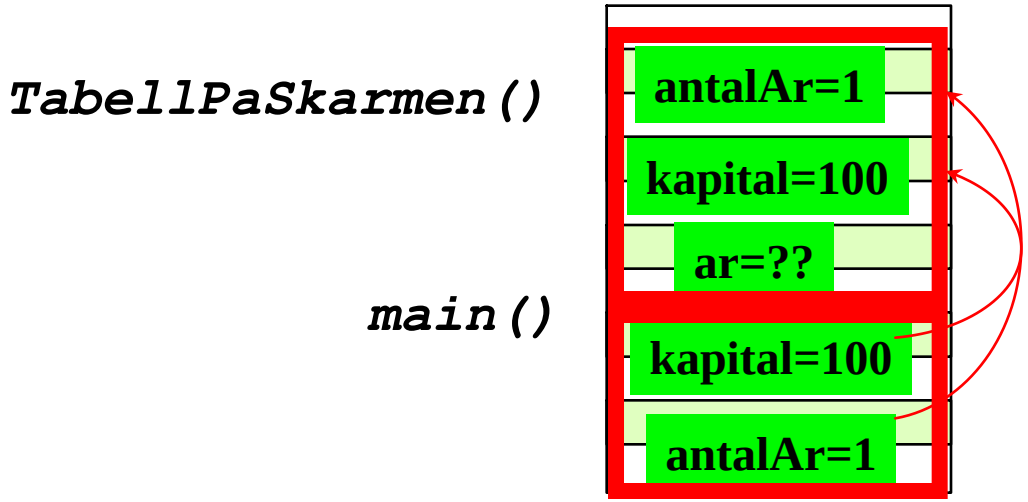
Aktiveringsposterna tas bort

- *TabellPaSkarmen*() dör
- *main*() dör - exekvering slut



Värdeöverföring

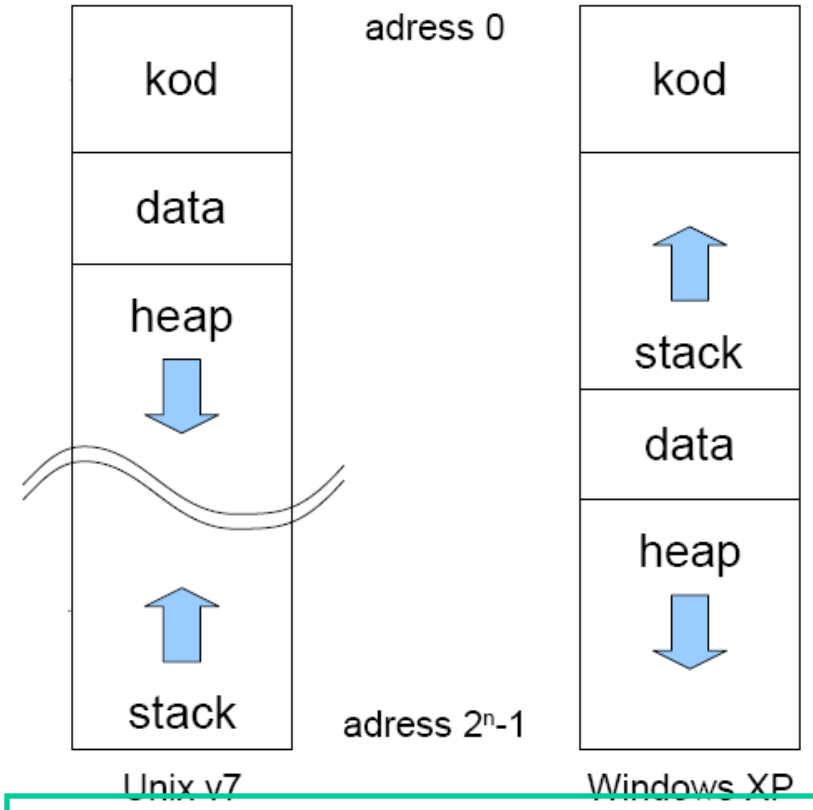
- Talen *1* och *100* har värdeöverförts, kopierats in i funktionen *TabellPaSkarmen*()
- Att nu förändra kopian ändrar *inte* på originalet (originalen är skyddade)



C:s minneskarta

- **Kod-segmentet** innehåller programmets körbara kod
- **Data-segmentet** innehåller variabler som är globala samt programkonstanter
- **Stack-segmentet** innehåller Automatiska variabler. Funktionernas parametrar, returvärden och lokala variabler.
- **Heap-segmentet** innehåller dynamiska variabler skapade under programkörningen.

Begreppen Stack och Heap (två olika ”högar”) tillhör dataområdets allmänbildning!



Se **C_stack_heap.ppt** i kursen ID120V !

C:s minnesmodell – var hamnar variabler?

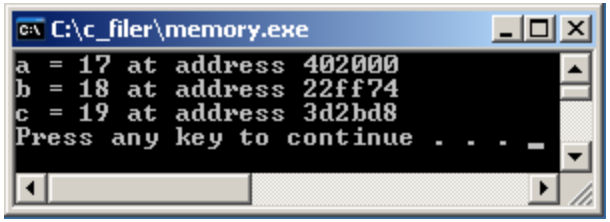
```
#include <stdio.h>
int a = 17;
int main(void)
{
  int b = 18;
  int * c;
  c = malloc( sizeof( int ) );
  *c = 19;
  printf("a = %d at address %x\n", a, &a);
  printf("b = %d at address %x\n", b, &b);
  printf("c = %d at address %x\n", *c, c);
  free(c);
  system("PAUSE");
  return 0;
}
```

Global variabel = Data segmentet

Automatisk variabel = Stack

Dynamisk variabel = Heap

Återställ dynamiskt minne!



William Sandqvist william@kth.se

Data-segmentet

”Globala” variabler, dvs variabler som deklarerats *utanför* programmets funktioner hamnar i datasegmentet.

Initialiserade variabler förses med sina startvärden.
Oinitialiserade variabler placeras för sig (`.bss`), de kan då enkelt åtminstone 0-ställas.

I datasegmentet kan också konstanter och strängkonstanter lagras.

Datasegmentet är *statiskt*, variablerna upptar plats under hela programkörningen både när de används och när de inte används.

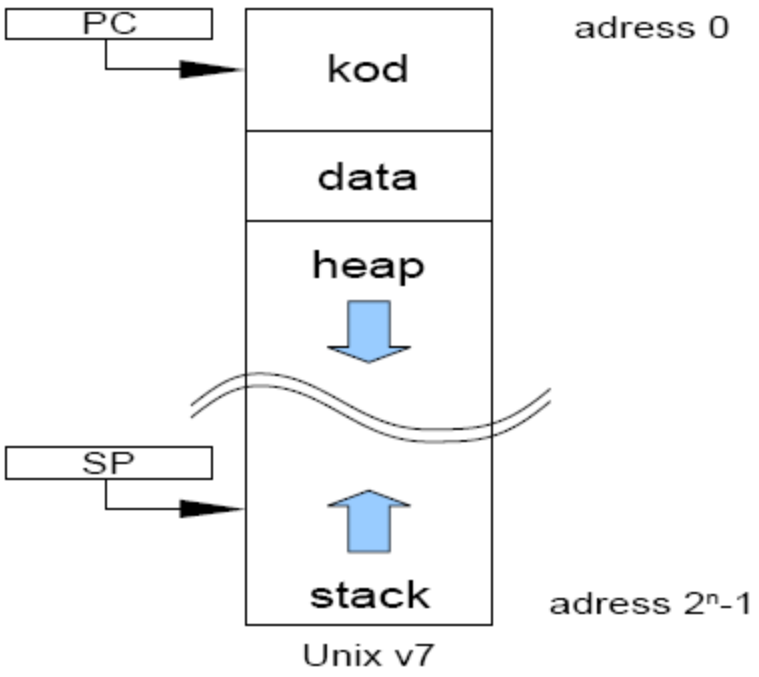
Nackdelen med de globala variablerna är att de kan nås från alla funktioner. De kan vara både parametrar och returvärden på en och samma gång för alla funktioner som använder dem – på ett oöverskådligt sätt – som gjort för problem!

