Pekare och Arrayer

Ulf Assarsson

Originalslides av Viktor Kämpe

www.cse.chalmers.se/~uffe/

Google YouTube Dictionary.com Eniro Personer Ulf Assarsson's Home Page Översätt Google Maps Insightly - Login badrum Baby Sleep ...mv - YouTube

Kurshemsida i PP - Innehåll - LEU500... | LEU500 2014/2015 Kurs PM - Veckopl... | Ulf Assarsson's Home Page | Ulf Assarsson's Home Page

CHALMERS GÖTEBORG UNIVERSITY

Computer Science and Engineering

- Home
- Research Group
- Publications
- Teaching
- Soft Shadows
- BART
- Bachelor's Theses
- Master's Theses

Teaching

My courses at the Department of Computer Engineering, Chalmers University of Technology:

- Seminar Course in Advanced Computer Graphics
- TDA 361 - Computer Graphics
- Maskinorienterad Programmering / Programmering av Inbyggda System
- EDA 425 -Advanced Computer Graphics
- TDA 360 - Computer Graphics

Main research interests: Shadow algorithms - both real-time and non real-time - for hard and soft shadows including shadows in hair, fur, smoke and volumetric participating media (shafts of light), Parallelism, GPGPU-algorithms, GPU-Ray Tracing, Real-time Global Illumination, Real-time rendering algorithms.

My PhD-students:

- Markus Billeter
- Ola Olsson
- Erik Sintorn
- Viktor Kämpe

Previous commitments:
1997-1999: Prosolvia

Computer Graphics Research Group

www.cse.chalmers.se/~uffe/MOP/mop.html

Google YouTube Dictionary.com Eniro Personer Ulf Assarsson's Home Page Översätt Google Maps Insightly - Login badrum >>

C-kurs

Delkurs i Maskinorienterad Programmering samt Programmering av Inbyggda System. Slides:

- Intro
- Pekare och Fält
- Pekare fortsättning
- Kodningskonventioner

Länk till space-invaderspel i CodeLite och C.

./game

Level 6
Health: 3
Time: 00301

Här jag jag lagt upp ett något större projekt som är programmerat i C, så att ni kan se ett mer realistiskt exempel på kodstruktur och hantering av pekare och fält i C.

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.

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

CHALMERS

C – Bakgrund

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

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 — 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.
- BASIC, 1964.
- ALGOL 60 (**ALGO**rithmic **L**anguage 19**60**).
- Simula, 60:ies.
- C, ~1969.
- Prolog, 1972.
- Ada, ~1975
- Pascal, ~1975.
- ML, 1978.

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 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

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 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

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 — 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**rithm
- Simula, 60:ies.
- C, ~1969.
- Prolog, 1972.
- Ada, ~1975
- Pascal, ~1975.
- ML, 1978.

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 */
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
    - Basic, Ada 95 (och senare versioner), C, C++, C# (med nyckelordet *unsafe*), Objective-C, D, COBOL, Fortran.

    - C - 1969
    - C++ - 1983
    - (Java – 1995)
    - C# - 2000, strong type checking, garbage collection, obj. oriented, “COOL”.
    - D - 2001

C++

- EON
- Autodesk
- Surgical Science
- Mentice
- Vizendo
- Spark Vision
- ABB
- DICE / EA
- EA Ghost Games
- 2K Sports - C
- Ericsson – C / Java
- Microsoft – wants to promote XNA and C#
  - Advantage: works on Xbox 360
  - Xbox one: C++ / C#
- Playstation 3 – uses OpenGL ES and C++
- Playstation 4 – C++ (C#, C, …)

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?
  - Färre rader kod, mindre risk för fel, snabbare…
- Varför förstå hur C kompileras till assembler?
  - prestandaoptimering och resonera kring prestanda (tex för datorgrafik, GPU:er, HPC).
    - Hur mycket snabbare är en while-loop än rekursion?
    - Loop-unroll?
  - energikonsumption
  - säkerhet/robusthet/risker
    - Kunna debugga
  - Kunna mixa C/asm vid drivrutinsprogrammering eller prestandakritiska förlopp.

