Introduktion C-programmering

Ulf Assarsson

Originalslides av Viktor Kämpe

C – Bakgrund

- Short Code, 1949, possibly 1:st high level language for a real machine
- Autocode, early 50'ies.
- Fortran, IBM, ~57.
- Lisp, 58.
- Cobol 60 (Common Business-oriented language.
- BASIC, 1964.
- ALGOL 60 (**ALGO**rithmic **L**anguage 19**60**).
- Simula, 60:ies.
- C, ~1969.
- Prolog, 1972.
- Ada, ~1975
- Pascal, ~1975.
- ML, 1978.

C – Bakgrund

- Short Code, 1949, 1:st high level language
- Autocode, early 50'ies.
- Fortran, IBM, ~57.
- Lisp, 58
- Cobol 60 (Common Business-oriented language.
- BASIC, 1964.
- ALGOL 60 (**ALGO**rithmic **L**anguage 19**60**).
- Simula, 60:ies.
- C, ~1969.
- Prolog, 1972.
- Ada, ~1975
- Pascal, ~1975.
- ML, 1978.

Kompilerades varje gång

Hade compiler

Finns kvar pga mycket legacy code

C – Bakgrund

- Short Code, 1949, 1:st high level language
- Autocode, early 50'ies.
- Fortran, IBM, ~57.
- L
- C
- B
- A
- S
- C
- F
- A
- F
- M

Kompilerades varje gång

Hade compiler

Finns kvar pga mycket legacy code

Fortran

```
C AREA OF A TRIANGLE - HERON'S FORMULA
C INPUT - CARD READER UNIT 5, INTEGER INPUT
C OUTPUT - LINE PRINTER UNIT 6, REAL OUTPUT
C INPUT ERROR DISPAY ERROR OUTPUT CODE 1 IN JOB CONTROL LISTING
      INTEGER A,B,C
      READ(5,501) A,B,C
  501 FORMAT(3I5)
      IF(A.EQ.0 .OR. B.EQ.0 .OR. C.EQ.0) STOP 1
      S = (A + B + C) / 2.0
      AREA = SQRT( S * (S - A) * (S - B) * (S - C))
      WRITE(6,601) A,B,C,AREA
  601 FORMAT(4H A= ,I5,5H  B= ,I5,5H  C= ,I5,8H  AREA= ,F10.2,12HSQUARE UNITS)
      STOP
      END
```

C – Bakgrund

- Short Code, 1949, 1:st high level language
- Autocode, early 50'ies.
- Fortran, IBM, ~57.
- Lisp, 58 För matematik/AI.
- Cobol 60 (Common Business-oriented language.
- BASIC, 1964.
- ALGOL 60 (**ALGO**rithmic **L**anguage 19**60**).
- Simula, 60:ies.
- C, ~1969.
- Prolog, 1972.
- Ada, ~1975
- Pascal, ~1975.
- ML, 1978.

Kompilerades varje gång

Hade compiler

Finns kvar pga mycket legacy code

Lisp

```
(defun factorial (n)
  (if (= n 0) 1
      (* n (factorial (- n 1)))))
```

C – Bakgrund

- Short Code, 1949, 1:st high level language
- Autocode, early 50'ies.
- Fortran, IBM, ~57.
- Lisp, 58
- Cobol 60 (Common Business-oriented language.
- BASIC, 1964.
- ALGOL 60 (**ALGO**rithmic **L**anguage 19**60**).
- Simula, 60:ies.
- C, ~1969.
- Prolog, 1972.
- Ada, ~1975
- Pascal, ~1975.
- ML, 1978.

Kompilerades varje gång

Hade compiler

Finns kvar pga mycket legacy code

För matematik/AI.

Imperativt, procedurellt, idag objektorienterat
För business + finance.
"Pratigt".