Stacken

När en funktion anropas så skapas en **aktivitetspost** på stacken. Den består av utrymme för funktionens parametrar och returvärde, samt de variabler som deklareras inuti funktionen (= lokala variabler).

För en registerrik processor så ligger en del parametrar och returvärdet i register.

När en funktion avslutas så tas aktivitetsposten bort från stacken.

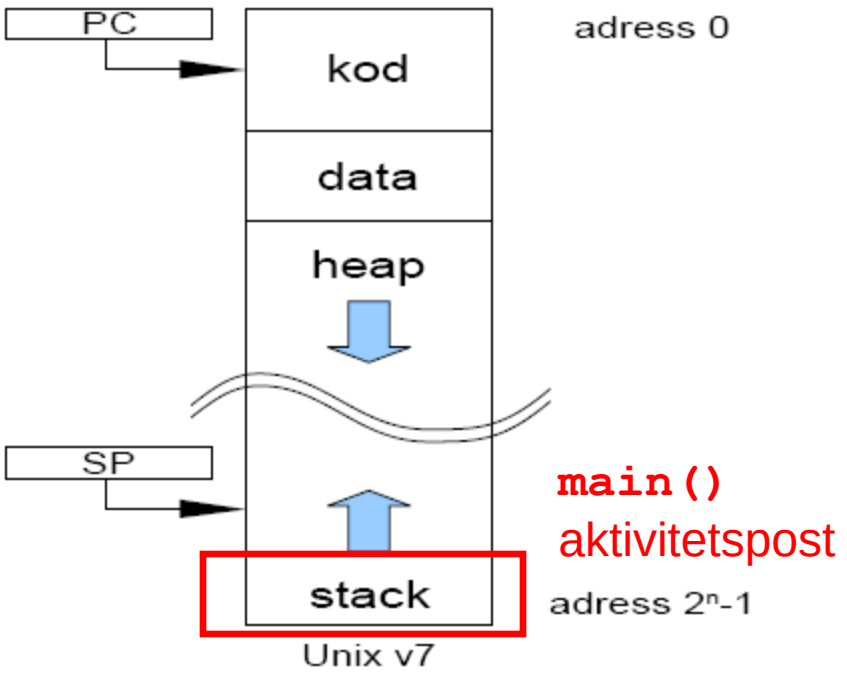


Stacken

När en funktion anropas så skapas en **aktivitetspost** på stacken. Den består av utrymme för funktionens parametrar och returvärde, samt de variabler som deklareras inuti funktionen (= lokala variabler).

För en registerrik processor så ligger en del parametrar och returvärdet i register.

När en funktion avslutas så tas aktivitetsposten bort från stacken.

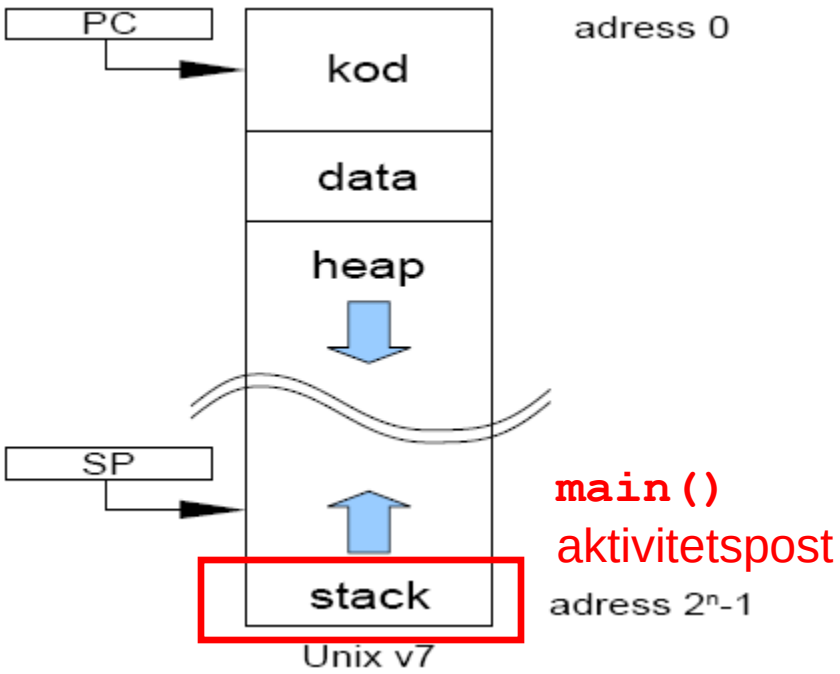


Funktionen `main()` är speciell genom att den avslutas först när hela programmet kört klart. Dess aktivitetspost ligger kvar under programkörningen.

William Sandqvist william@kth.se

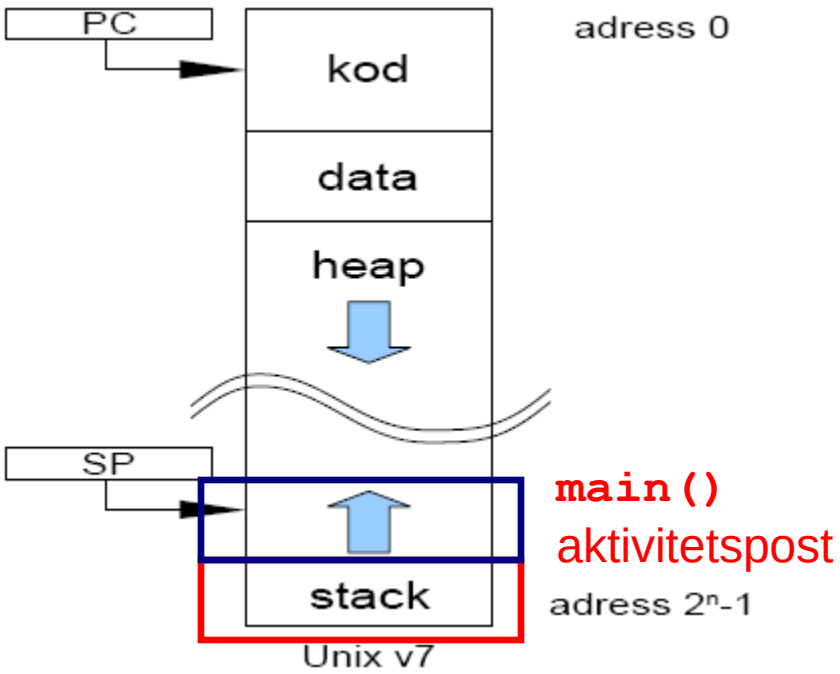
Automatiska variabler

`main()` anropar funktionen `funk1()`. En ny aktivitetspost skapas nu på stacken för den funktionens variabler.



Automatiska variabler

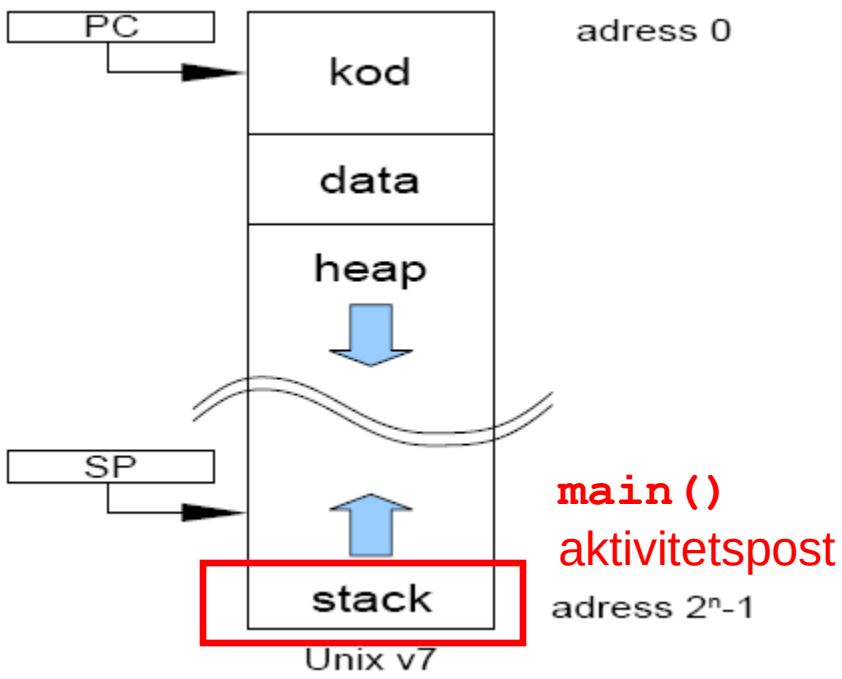
`main()` anropar funktionen `funk1()`. En ny aktivitetspost skapas nu på stacken för den funktionens variabler.