Översikt C – trelektioner

- Lekt 1: intro + pekare
- Lekt 2: pekare forts. (Portadressering)
- Lekt 3: mixa C och assembler. (Kodningskonventioner)

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;
}
```

C vs Java. Några skillnader:

- C: saknar klasser. Har dock structs för sammansatta datatyper.

```
struct Course {
    char* name;
    float credits;
    int numberOfParticipants;
};
```

- Booleans är ej egen typ. true/false finns ej. 0 är false. Ett värde != 0 är true. Därför definierar man ofta TRUE/FALSE som 1 resp 0. 1 && 1 = “Implementationsspecifikt för kompilatorn”.

Mer C:

- Type conversion
  - default: convert narrower format to the wider format
  - Synlighet:
    - global synlighet,
    - synlighet i funktion
    - resp scope.

```
float a;
a = 1.0 / 3;
// a == 0.33333343
```

```
#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;
}
```

Synlighet/Visibility/Scope

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

Synlighet

```
#include <stdlib.h>

char x;

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

char y;
```

CHALMERS

Synlighet på funktionsnivå

```
#include <stdlib.h>

char x;

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

Synlighet på funktionsnivå

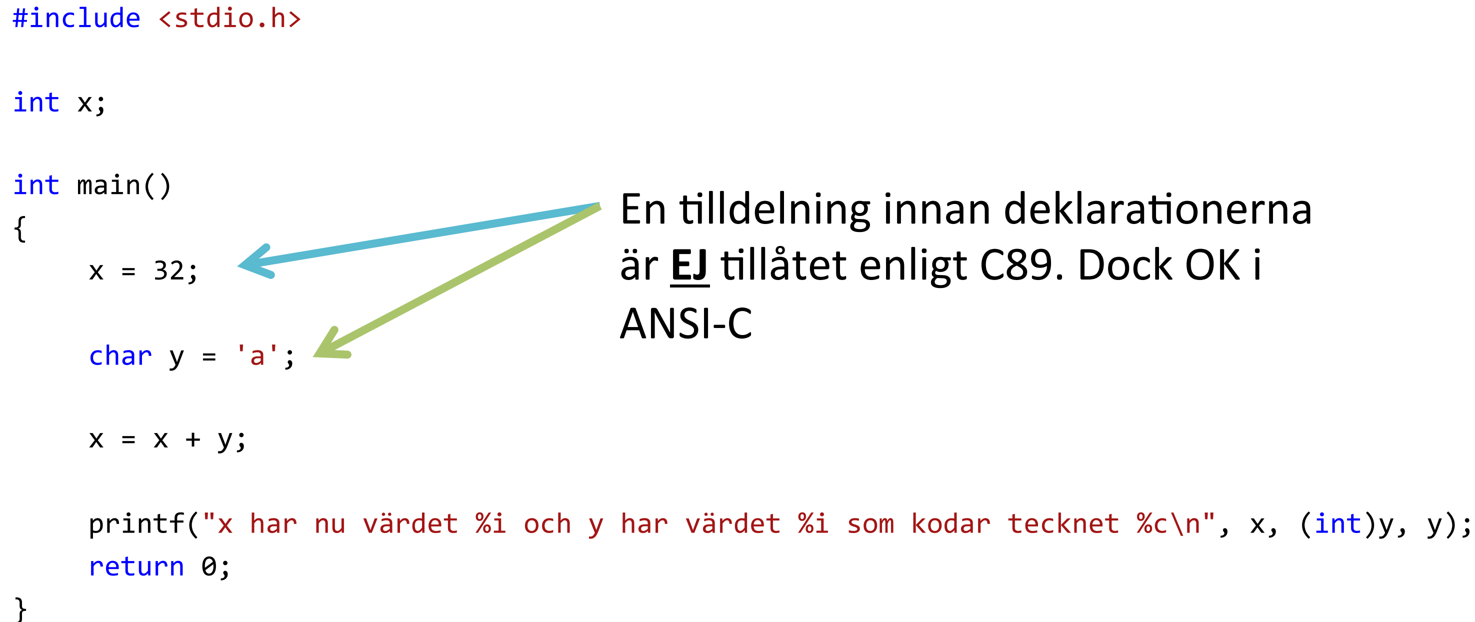
```c
#include <stdlib.h>

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 printas? ... Svar: 4

C89 – deklarationer först




