

Föreläsning 2

Tobias Wrigstad

Kursansvarig

*Grundläggande datatyper,
deklaration, uttryck och satser*



Vad är imperativ programmering?

C

- Maskinnära språk

Resurskritiska applikationer, hårdvarunära programmering, effektivitet

- Skapades ca 1969, användes för att implementera UNIX

- Språk som kan ersätta C: C++, D, Go, Java, Rust

På denna kurs använder vi C för att det inte gömmer komplexitet

Några skillnader mellan C och Haskell

- C är imperativt och eager ("ivrigt"), Haskell är funktionellt och lazy
- Språken tillhör olika syntaxfamiljer
- C är manifest typat: alla variabler måste ges en explicit typ av programmeraren
- C är svagt typat: vissa typomvandlingar görs automatiskt och okontrollerade brutala typomvandlingar tillåts
- C har ingen list-typ
- I C kan man arbeta direkt med minnesadresser (pekare)
- Minneshanteringen i C måste ofta göras explicit
- Det görs vanligen ingen runtime-kontroll när C-program exekverar (vild adressering, arraygränser, odefinierade variabelvärden . . .)

```
#include<stdio.h>

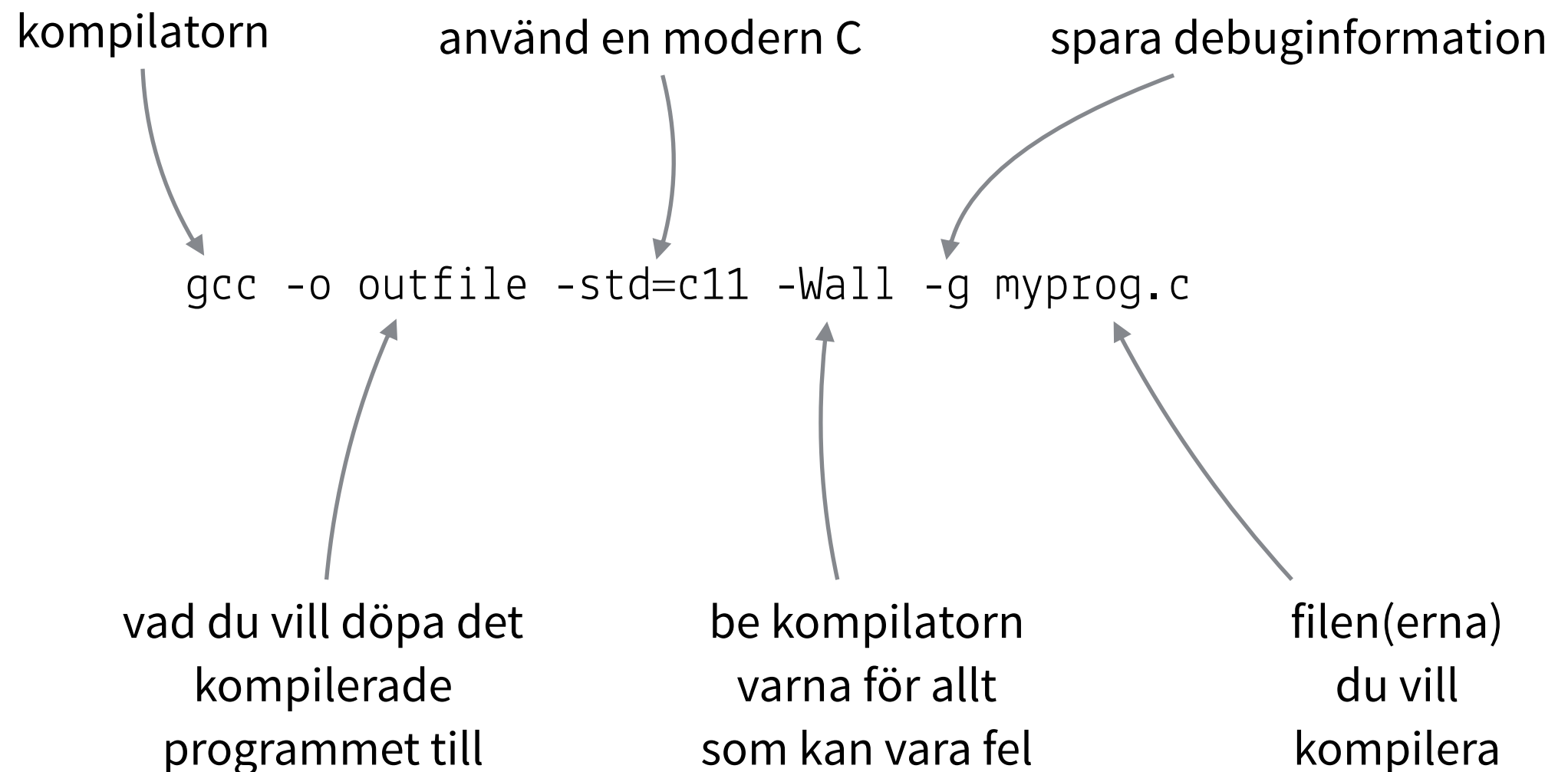
int main(int argc, char *argv[])
{
    puts("Hello, world!");
    return 0;
}
```