Automatiska variabler

`main()` anropar funktionen `funk1()`. En ny aktivitetspost skapas nu på stacken för den funktionens variabler.

När funktionen kört klart returnerar den till `main()` och aktivitetposten tas bort.

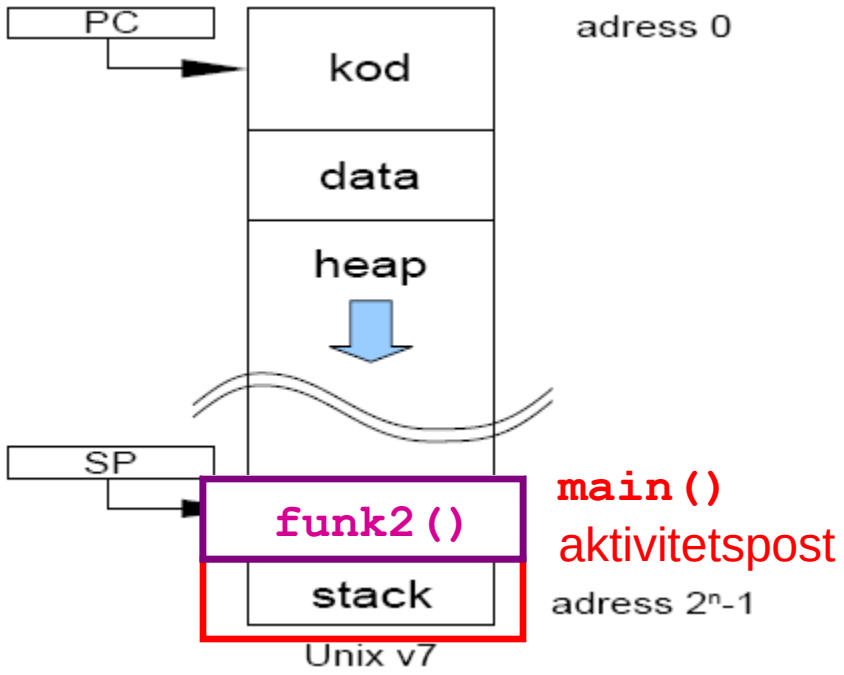


Automatiska variabler

`main()` anropar funktionen `funk1()`. En ny aktivitetspost skapas nu på stacken för den funktionens variabler.

När funktionen kört klart returnerar den till `main()` och aktivitetposten tas bort.

Nu anropas en annan funktion `funk2()`. Den funktionens aktivitetspost kan nu utnyttja *samma* minne på stacken.

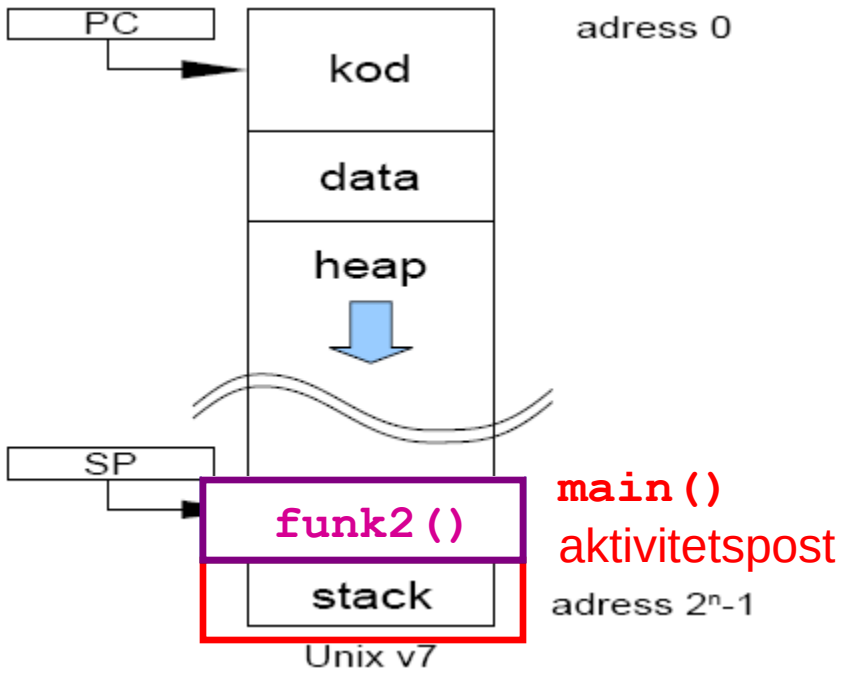


Automatiska variabler

`main()` anropar funktionen `funk1()`. En ny aktivitetspost skapas nu på stacken för den funktionens variabler.

När funktionen kört klart returnerar den till `main()` och aktivitetposten tas bort.

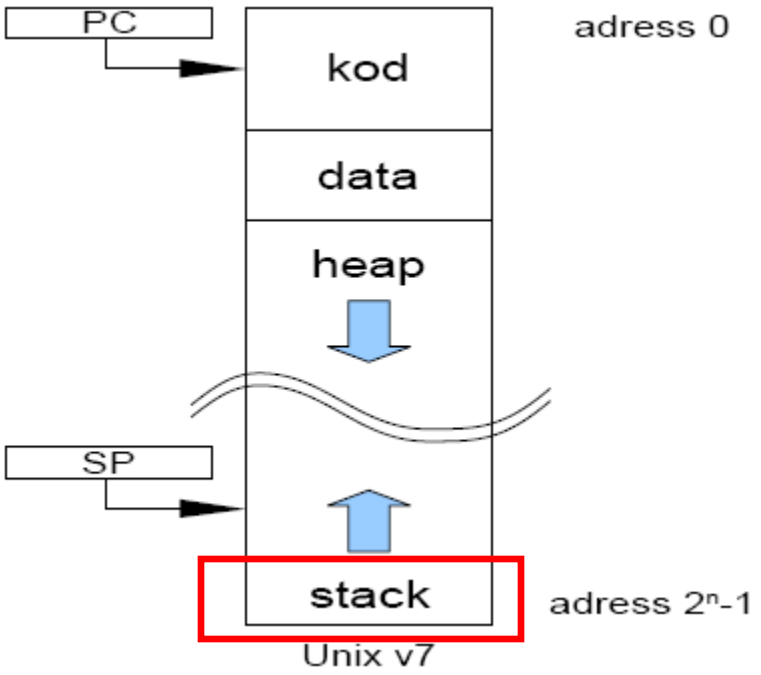
Nu anropas en annan funktion `funk2()`. Den funktionens aktivitetspost kan nu utnyttja *samma* minne på stacken.



Funktionernas variabler på stacken kallas för **automatiska variabler**. Minnesutrymmet på stacken återanvänds av funktionerna och utnyttjas därmed effektivt.