```c
#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;
}
```

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

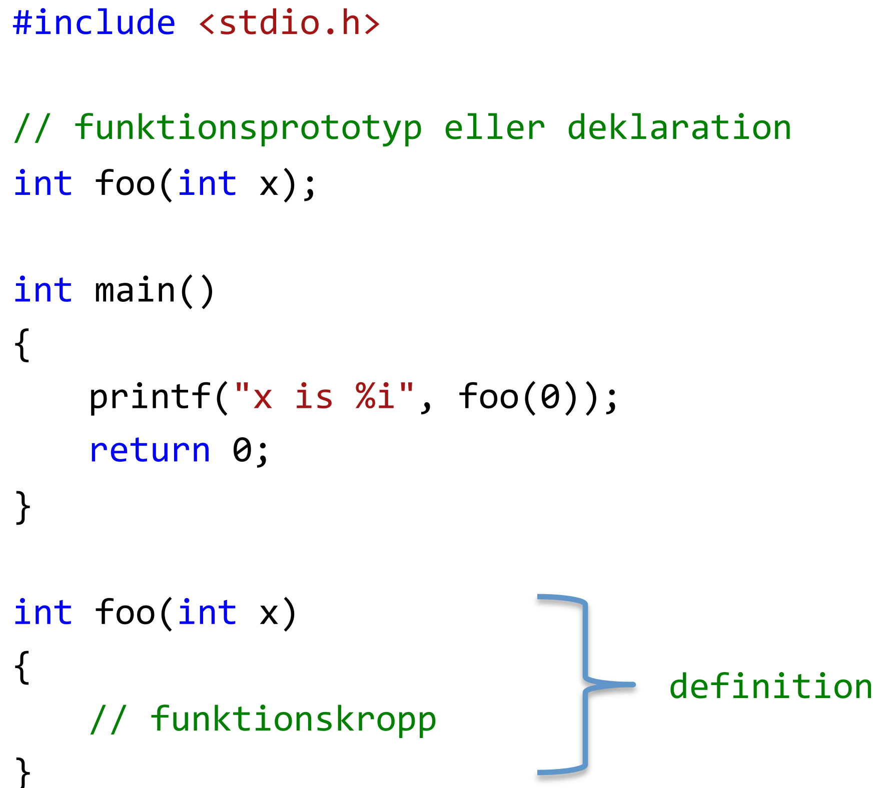
Returvärde av returtyp

Argumenten är ”pass-by value”.

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

var2 har fortfarande värdet 7 efter funktionsanropet

Funktionsprototyper



```
#include <stdio.h>

// funktionsprototyp eller deklaration
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

```c
// 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.

```
> gcc -o hello.exe main.c foo.c
> gcc -o hello.exe main.o foo.o
```

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

Sammansatta tilldelsningsoperatorer

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

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

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

```c
int x = 5;

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

```c
int x = 5;

while( x )
    x--;
```

```c
int x;

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

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

Pekare

- Pekarens värde är en adress.
- Pekarens typ berättar hur man tolkar bitarna som finns på adressen.

```
// array av chars
char str[] = "apa";

// pekare till char
char* p = str; // == &str[0] == &(str[0])

*p == 'a';
```

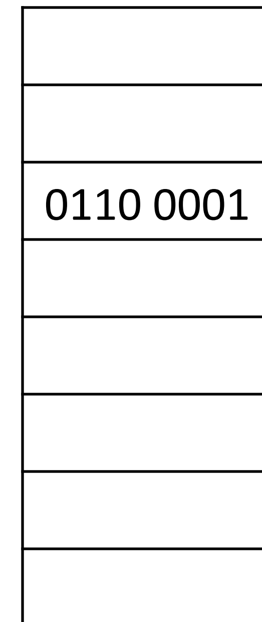
Pekare

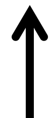
```
char str[] = ”abc";
char* p = str;
```

- Pekarens värde är en adress.
- Pekarens typ berättar hur man tolkar bitarna som finns på adressen.