../L1/hello.c

```
$ ./hello
Hello, world!
$ _
```



Kompilera ditt program



Kör ditt program: `./outfile`

Variabeldeklaration

Syntax: `typ variabelnamn;`
 `typ variabelnamn = expr;`

Exempel: `int age;`
 `int age = 42;`

- Variabler är symboliska namn för värden


Namnet är extremt viktigt för det ger mening för programmeraren

Variabers värde kan **förändras**

Oinitierade variablers värden är **odefinierade**

Tilldelning till variabler

Syntax:

variabelnamn = *expr*; 

hela uttrycket har typ T

måste ha rätt typ T

Exempel:

age = 100; // Tilldela 100 till variabeln age

age = age + 1; // Öka variabelns värde med 1

total = age = age + 1; // OK, men vansinne

Datatyper [de vanligaste för nu]

Vanliga datatyper		
	Beskrivning	Storlek
char	Ett tecken	Minst 8 bitar
short	Litet heltal	Minst 16 bitar
int	Heltal	Minst 16 bitar
long	Stort heltal	Minst 32 bitar
float	Litet flyttal	Ospecificerat
double	Stort flyttal	Minst som float
bool	Sedan C99	[true, false]

↑
Kräver biblioteket `stdbool.h`

*Storlekarna är
beroende av vilken
hårdvara
programmet är
kompilerat på/för.*

```

#include <stdbool.h>
#include <stdio.h>

int main(void)
{
    printf("bool           %zd\n", sizeof(bool));
    printf("char           %zd\n", sizeof(char));
    printf("short          %zd\n", sizeof(short));
    printf("int            %zd\n", sizeof(int));
    printf("long           %zd\n", sizeof(long));
    printf("long long      %zd\n", sizeof(long long));
    printf("float          %zd\n", sizeof(float));
    printf("double         %zd\n", sizeof(double));
    printf("long double    %zd\n", sizeof(long double));
    return 0;
}

```

data-type-sizes.c

```

$ ./data-type-sizes
bool           1
char           1
short          2
int            4
long           8
long long      8
float          4
double         8
long double    16
$ _

```



Operatörer

Aritmetik	
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
%	Modulo
++	Inkrementera
--	Dekrementera

Relationer	
==	Likhet
!=	Olikhet
<	Strikt mindre än
<=	Mindre än
>	Strikt större än
=>	Större än

Logik	
&&	Och
	Eller
!	Negation

Plus bitoperatorer — vi återkommer till dem senare i kursen

Många olika varianter av tilldelning

Kortform	Långform	Kommentar
age += 1	age = age + 1	
age -= 1	age = age - 1	
age++	tmp = age; age = age + 1; tmp	<i>Vanlig felkälla!</i>
++age	age = age + 1	
age--	tmp = age; age = age - 1; tmp	<i>Vanlig felkälla!</i>
--age	age = age - 1	
age /= 2	age = age / 2	
age *= 2	age = age * 2	

Villkorssatser (conditionals)

Syntax:

```
if (expr) { expr; }
```

```
if (expr) { expr; } else { expr; }
```

```
expr ? expr : expr
```

Exempel:

```
if (age > 100) { puts("Very old"); }
```

```
if (age % 2 == 0) { puts("Even"); } else { puts("Odd"); }
```

```
a < b ? b : a;
```

- Den vanligaste formen av villkorssats returnerar inget värde
- Den något kryptiska `?:`-formen har returvärde