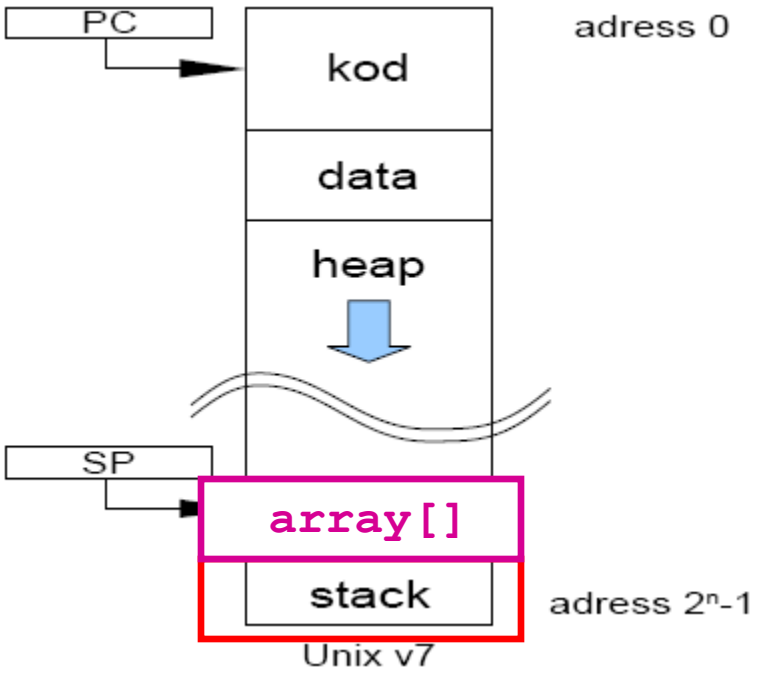
En funktion som genererar data

Antag att funktionen **`funk2()`** genererar data och lagrar dessa i en lokal **`array[]`** på stacken.



En funktion som genererar data

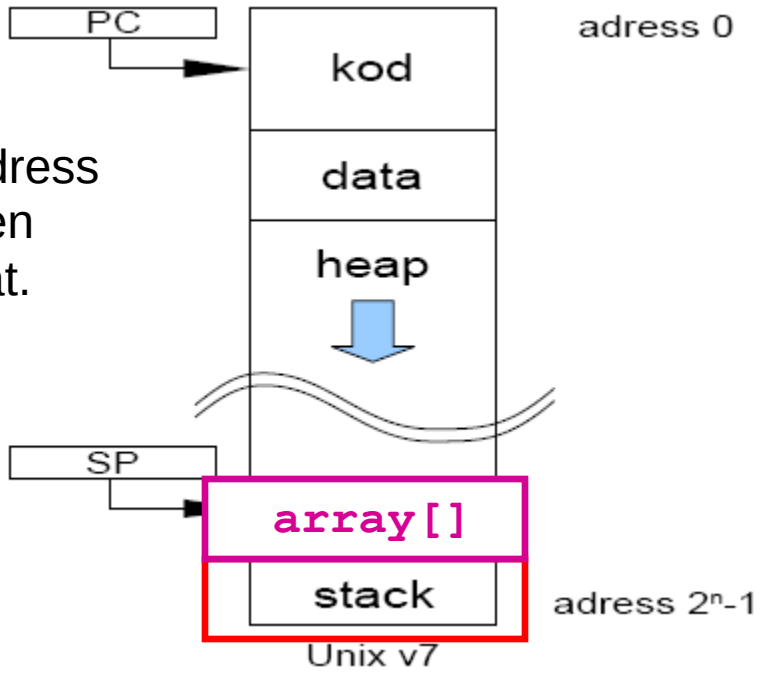
Antag att funktionen **`funk2()`** genererar data och lagrar dessa i en lokal **`array[]`** på stacken.



En funktion som genererar data

Antag att funktionen **`funk2()`** genererar data och lagrar dessa i en lokal **`array[]`** på stacken.

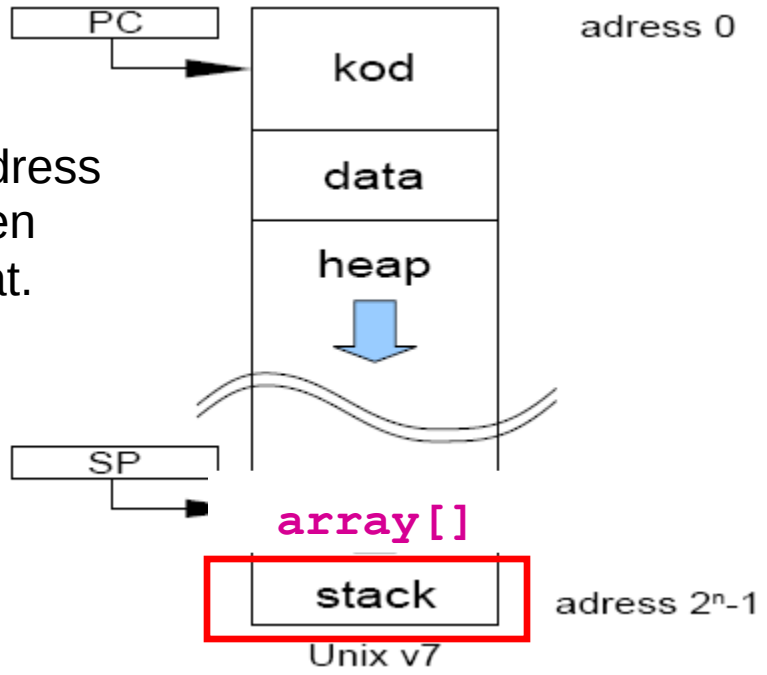
Funktionen returnerar arrayens adress till **`main()`**, med hjälp av adressen kan **`main()`** nå och använda datat.



En funktion som genererar data

Antag att funktionen **`funk2()`** genererar data och lagrar dessa i en lokal **`array[]`** på stacken.

Funktionen returnerar arrayens adress till **`main()`**, med hjälp av adressen kan **`main()`** nå och använda datat.

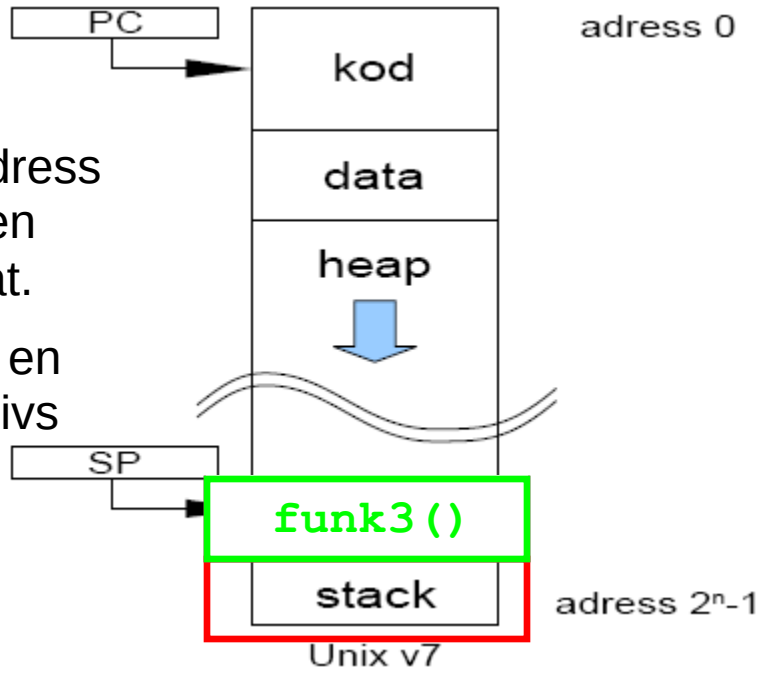


En funktion som genererar data

Antag att funktionen **`funk2()`** genererar data och lagrar dessa i en lokal **`array[]`** på stacken.

Funktionen returnerar arrayens adress till **`main()`**, med hjälp av adressen kan **`main()`** nå och använda datat.

Datat på stacken lever farligt! Om en ny funktion, **`funk3()`**, körs så skrivs datat över.