COBOL

```
IDENTIFICATION DIVISION.
    PROGRAM-ID. HELLO-WORLD.
    PROCEDURE DIVISION.
      DISPLAY 'Hello, world'.
      STOP RUN.
```

C – Bakgrund

- Short Code, 1949, 1:st high level language
- Autocode, early 50'ies.
- Fortran, IBM, ~57.
- Lisp, 58
- Cobol 60 (Common Business-oriented language.
- BASIC, 1964.
- ALGOL 60 (**ALGO**rithmic **L**anguage 19**60**).
- Simula, 60:ies.
- C, ~1969.
- Prolog, 1972.
- Ada, ~1975
- Pascal, ~1975.
- ML, 1978.

För matematik/AI.

Kompilerades varje gång

Hade compiler

Finns kvar pga mycket legacy code

Imperativt, procedurellt, idag objektorienterat

BASIC

```
5 LET S = 0
10 MAT INPUT V
20 LET N = NUM
30 IF N = 0 THEN 99
40 FOR I = 1 TO N
45 LET S = S + V(I)
50 NEXT I
60 PRINT S/N
70 GO TO 5
99 END
```

C – Bakgrund

- Short Code, 1949, 1:st high level language — Kompilerades varje gång
- Autocode, early 50'ies. — Hade compiler
- Fortran, IBM, ~57. — Finns kvar pga mycket legacy code
- Lisp, 58 — För matematik/AI.
- Cobol 60 (Common Business-oriented language. — Imperativt, procedurellt, idag objektorienterat
- BASIC, 1964.
- ALGOL 60 (**ALGO**rithmic **L**anguage 19**60**). — Influerade C
- Simula, 60:ies.
- C, ~1969.
- Prolog, 1972.
- Ada, ~1975
- Pascal, ~1975.
- ML, 1978.

C – Bakgrund

- Short Code, 1949, 1:st high level language
- Autocode, early 50'ies.
- Fortran, IBM, ~57.
- Lisp, 58
- Cobol 60 (Common Business-oriented language.
- BASIC, 1964.
- ALGOL 60 (**ALGO**rithm
- Simula, 60:ies.
- C, ~1969.
- Prolog, 1972.
- Ada, ~1975
- Pascal, ~1975.
- ML, 1978.

Kompilerades varje gång

Hade compiler

För matematik/AI.

Finns kvar pga mycket legacy code

Imperativt, procedurellt, idag objektorienterat

ALGOL

```
procedure Absmax(a) Size:(n, m) Result:(y) Subscripts:(i, k);
   value n, m; array a; integer n, m, i, k; real y;
begin integer p, q;
   y := 0; i := k := 1;
   for p:=1 step 1 until n do
   for q:=1 step 1 until m do
      if abs(a[p, q]) > y then
         begin y := abs(a[p, q]);
         i := p; k := q
         end
end Absmax
```

C – Bakgrund

- Short Code, 1949, 1:st high level language — Kompilerades varje gång
- Autocode, early 50'ies. — Hade compiler
- Fortran, IBM, ~57. — Finns kvar pga mycket legacy code
- Lisp, 58 — För matematik/AI.
- Cobol 60 (Common Business-oriented language. — Imperativt, procedurellt, idag objektorienterat
- BASIC, 1964.
- ALGOL 60 (**ALGO**rithmic **L**anguage 19**60**). — Influerade C
- Simula, 60:ies. — 1:a objektorienterade språk, klasser, virtuella metoder, objekt, arv
- C, ~1969.
- Prolog, 1972.
- Ada, ~1975
- Pascal, ~1975.
- ML, 1978.

```
! Simula
Begin
  OutText ("Hello World!");
  Outimage;
End;
```