Läsbarhet och frihet [alla dessa är semantiskt ekvivalenta]

```
if (age > 100) { puts("Very old"); }
```

```
if (age > 100)
{
    puts("Very old");
}
```

```
if (age > 100) {
    puts("Very old");
}
```

```
if (age > 100) puts("Very old");
```

```
if (age > 100)
    puts("Very old");
```

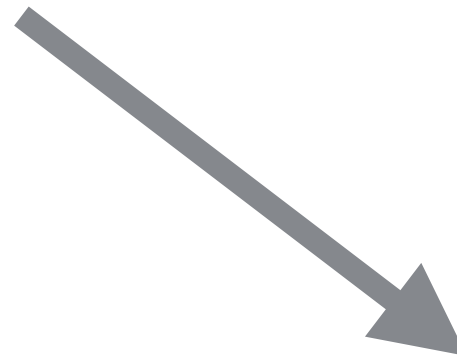
Läsbarhet och frihet

```
puts(age > 100 ? "Very old" : "");
```

Läsbarhet: Apples #gotofail SSL bug [1/2]

```
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;
```

Indenteringen ljuger!



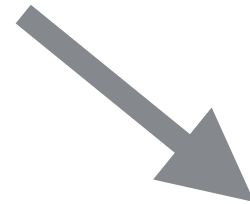
Indenteringen lyfter fram felet!

```
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;
```


Läsbarhet: Apples #gotofail SSL bug [2/2]

```
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;
```

Inga block — fail!



Block — inget fail!

```
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
{
    goto fail;
}
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
{
    goto fail;
    goto fail;
}
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
{
    goto fail;
}
```

Switchsatser

Syntax: **switch** (*expr*)
 {
 case *literal* *body*; **break**;
 default: *body*;
 }

Exempel: **switch** (n)
 {
 case 0: result = 0; **break**;
 case 1: result = 1; **break**;
 default: result = fib(n - 1) + fib(n - 2);
 }

- Vanlig felkälla — bevisat dålig design

Exempel på en trasig switchsats

```
switch (n)
{
  case 0:  result = 0;
  case 1:  result = 1;
  default: result = fib(n - 1) + fib(n - 2);
}
```

- Vad händer om $n == 1$?

Iteration med loopar: while

Syntax:

```
while (cond) { body }
```

```
while (cond) expr;
```

*Om sant, gå ett
"till varv" i loopen*

*"Loop-kroppen" — det
som körs varje "varv"*

Exempel:

```
int n_fakultet = 1;  
int n = 6;
```

```
while (n >= 1)  
{  
    n_fakultet *= n;  
    n = n - 1;  
}
```

```
printf("%d! = %d\n", n, n_fakultet);
```

Loopar är bekväma och nödvändiga

```
// n == 6  
while (n)  
    n_fakultet *= n--;
```

← *Vad du skrev*

```
n_fakultet *= n--;  
n_fakultet *= n--;  
n_fakultet *= n--;  
n_fakultet *= n--;  
n_fakultet *= n--;  
n_fakultet *= n--;
```

*Vad kompilatorn gjorde (unrolling)
(bara möjligt om $n == 6$ går att avgöra)*

←

Loopar är bekväma och nödvändiga

```
// n == 6  
while (n)  
    n_fakultet *= n--;
```

← *Vad du skrev*

```
n_fakultet *= 6;  
n_fakultet *= 5;  
n_fakultet *= 4;  
n_fakultet *= 3;  
n_fakultet *= 2;  
n_fakultet *= 1;
```

*Vad kompilatorn gjorde (unrolling)
(bara möjligt om $n == 6$ går att avgöra)*

←

Läsbarhet [identiska satser enligt kompilatorn]

```
while (n >= 1)
{
    n_fakultet *= n;
    n = n - 1;
}
```

```
while (n)
{
    n_fakultet *= n--;
}
```

```
while (n) n_fakultet *= n--;
```

```
while (n)
    n_fakultet *= n--;
```

Kort utvikning: do-while

```
do
{
    n_fakultet *= n;
}
while (n--);
```


Iteration med loopar: for

Syntax:

```
for (init; pre; post) { body }
```

```
for (init; pre; post) expr;
```

*Deklarera och initiera
loopvariabler*

*Om sant, gå ett
"till varv" i loopen*

*Utförs alltid sist
i varje varv*

Exempel:

```
int n_fakultet = 1;  
for (int i = 1; i <= n; i = i + 1)  
{  
    n_fakultet *= i;  
}  
  
printf("%d! = %d\n", n, n_fakultet);
```