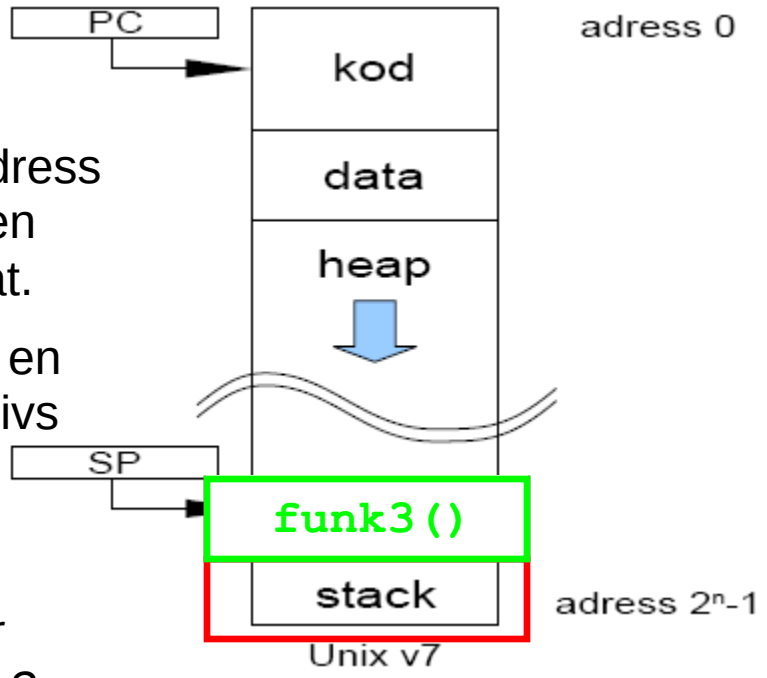
En funktion som genererar data

Antag att funktionen **`funk2()`** genererar data och lagrar dessa i en lokal **`array[]`** på stacken.

Funktionen returnerar arrayens adress till **`main()`**, med hjälp av adressen kan **`main()`** nå och använda datat.

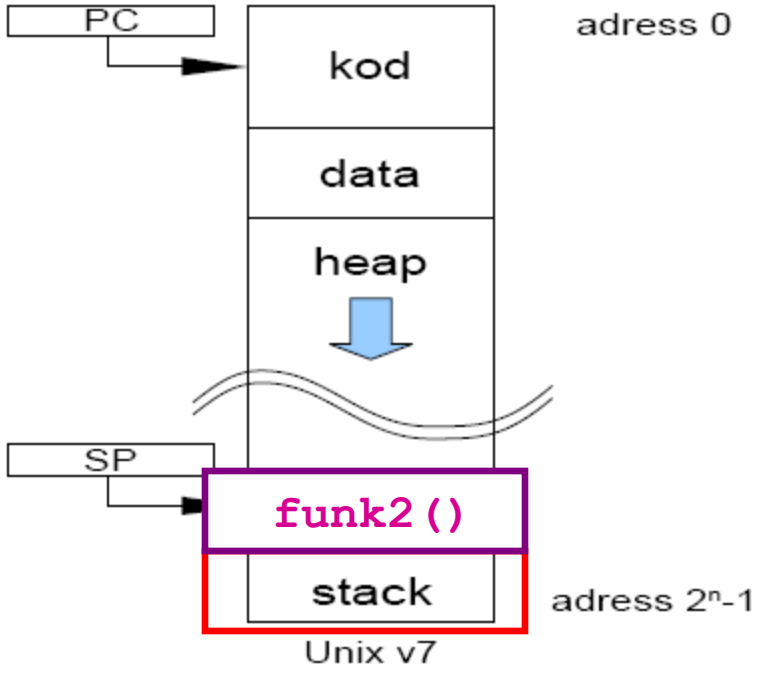
Datat på stacken lever farligt! Om en ny funktion, **`funk3()`**, körs så skrivs datat över.

Det behövs någon annan plats för datat. – Var skulle det kunna vara?



Heapen

Funktionen `funk2()` behöver lagra sina data någon annanstans.

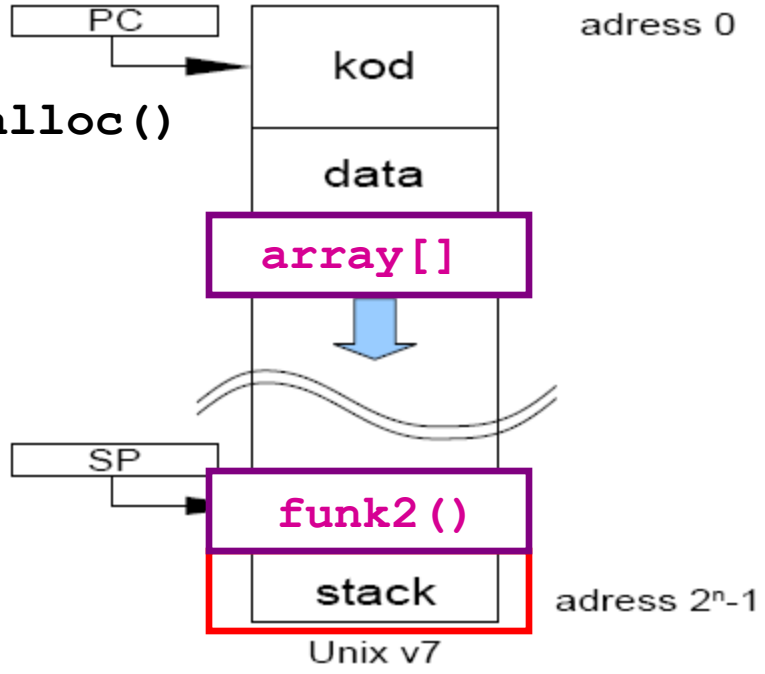


William Sandqvist william@kth.se

Heapen

Funktionen `funk2()` behöver lagra sina data någon annanstans.

Den kan anropa C:s funktion `malloc()` för att få utrymme på heapen.

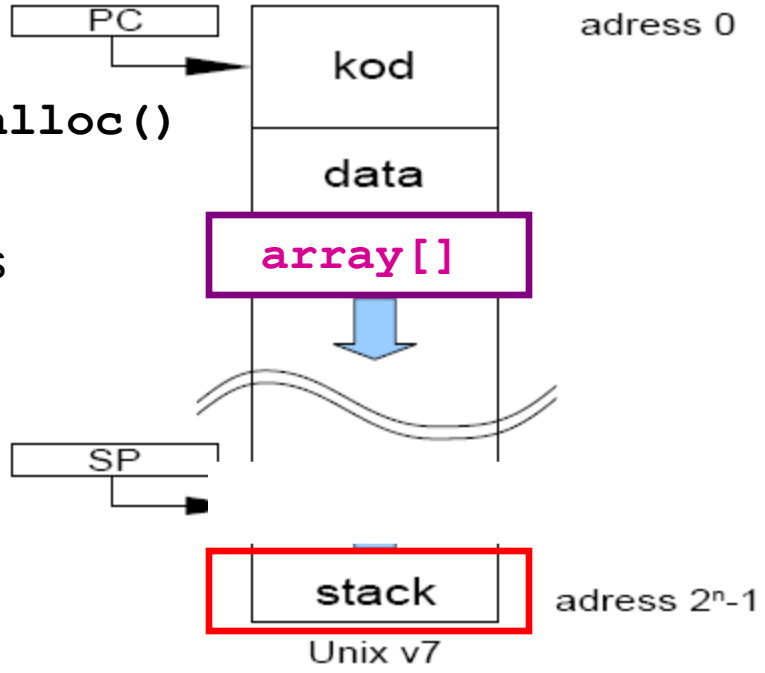


Heapen

Funktionen `funk2()` behöver lagra sina data någon annanstans.

Den kan anropa C:s funktion `malloc()` för att få utrymme på heapen.

`funk2()` returnerar nu arrayens adress till `main()`.