C – Bakgrund

- Short Code, 1949, 1:st high level language — Kompilerades varje gång
- Autocode, early 50'ies. — Hade compiler
- Fortran, IBM, ~57. — Finns kvar pga mycket legacy code
- Lisp, 58 — För matematik/AI.
- Cobol 60 (Common Business-oriented language. — Imperativt, procedurellt, idag objektorienterat
- BASIC, 1964.
- ALGOL 60 (**ALGO**rithmic **L**anguage 19**60**). — Influerade C
- Simula, 60:ies. — 1:a objektorienterade språk, klasser, virtuella metoder, objekt, arv
- C, ~1969.
- Prolog, 1972.
- Ada, ~1975
- Pascal, ~1975.
- ML, 1978.

```c
/* C */
int main()
{
   printf("Hello World!\n");
   return 0;
}
```

C – Bakgrund

- Short Code, 1949, 1:st high level language — Kompilerades varje gång
- Autocode, early 50'ies. — Hade compiler
- Fortran, IBM, ~57. — Finns kvar pga mycket legacy code
- Lisp, 58 — För matematik/AI.
- Cobol 60 (Common Business-oriented language. — Imperativt, procedurellt, idag objektorienterat
- BASIC, 1964.
- ALGOL 60 (**ALGO**rithmic **L**anguage 19**60**). — Influerade C
- Simula, 60:ies. — 1:a objektorienterade språk, klasser, virtuella metoder, objekt, arv
- C, ~1969.
- Prolog, 1972. — AI/lingvistik. Facts/rules, queries over relationships
- Ada, ~1975
- Pascal, ~1975.
- ML, 1978.

Prolog

```
?- write('Hello world!'), nl.
Hello world!
true.

?-
```

C – Bakgrund

- Short Code, 1949, 1:st high level language — Kompilerades varje gång
- Autocode, early 50'ies. — Hade compiler
- Fortran, IBM, ~57. — Finns kvar pga mycket legacy code
- Lisp, 58 — För matematik/AI.
- Cobol 60 (Common Business-oriented language. — Imperativt, procedurellt, idag objektorienterat
- BASIC, 1964.
- ALGOL 60 (**ALGO**rithmic **L**anguage 19**60**). — Influerade C
- Simula, 60:ies. — 1:a objektorienterade språk, klasser, virtuella metoder, objekt, arv
- C, ~1969.
- Prolog, 1972. — AI/lingvistik. Facts/rules, queries over relationships
- Ada, ~1975 — Imperativt, procedurellt, idag objektorienterat
- Pascal, ~1975.
- ML, 1978.

```
with Ada.Text_IO; use Ada.Text_IO;
procedure Hello is
begin
  Put_Line ("Hello, world!");
end Hello;
```

C – Bakgrund

- Short Code, 1949, 1:st high level language — Kompilerades varje gång
- Autocode, early 50'ies. — Hade compiler
- Fortran, IBM, ~57. — Finns kvar pga mycket legacy code
- Lisp, 58 — För matematik/AI.
- Cobol 60 (Common Business-oriented language. — Imperativt, procedurellt, idag objektorienterat
- BASIC, 1964.
- ALGOL 60 (**ALGO**rithmic **L**anguage 19**60**). — Influerade C
- Simula, 60:ies. — 1:a objektorienterade språk, klasser, virtuella metoder, objekt, arv
- C, ~1969.
- Prolog, 1972. — AI/lingvistik. Facts/rules, queries over relationships
- Ada, ~1975
- Pascal, ~1975.
- ML, 1978.

Pascal

```
Program HelloWorld;
Begin
  WriteLn('Hello world!')
  {no ";" is required after the last statement of a block -
   adding one adds a "null statement" to the program}
End.
```