Main – där alla C-program börjar

*Antalet kommandorads-
argument*

*De faktiska
kommandorads-
argumenten*

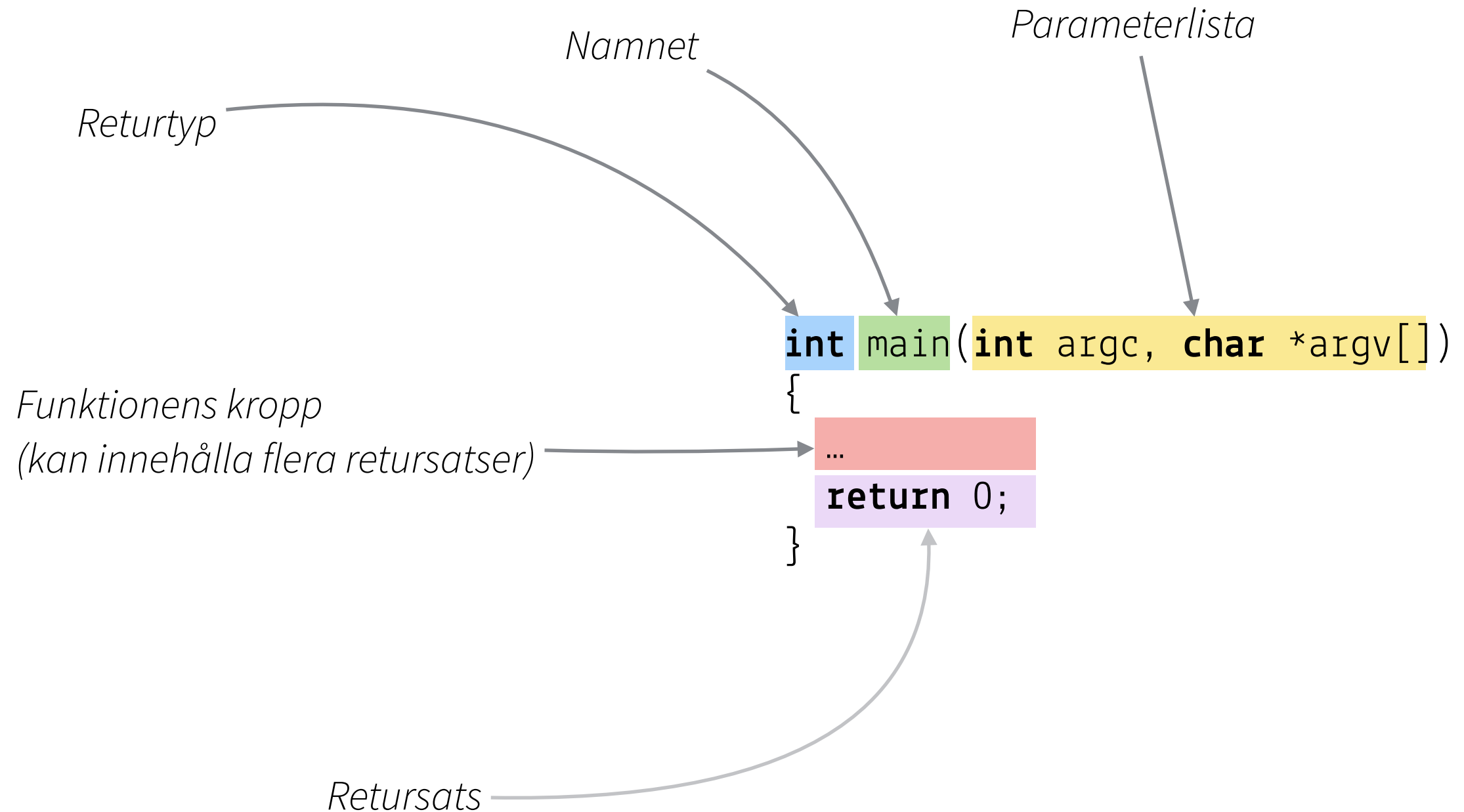
```
int main(void)
{
    ...
    return 0;
}
```