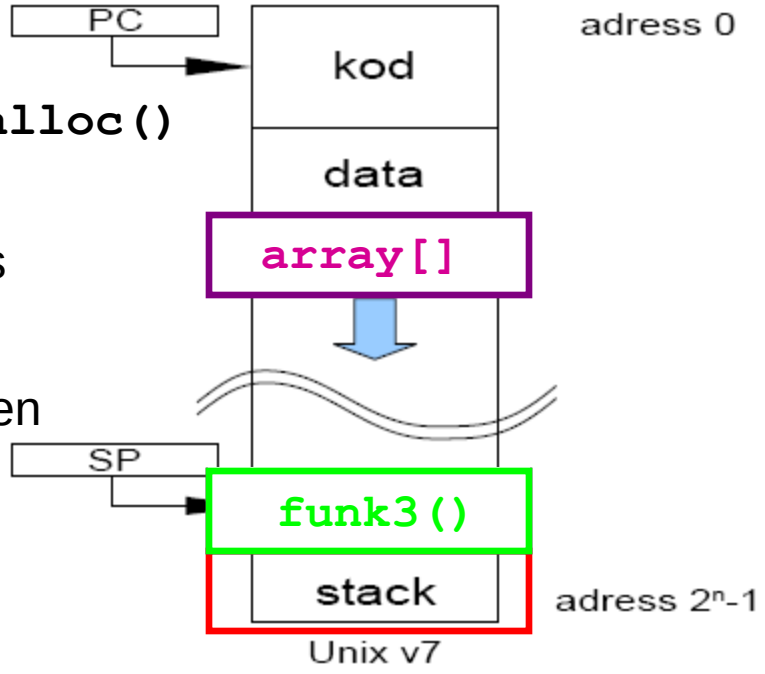
Heapen

Funktionen `funk2()` behöver lagra sina data någon annanstans.

Den kan anropa C:s funktion `malloc()` för att få utrymme på heapen.

`funk2()` returnerar nu arrayens adress till `main()`.

Datat finns fortfarande kvar om en ny funktion `funk3()` ska köra.



Heapen

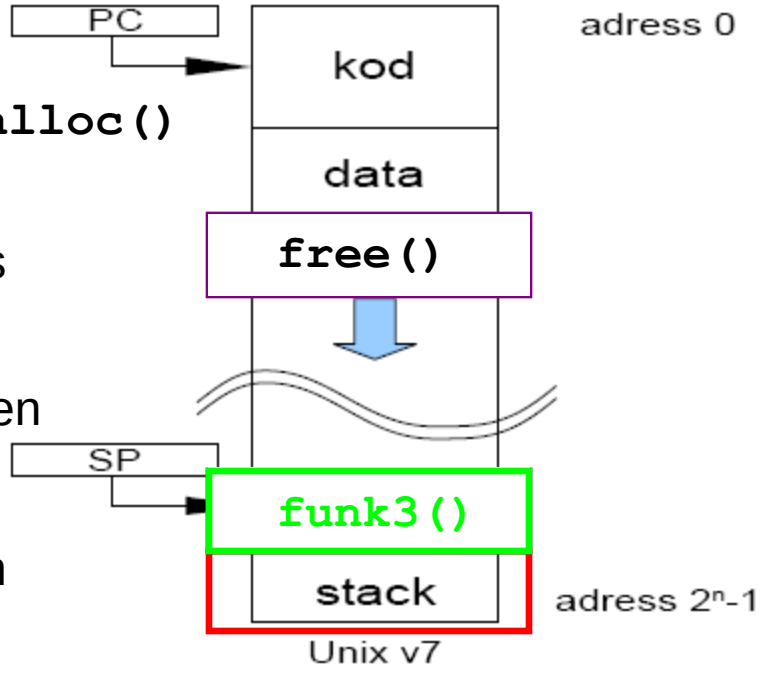
Funktionen `funk2()` behöver lagra sina data någon annanstans.

Den kan anropa C:s funktion `malloc()` för att få utrymme på heapen.

`funk2()` returnerar nu arrayens adress till `main()`.

Datat finns fortfarande kvar om en ny funktion `funk3()` ska köra.

Minnesutrymmet frigörs när man anropar funktionen `free()`.



Heapen

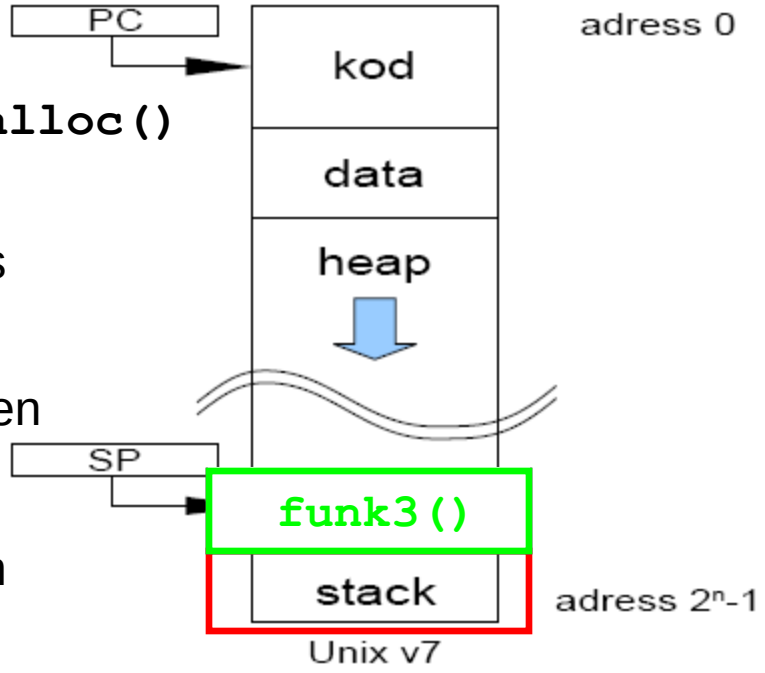
Funktionen `funk2()` behöver lagra sina data någon annanstans.

Den kan anropa C:s funktion `malloc()` för att få utrymme på heapen.

`funk2()` returnerar nu arrayens adress till `main()`.

Datat finns fortfarande kvar om en ny funktion `funk3()` ska köra.

Minnesutrymmet frigörs när man anropar funktionen `free()`.



Om C:s minneshanterare är bra skriven utnyttjas minnesutrymmet på heapen effektivt.

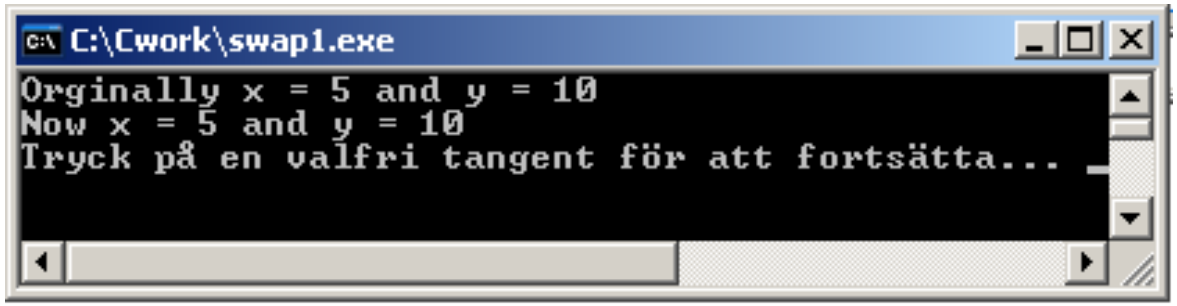
C:s parameteröverföring

En funktion som byter plats på två variabler …

Se **swap.ppt** i kursen ID120V !

Funktion som *byter plats* på två variabler

```
/* swap1.c first attempt at a swaping function          */
/* from Stephen Prata, C Primer Plus ISBN 1-571969-161-8 */
#include <stdio.h>
void interchange(int u, int v);
int main(void)
{
  int x = 5, y = 10;
  printf("Orginally x = %d and y = %d\n", x, y);
  interchange(x, y);
  printf("Now x = %d and y = %d\n", x, y);
  system("PAUSE");
  return 0;
}
void interchange(int u, int v)
{
  int temp;
  temp = u;
  u = v;
  v = temp;
}
```



William Sandqvist william@kth.se

Funktionen fungerar ej

Funktionen **`void interchange( int u, int v)`** får ju bara ***kopior*** av x och y och kan därför inte påverka orginalen talen x och y i main.