C – Bakgrund

- Short Code, 1949, 1:st high level language — Kompilerades varje gång
- Autocode, early 50'ies. — Hade compiler
- Fortran, IBM, ~57. — Finns kvar pga mycket legacy code
- Lisp, 58 — För matematik/AI.
- Cobol 60 (Common Business-oriented language. — Imperativt, procedurellt, idag objektorienterat
- BASIC, 1964.
- ALGOL 60 (**ALGO**rithmic **L**anguage 19**60**). — Influerade C
- Simula, 60:ies. — 1:a objektorienterade språk, klasser, virtuella metoder, objekt, arv
- C, ~1969.
- Prolog, 1972. — AI/lingvistik. Facts/rules, queries over relationships
- Ada, ~1975 — Imperativt, procedurellt, idag objektorienterat
- Pascal, ~1975. — Imperativt, procedurellt, idag objektorienterat
- ML, 1978. — Funktionellt, rekursivt. Föregångare till Haskell.

```
fun fac (0 : int) : int = 1
  | fac (n : int) : int = n * fac (n - 1)
```

C – Bakgrund

- Maskinnära programmering:
  - Behöver språk med pekare till absoluta adresser
    - Dvs definiera sin pekare till en godtycklig adress.
    - C, C++, D, (C#, COBOL, Fortran, Basic).

C – Historik

- B, Bell Labs ~1969
- C: Utvecklades först 1969 – 1973 av Dennis Ritchcie vid AT&T Bell Labs.
- Högnivå språk med kontakt mot maskinvara.
- Ett utav de mest använda språken.
- C++, D.

- Maskinnära, pekare, front/backend

C respektive Assembler

- Varför C istället för assembler?
- Varför förstå hur C kompileras till assembler?
  - prestandaoptimering och resonera kring prestanda (tex för datorgrafik, GPU:er, HPC).
  - energikonsumption
  - säkerhet/robusthet/risker
    - Kunna debugga