```
char* p = str; // == 0x3026
```

typ — värdet är en adress

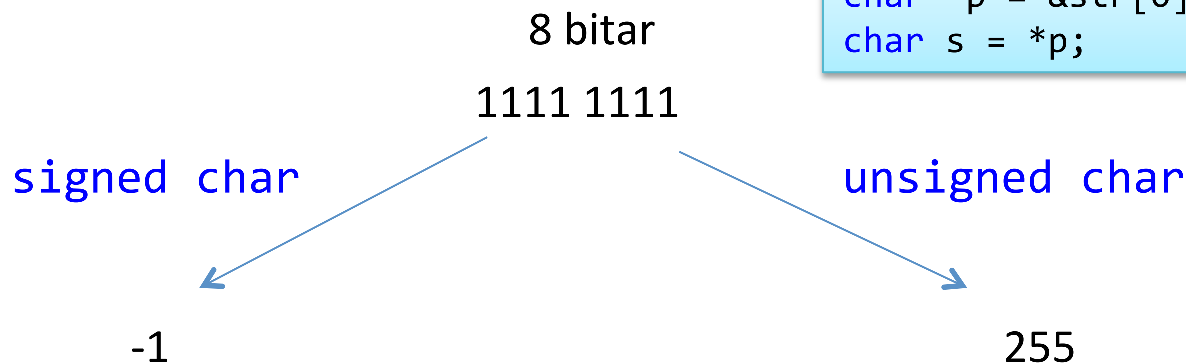




Dereferera

- När vi dereffererar en pekare så hämtar vi objektet som ligger på adressen.
  - Antalet bytes vi läser beror på typen
  - Tolkningen av bitarna beror på typen

```
char str[] = "apa”;
char* p = &str[0]; // = &(str[0])
char s = *p;
```



char är ett 8-bits värde, dvs ett 8-bits tal. Ett tal kan motsvara en bokstav via ASCII-tabellen.

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]*

Operatorer

```
#include <stdio.h>

int main()
{
    char a, b, *p;
    a = 'v';

    b = a;
    p = &a;

    printf("b = %c, p = 0x%p (%c) \n", b, p, *p);

    a = 'k';
    printf("b = %c, p = 0x%p (%c) \n", b, p, *p);
}
```

Deklaration av pekare

Adressen av ...

Dereferering

Utskift :

```
b = v, p = 0x0027F7C3 (v)
b = v, p = 0x0027F7C3 (k)
```

Asterisken (*) betyder

- I deklarationer
  - Pekartyp

```
char *p;
char* p;

void foo(int *pi);
```

- Som operator
  - Dereferens (”av-referera”)

```
char a = *p;
*p = 'b';
```

C vs Assembler

C	Assembler
short int v;	v RMB 2
v;	LDD v
&v;	LDD #v
*v;	LDX #v eller LDX v LDD [0, X] LDD 0, X
	På vissa CPU-arkitekturer kan man skriva: LDD [v]

Java har lurat er...

```
Dog myDog = new Dog("Rover");
```

myDog är (under ytan) en pekare till en Dog.

Pekare behövs oftast i ett program. Men ett programspråk kan dölja behovet genom språkets konstruktion.

Aritmetik på pekare

```
char *kurs = "Maskinorienterad Programmering";


*kurs;          // 'M'
*(kurs+2);      // 's'
kurs++;         // kurs pekar på 'a'
kurs +=4;       // kurs pekar på 'n'
```

Man ökar p med (n * typstorlek)

[Sträng-exempel i CodeLite]

Array (Fält)

```c
#include <stdio.h>

char namn1[] = {'E', 'm', 'i', 'l', '\0'};
char namn2[] = "Emilia";
char namn3[10];

int main()
{
    printf("namn1: %s \n", namn1);
    printf("namn2: %s \n", namn2);
    printf("sizeof(namn2): %i \n", sizeof(namn2));

    return 0;
}
```

Utskrift:

```
namn1: Emil
namn2: Emilia
sizeof(namn2): 7
```

Array - Likhet med pekare

- Har en adress och en typ.
  - `char s2[] = "Emilia";`
    - `- sizeof(s2) == 7`
    - `- Men sizeof(char*) == 4;`

- Indexering har samma resultat.
  - `char* s1 = "Emilia";`
  - `char s2[] = "Emilia";`
    - s1[0] == ’E’;
    - s2[0] == ’E’;
    - *s1 == ’E’;
    - *s2 == ’E’; // eftersom s2 är adress så kan vi dereferera den
      // precis som för en pekare

Indexering

```
#include <stdio.h>

char* s1 = "Emilia";
char s2[] = "Emilia";

int main()
{
    // tre ekvivalenta sätt att dereferera en pekare
    printf("'l' i Emilia (version 1): %c \n",   *(s1+3));
    printf("'l' i Emilia (version 2): %c \n",   s1[3]  );
    printf("'l' i Emilia (version 3): %c \n",   3[s1]  );

    return 0;
}
```

`x[y]` översätts till `*(x+y)` och är alltså ett sätt att dereferera en pekare.