För att byta plats på två tal krävs ***två*** returvärden, och C-funktioner har som mest ***ett*** returvärde.

Pekare, Adressoperatorn `&` och avrefereringsoperatorn `*`

`int b;` Deklaration av heltalsvariabeln b. Plats reserveras.

`b = 18;` Definition av variabeln b. Nu innehåller den talet 18.

`&b` Adressoperatorn. Adressen till variabeln b.

`int * c;` Deklaration av int-pekarvariabeln c.

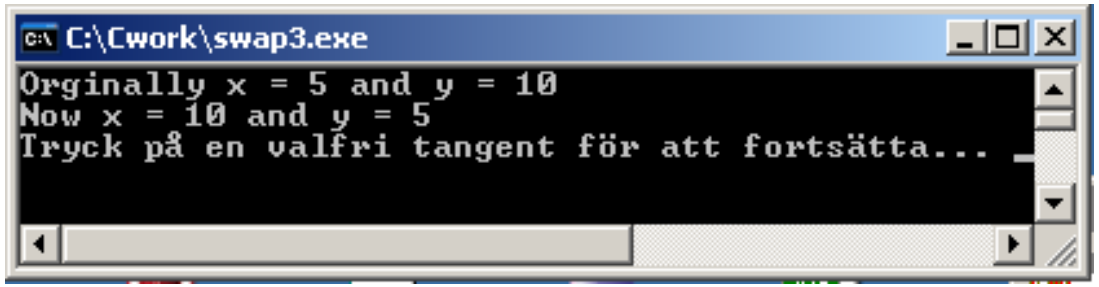
`*c` Avrefereringsoperatorn. Det som c pekar på.

`*c = 19;` Talet 19 lagras på den plats som c pekar ut.

William Sandqvist william@kth.se

Funktion som *byter plats* på två variabler

```
/* swap3.c pointers to make a swapping function work     */
/* from Stephen Prata, C Primer Plus ISBN 1-571969-161-8 */
#include <stdio.h>
void interchange(int *, int *);
int main(void)
{
  int x = 5, y = 10;
  printf("Orginally x = %d and y = %d\n", x, y);
  interchange(&x, &y);
  printf("Now x = %d and y = %d\n", x, y);
  system("PAUSE");
  return 0;
}
void interchange(int * u, int * v)
{
  int temp;
  temp = *u;
  *u =*v;
  *v = temp;
}
```



William Sandqvist william@kth.se

Nu fungerar det!

Funktionens parametrar, u och v är nu ”kopior” av adresserna till variablerna x och y (&x, &y).
Med hjälp av de medskickade adresserna kan funktionen returnera värden till x och y.

```
temp = *u;   *u = *v;  *v = temp;
```

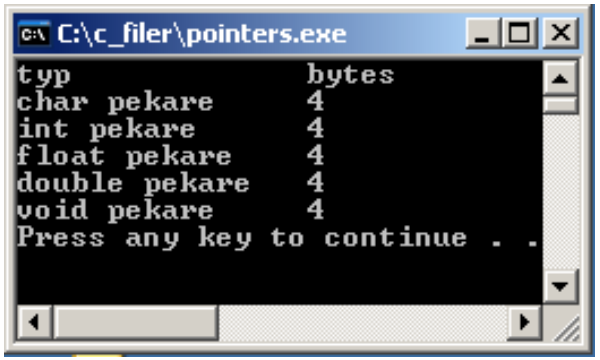
Förutom en C-funktions returvärde kan funktionen även ge retur till adresser som angivits i parameterlistan!

Hur stora är pekare?

Det får man veta genom att köra ett testprogram på sin dator:

```
#include <stdio.h>
int main(void)
{
  printf("typ\tbytes\n");
  printf("char pekare \t%d\n", sizeof(char *));
  printf("int pekare \t%d\n", sizeof(int *));
  printf("float pekare \t%d\n", sizeof(float *));
  printf("double pekare \t%d\n", sizeof(double *));
  printf("void pekare \t%d\n", sizeof(void *));

  system("PAUSE");
  return 0;
}
```

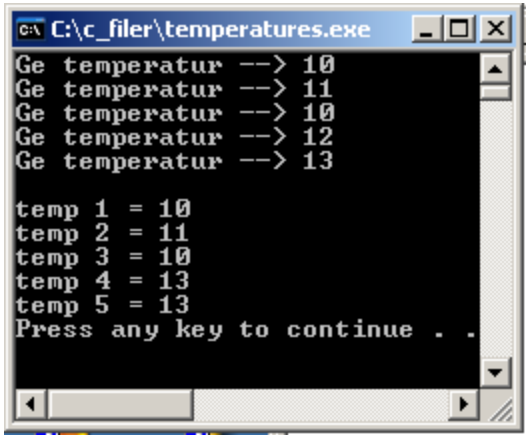
Void-pekare – pekare till vad som helst!



Array (*vektor, matris*)

ett klumpigt program …

```
#include <stdio.h>
int main(void)
{
  int temp1, temp2, temp3, temp4, temp5;
  printf("Ge temperatur --> "); scanf("%d",&temp1);
  printf("Ge temperatur --> "); scanf("%d",&temp2);
  printf("Ge temperatur --> "); scanf("%d",&temp3);
  printf("Ge temperatur --> "); scanf("%d",&temp4);
  printf("Ge temperatur --> "); scanf("%d",&temp5);
  printf("\ntemp 1 = %d", temp1);
  printf("\ntemp 2 = %d", temp2);
  printf("\ntemp 3 = %d", temp3);
  printf("\ntemp 4 = %d", temp5);
  printf("\ntemp 5 = %d\n", temp5);
  system("PAUSE");
  return 0;
}
```

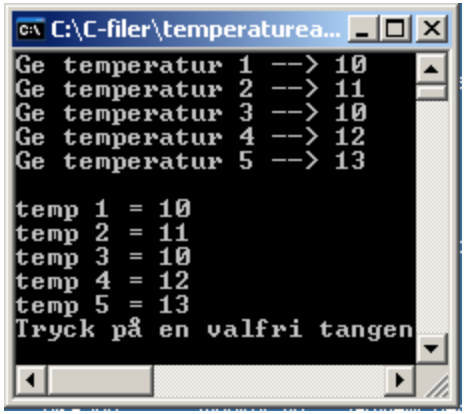


William Sandqvist william@kth.se

Vi behöver indexerade variabler!

```
#include <stdio.h>
int main(void)
{
  int temp[5];
  int i;
  for(i=1 ; i<5 ; i++)
   {
     printf("Ge temperatur %d --> ",i); scanf("%d",&temp[i-1]);
   }
  for (i=1 ; i<5 ; i++)
     printf("\ntemp %d = %d",i, temp[i+1]);
  printf("\n");
  system("PAUSE");
  return 0;
}
```

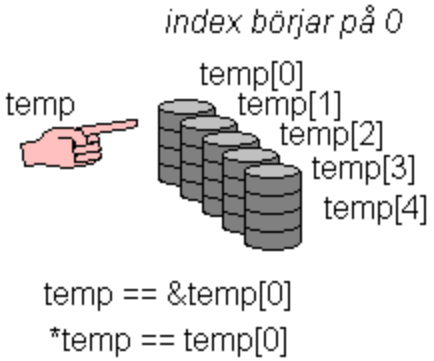
En array deklareras!



William Sandqvist william@kth.se

Beteckningar

```
int temp[5];
```



Namnet på arrayen är samma sak som adressen till arrayen!