Översikt C – fyra lektioner

- Lekt 1: intro till C
- Lekt 2+3: pekare
- Lekt 4: mixa C och assembler

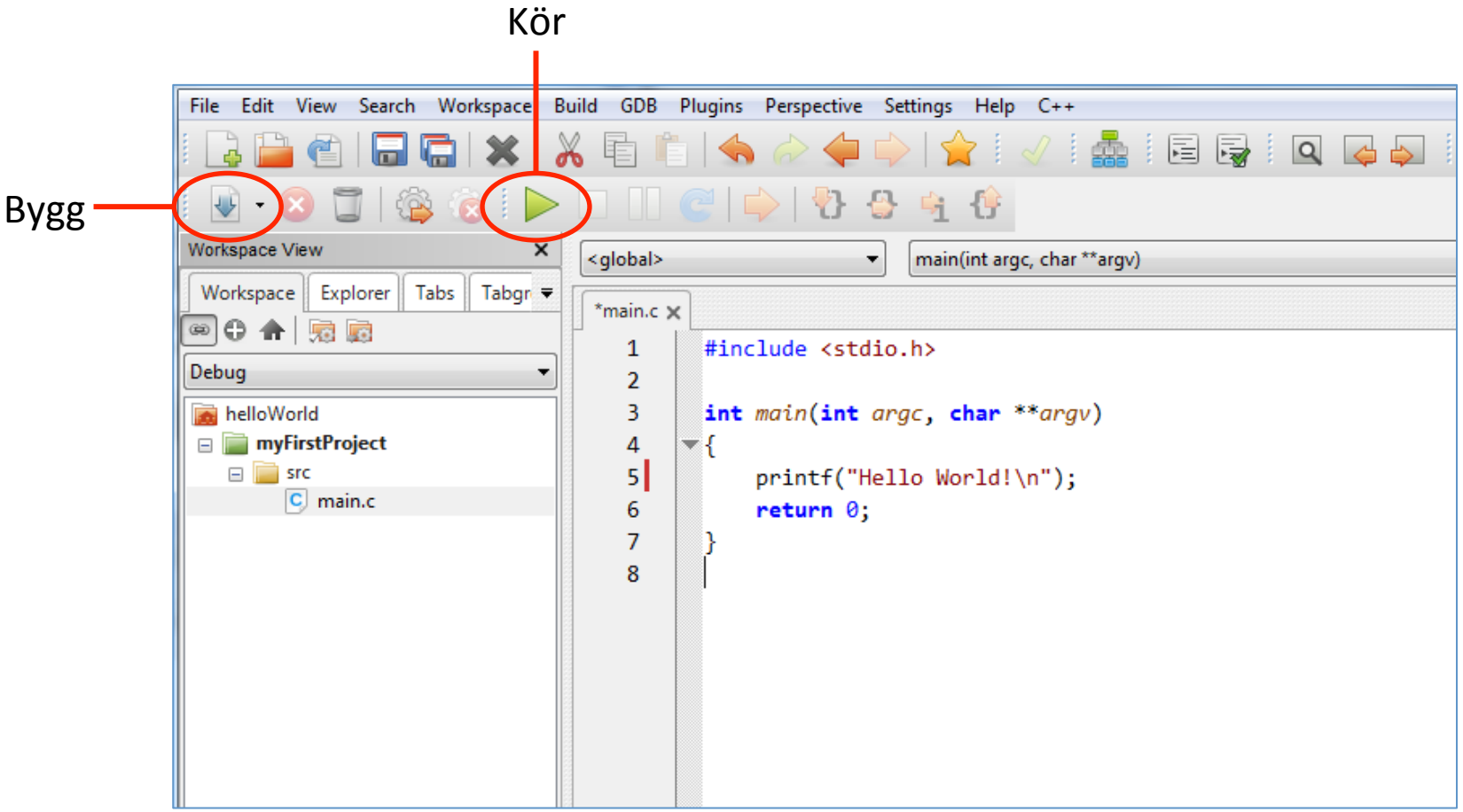
C – Standarder

- 1978, K&R C (Kernighan & Ritchie)
- 1989, C89/C90 (ANSI-C)
- 1999, C99 (Rev. ANSI-standard)
- 2011, C11 (Rev. ANSI-standard)

Hello world! – program

```
#include <stdio.h>

int main()
{
   printf(“Hello World!\n");
   return 0;
}
```

Integrerad utvecklings miljö (IDE)



T ex CodeLite som är gratis och open-source. (http://codelite.org/)

Från terminalen

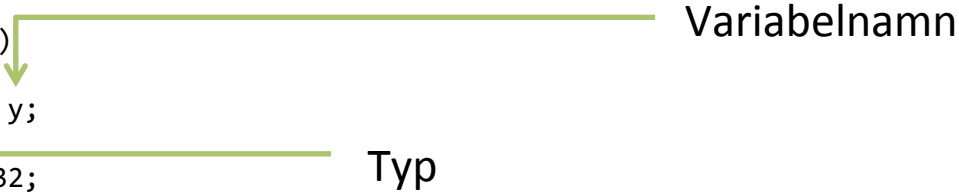
```
> gcc -o hello.exe main.c
> hello.exe
Hello World!
>
```


Bygg


Kör

Variabler

```c
#include <stdio.h>

int x;

int main()
{
     char y;
     x = 32;
     y = 'a';

     x = x + y;

     printf("x har nu värdet %i och y har värdet %i som kodar tecknet %c\n", x, (int)y, y);
     return 0;
}
```

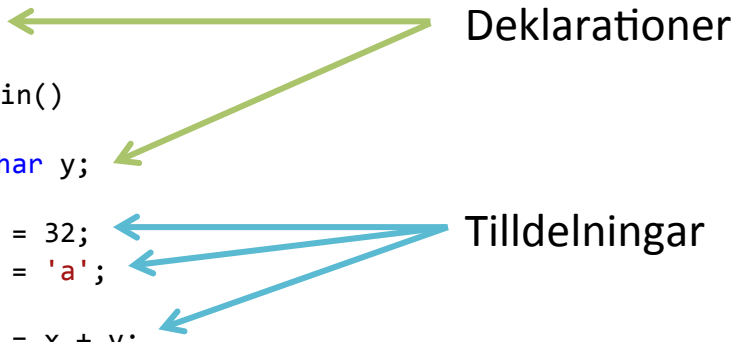


Utskrift:

```
x har nu värdet 129 och y har värdet 97 som kodar tecknet a
```

Deklarationer och tilldelningar

```
#include <stdio.h>

int x;

int main()
{
     char y;

     x = 32;
     y = 'a';

     x = x + y;

     printf("x har nu värdet %i och y har värdet %i som kodar tecknet %c\n", x, (int)y, y);
     return 0;
}
```