```
char* s1 = "Emilia";
char s2[] = "Emilia";
```

Skillnader mellan array och pekare

- Arrayer
  - s2 är symbol och kan ej ändra värde. Värdet är s2’s adress i minnet.
  - Adress känd i compile-time.
  - Storlek känd i compile-time.
    - Storlek för pekare är storlek på adress vilket är 4 bytes på 32-bitssystem.
  - Pekar-aritmetik inte möjlig.
    - a[] = ”hej”;
    - a++ ej OK
    - (a+1)[0] helt OK.
  - a är en symbol för adressen till ’h’.
  - char* p = a;
    - p är en variabel som går att ändra och här sätts till a, dvs adressen för ’h’.

Oftast en relativ adress. T ex 104 bytes efter första instruktionen.

Array (Fält)

```c
#include <stdio.h>

char namn1[] = {'E', 'm', 'i', 'l', '\0'};
char namn2[] = "Emilia";
char namn3[10];

int main()
{
    printf("namn1: %s \n", namn1);
    printf("namn2: %s \n", namn2);
    printf("sizeof(namn2): %i \n", sizeof(namn2));

    return 0;
}
```

Om ingen storlek anges i [ ] så blir arrayen ”tillräckligt stor” .
OBS. Funkar bara med initiering vid deklaration!

10 element stor array.

Utskrift:

```
namn1: Emil
namn2: Emilia
sizeof(namn2): 7
```

Array (Fält)

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

char * s1 = "Emilia"; // s1 är pekare. Variabeln s1 är en variabel som går att ändra,
                     och vid start tilldelas värdet av adressen till 'E’:
char s2[] = "Emilia"; // s2 är array. Värdet på s2 är känt vid compile time. s2 är konstant,
                      dvs ingen variabel som går att ändra. Är adressen till 'E’.
int main()
{
    // tre ekvivalenta sätt att dereferera en pekare
    printf("'l' i Emilia (version 1): %c \n",     *(s1+3)    );
    printf("'l' i Emilia (version 2): %c \n",     s1[3]      );
    printf("'l' i Emilia (version 3): %c \n",     3[s1]      );
    printf("'l' i Emilia (version 3): %c \n",     *(s2+3)    );
    printf("'l' i Emilia (version 3): %c \n",     (s2+3)[0]  );

    char a[] = "hej";
    (a+1)[0] = 'o';
    char* p = a;
    p = "bye"; // funkar. Strängen ”bye" allokeras i compile time i strängliteralminne.

    char b[10]; // b blir 10 element stor och får adressvärde
    // b = "då"; // här försöker vi ändra b's värde och det GÅR inte.
    printf("%s\n", p);

    return 0;
}
```

Arrayer som funktionsargument blir pekare

```
void foo(int pi[]);
```

```
void foo(int *pi);
```

[ ] – notationen finns, men betyder pekare!

Undviker att hela arrayen kopieras. Längd inte alltid känd i compile time.
Adressen till arrayen läggs på stacken och accessas via stackvariabeln pi.

(En struct kopieras och läggs på stacken).

[Array-exempel i CodeLite]

Antal bytes med sizeof()

```
#include <stdio.h>

char* s1 = "Emilia";
char s2[] = "Emilia";

int main()
{
    printf("sizeof(char):  %i \n", sizeof(char) );
    printf("sizeof(char*): %i \n", sizeof(char*) );
    printf("sizeof(s1):    %i \n", sizeof(s1) );
    printf("sizeof(s2):    %i \n", sizeof(s2) );

    return 0;
}
```

```
sizeof(char):  1
sizeof(char*): 4
sizeof(s1):    4
sizeof(s2):    7
```

Sizeof utvärderas i compile-time. En (av få) undantag där arrayer och pekare är olika.