Adress, *konstant* pekare 

Pekare, minnesutrymme som kan lagra adresser 

Minnesutrymme, variabel 

Index- och pekarstegning

Indexstegning:

```
for (i=0 ; i<5 ; i++)
  printf("%d\t%d\n",i,temp[i] );
```

Pekarstegning:

```
int *  Pek = temp;
for (i=0 ; i<5 ; i++)
  printf("%d\t%d\n",i ,*Pek++ );
```

Pekarstegning är i allmänhet snabbare – därför kommer kompilatorn att använda pekarstegning även om Du programmerar indexstegning!

William Sandqvist william@kth.se

C character och string

En **`char`** är en 8-bitars heltalsvariabel. Namnet antyder att den kan lagra en bokstav, men en **`unsigned char`** kan lika gärna användas till att hålla reda på heltal upp till 255, och en **`signed char`** heltal mellan -128 . . . +127.

Kompilatorn ”slår upp” koden i ASCII-tabellen.

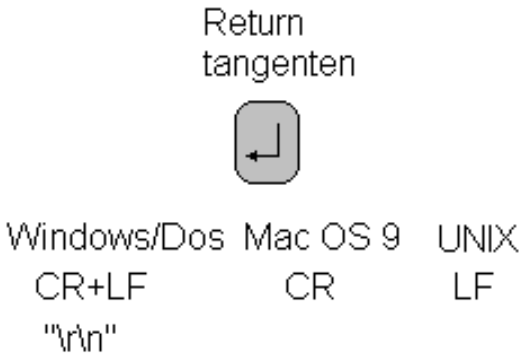
```
char tecken = 'W';
```

7-bitars ASCII-tabell

	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

Ett tecken lagras i en Byte.

Hej!
48 65 6A 21
01001000 01100101 01101010 00100001



http://ascii-table.com/

Strängar, char array

En sträng är en char array som avslutats med strängslutstecknet `'\0'`. ASCII nr 0.

```
char str[]={'H','e','l','l','o',' ','w','o','r','l','d','!','\0'};
```

Eller mer ”rakt på sak”:

```
char str[]=”Hello world!”;
```

OBSERVERA!

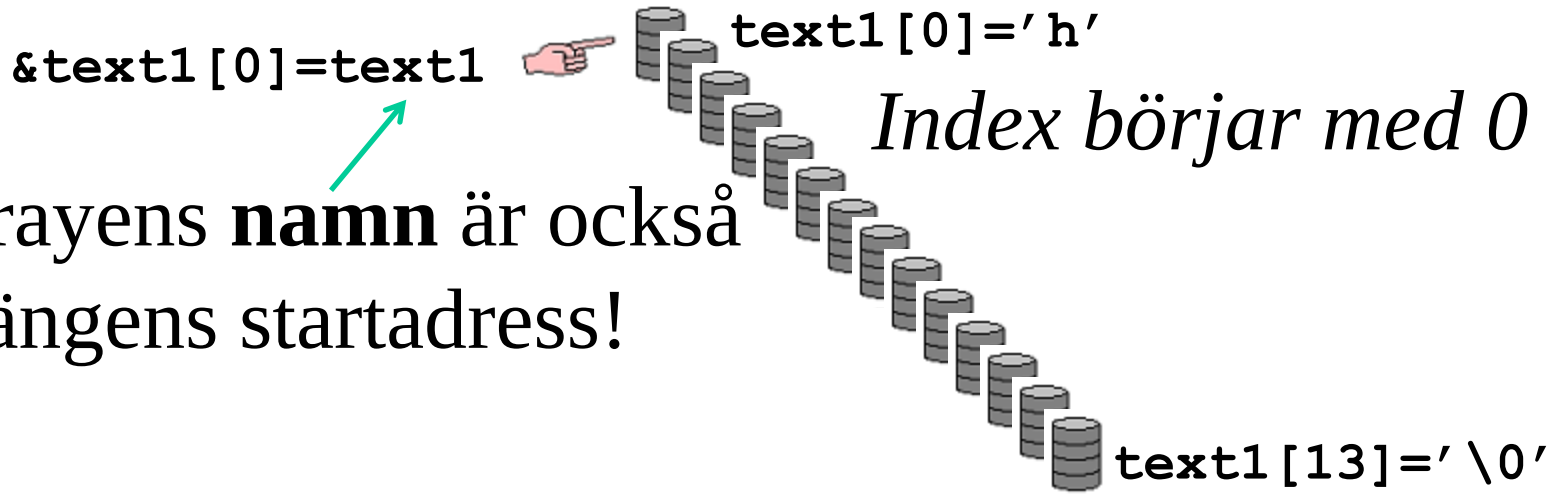
`if(str1 == str2)` betyder att **`str1`** och **`str2`** har samma adress, *inte* att strängarna har samma innehåll!

Strängar jämförs i stället med funktionen **`strcmp().`**

C-språkets ”strängar”

```
const char text1[]=”hello world\r\n”;
```

En sträng är en char-array som avslutats med ’\0’



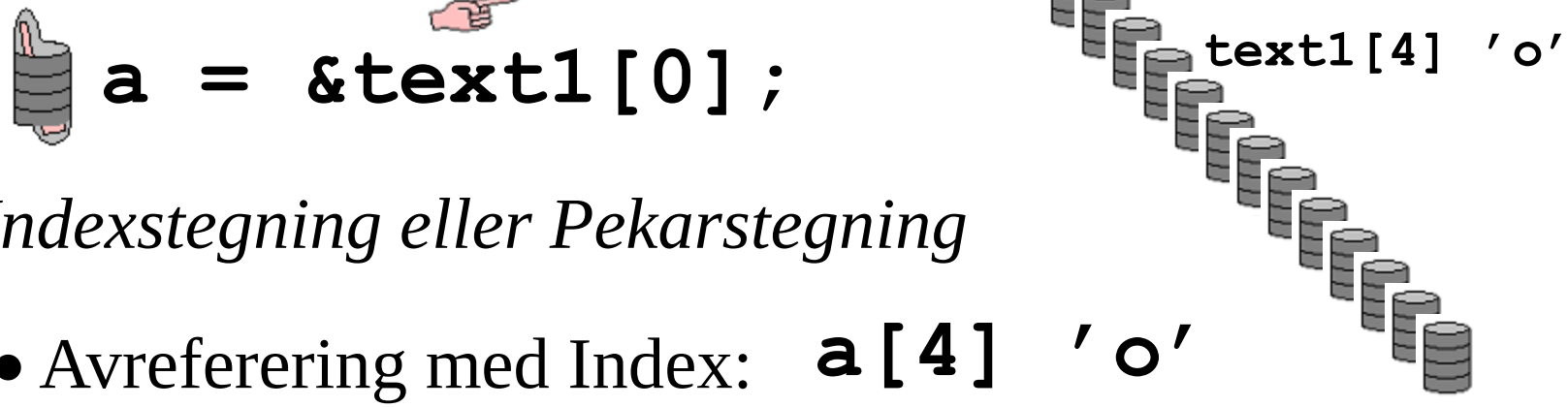
Index börjar med 0

Arrayens **namn** är också strängens startadress!

William Sandqvist william@kth.se

Adresser lagras i pekarvariabler

```
char * a;
```




```
a = &text1[0];
```

Indexstegning eller Pekarstegning

- Avreferering med Index: `a[4] 'o'`
- Avreferering av Pekare: `*(a+4) 'o'`

Adresser som funktionsparametrar

Om en funktion ska kunna skriva ut ***olika*** strängar behöver man kunna skicka med strängens startadress

Funktionsdeklaration:

```
void string_out( const char * );
```

Plats för startadressen

William Sandqvist william@kth.se

Adresser som funktionsparametrar

Funktionsdefinition:

```
void string_out( const char * s )
{
  char i,k;
  for(i = 0 ; ; i++)
   {
    k = s[i];
    if( k == '\0') return;
    putchar(k);
   }
}
```

Hämta bokstav

Här sker avrefererings med index

tills strängen är slut

Adresser som funktionsparametrar

Funktionsanrop:

skicka med startadressen

```
string_out( &text1[0] );
```

eller

```
string_out( ”hello world\r\n” );
```

skriver man så här så lagras strängkonstanten i minnet men det är bara startadressen som skickas som parameter