En deklarerad variabel som ännu inte tilldelats ett värde är *oinitialiserad*

Typkonverteringar

```c
#include <stdio.h>

int x;

int main()
{
     char y;

     x = 32;
     y = 'a';

     x = x + y;

     printf("x har nu värdet %i och y har värdet %i som kodar tecknet %c\n", x, (int)y, y);
     return 0;
}
```

Implicit typkonvertering

Explicit typkonvertering

Typkonvertering kallas också *cast*, och man säger att man *castar*.

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

C89 – deklarationer först

```
#include <stdio.h>

int x;

int main()
{
     x = 32;

     char y = 'a';

     x = x + y;

     printf("x har nu värdet %i och y har värdet %i som kodar tecknet %c\n", x, (int)y, y);
     return 0;
}
```

En tilldelning innan deklarationerna är **EJ** tillåtet enligt C89

Fungerar ibland ändå (t ex i CodeLite), men inte i XCC12 som vi ska använda senare.

Funktioner

```
#include <stdlib.h>

int foo(int x, char y)
{
    int sum = 0;

    while(y-- > 0) {
        sum += x*y;
    }

    return sum;
}
```

argument

Returvärde av returtyp

Argumenten är ''pass-by value''.

```
int var1;
char var2 = 7;
var1 = foo(5, var2);
```

var2 har fortfarande värdet 7 efter funktionsanropet

Synlighet/Visibility/Scope

- Global synlighet (global scope)
- Filsynlighet (file scope)
- Lokal synlighet (e.g. function scope)

Synlighet

```c
#include <stdlib.h>

char x;

int foo()
{
    // x är synlig
    // y är inte synlig
}

char y;
```

Synlighet på funktionsnivå

```c
#include <stdlib.h>

char x;

int foo(float x)
{
    // argumentet x (float) är synligt
}
```

Synlighet på funktionsnivå

```c
#include <stdlib.h>

char x;

int foo()
{
    int x = 4;
    return x;
}
```

Vilken synlighet har högst prioritet?

```
#include <stdio.h>

int x;

int foo(int x)
{
    if( x == 0 ){
        int x = 4;
        return x;
    }

    return x;
}

int main()
{
    x = 1;
    x = foo(0);
    printf("x is %i", x);
    return 0;
}
```

Vad är x?

Funktionsprototyper

```c
#include <stdio.h>

// funktionsprototyp
int foo(int x);

int main()
{
    printf("x is %i", foo(0));
    return 0;
}

int foo(int x)
{
    // funktionskropp
}
```

Programstruktur

```c
// main.c
#include <stdio.h>
#include "foo.h"

int main()
{
     printf("x is %i", foo(0));
     return 0;
}
```

c-fil
Inkluderar header-fil

```c
// foo.h
int foo(int x);
```

header-fil
Innehåller funktionsprototyper

```c
// foo.c
#include <stdlib.h>

int foo(int x)
{
     if( x == 0 ){
          int x = 4;
          return x;
     }

     return x;
}
```

c-fil

header-filen måste inte ha samma namn som c-filen, men det är enklare så.

Från källkod till exekverbar

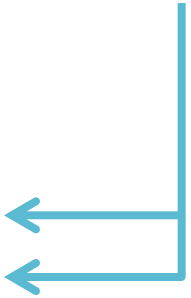
1. Preprocessing
2. Kompilering
3. Länkning

Preprocessorn

```
// main.c
#include <stdio.h>
#include "foo.h"

#define MAX_SCORE 100
#define SQUARE(x) (x)*(x)

int main()
{
    printf("Högsta möjliga poäng är %i\n", MAX_SCORE);
    printf("Kvadraten av 3 är %i\n", SQUARE(1+2));
    printf("x is %i", foo(0));
    return 0;
}
```

Copy-paste av filer

Find-and-replace av strängar



Preprocessorn arbetar på källkoden på ’’textnivå’’.