Dynamisk minnesallokering

- `malloc()` Allokera minne
- `free()` Frigör minne

Funktionsprototyp via:

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

Dynamisk minnesallokering

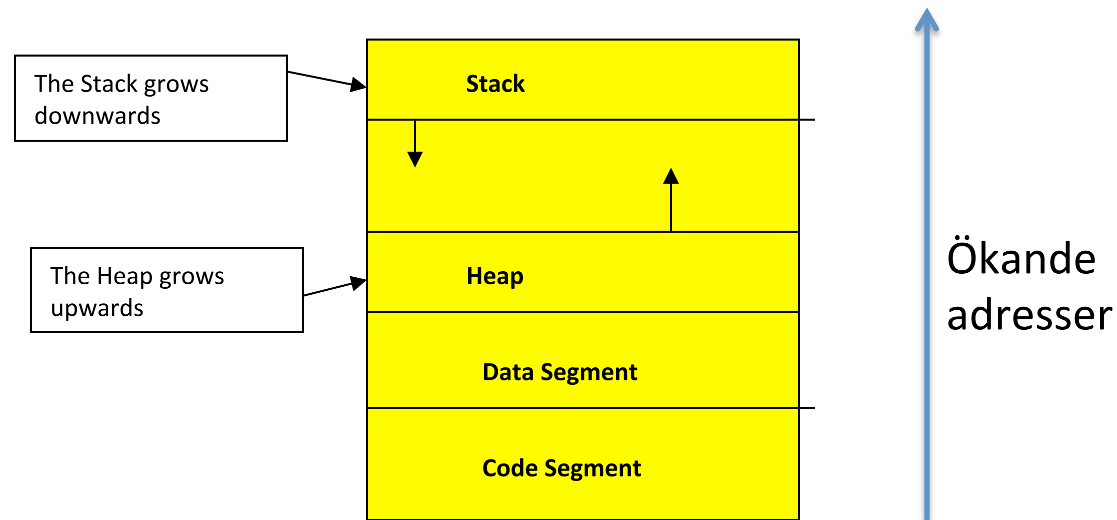
```
#include <stdlib.h>

char s1[] = "This is a long string. It is even more than one sentence.";

int main()
{
    char* p;

    // allokera minne dynamiskt
    p = (char*)malloc(sizeof(s1));

    // gör något med minnet som vi reserverat

    // frigör minnet
    free(p);
    return 0;
}
```

Antal bytes vi allokerar

OUT OF MEMORY
virtuellt adressutrymme uppdelat i pages som swappas mot hårddisk. Även detta kan ta slut -> krasch.

Ett Programs Adressrymd

- Alla program som körs, har något associerat minne, vilket typiskt är indelat i:
  - Code Segment
  - Data Segment (Holds Global Data)
  - Stack (where the local variables and other temporary information is stored)
  - Heap



Exempel på skrivskyddat minne…

[Minnesallokering-exempel]

Minnesallokeringsexempel

```
// Kopiera från s1 till det allokerade minnet som pekas ut av p.
#include <stdio.h>
#include <conio.h>
char s1[] = "This is a long string. It is even more than one sentence.";
int main()
{
        char* p;
        int i;
        // allokera minne dynamiskt
        p = (char*)malloc(sizeof(s1));

        // gör något med minnet som vi reserverat
        for( i=0; i<sizeof(s1); i++)
                *(p+i) = s1[i];
        printf("%s", p);

        // frigör minnet
        free(p);
        getch();
        return 0;
}
```

Minnesläckor

- En minnesläcka uppkommer om vi inte frigör det minne som vi allokerat med `malloc()`.
- Minnesläckor kan orsaka systemhaveri om minnet tar slut.
- Minnesläckan försvinner när programmet terminerar.

Hitta minnesläckor

- Vi kommer använda en minnes analysator DrMemory (http://www.drmemory.org/)
- DrMemory ersätter standard biblioteket, och analyserar anrop till `malloc()` och `free()`.
- `Hittar även accesser till oinitierat minne`

[DrMemory-exempel]

Nästa lektion

- Mer pekare och arrayer
  - Dubbelpekare
  - Arrayer av arrayer
- Pekare till portar
  - absolutadressering
- Pekare till funktioner
- Structs (sammansatta datatyper)
  - Länkade listor