Kompilering

- Processar en c-fil i taget
- Skapar en objektfil som innehåller:
  - Maskinkod för instruktioner
  - Symboler för addresser
    - För funktioner/variabler i objektfilen.
    - För funktioner/variabler i andra objektfiler/bibliotek.

Länkning

- Sätter samman (flera) objektfiler till en exekverbar fil (.exe).
- Översätter symbolerna till (relativa) adresser.

Aritmetiska operatorer

Basic assignment		a = b
Addition		a + b
Subtraction		a - b
Unary plus (integer promotion)		+a
Unary minus (additive inverse)		-a
Multiplication		a * b
Division		a / b
Modulo (integer remainder)		a % b
Increment	Prefix	++a
	Postfix	a++
Decrement	Prefix	--a
	Postfix	a--

http://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B#Arithmetic_operators

Jämförelseoperatorer

Equal to	a == b
Not equal to	a != b
Greater than	a > b
Less than	a < b
Greater than or equal to	a >= b
Less than or equal to	a <= b

http://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B#Comparison_operators.2Frelational_operators

Logiska operatorer

Logical negation (NOT)	**!**a
Logical AND	a **&&** b
Logical OR	a **||** b

http://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B#Logical_operators

Bit operationer

Bitwise NOT	**~**a
Bitwise AND	a **&** b
Bitwise OR	a **|** b
Bitwise XOR	a **^** b
Bitwise left shift	a **<<** b
Bitwise right shift	a **>>** b

http://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B#Bitwise_operators

Sammansatta tilldelsningsoperatorer

Operator name	Syntax	Meaning
Addition assignment	a **+=** b	a = a **+** b
Subtraction assignment	a **-=** b	a = a **-** b
Multiplication assignment	a ***=** b	a = a ***** b
Division assignment	a **/=** b	a = a **/** b
Modulo assignment	a **%=** b	a = a **%** b
Bitwise AND assignment	a **&=** b	a = a **&** b
Bitwise OR assignment	a **|=** b	a = a **|** b
Bitwise XOR assignment	a **^=** b	a = a **^** b
Bitwise left shift assignment	a **<<=** b	a = a **<<** b
Bitwise right shift assignment	a **>>=** b	a = a **>>** b

http://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B#Compound_assignment_operators

If-else satser

```
int x = -4;

if( x == 0 ){
    // ...
}

if( x ){
    // ...
}
else {
    // ...
}
```

Utvärderar till falskt, kör ej.

Utvärderas till sant, kör, ty (x != 0)

- Noll betraktas som falskt.
- Allt som är skilt från noll betraktas som sant.

Loopar

```
int x = 5;

while( x!=0 )
    x--;
```

```
int x = 5;

while( x )
    x--;
```

```
int x;

for( x=5; x; )
    x--;
```

Tre ekvivalenta loopar. Om inga måsvingar används så är loop-kroppen ett enda uttryck.

Nästa föreläsning: Pekare

Pekare

- Har ett värde och en typ
  - Värdet är en minnesadress.
  - Typen talar om vad som finns där.

Operatorer för pekare

Operator name	Syntax
Array subscript	a**[**b**]**
Indirection ("object pointed to by *a*")	***a**
Reference ("address of *a*")	**&**a
Structure dereference ("member *b* of object pointed to by *a*")	a**->**b
Structure reference ("member *b* of object *a*")	a**.**b

http://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B#Member_and_pointer_operators

Sista operatorn är för att referera medlemmar av en struktur (`struct`), så ej en pekare operator.