OK

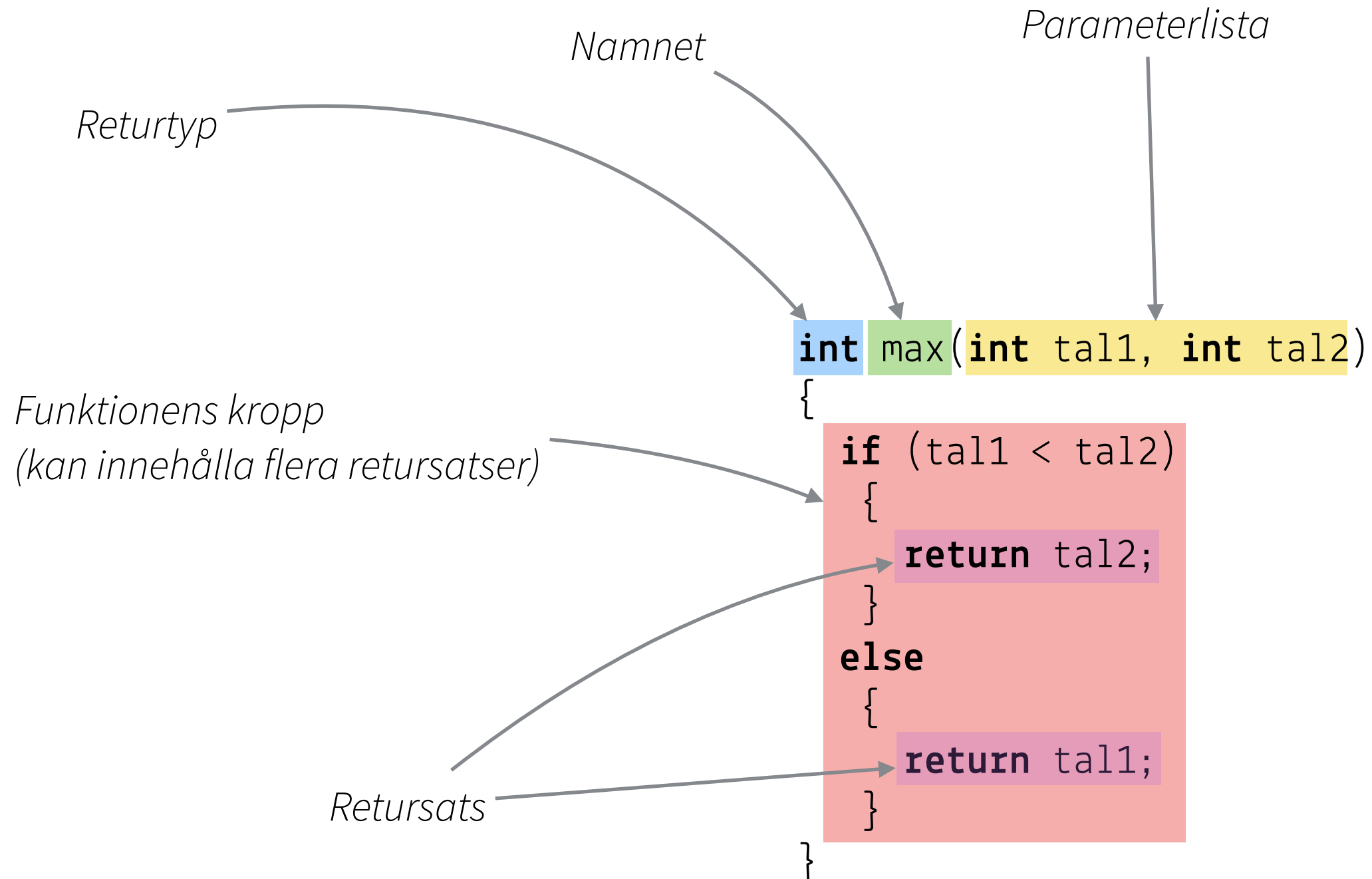
```
int main(int argc, char *argv[])
{
    ...
    return 0;
}
```

Bättre

Funktionens anatomi



Deklarera egna funktioner



Deklarera egna funktioner

// Exempel på anrop

```
int a = max(512, 1024);
```

```
int max(int tal1, int tal2)
{
    if (tal1 < tal2)
    {
        return tal2;
    }
    else
    {
        return tal1;
    }
}
```

Inkludera funktioner från andra bibliotek

#include <filnamn.h> ← Inkludera från standardbibliotek

#include "filnamn.h" ← Inkludera från ditt eget program

Plus extra länkning i kompileringssteg. Vi återkommer till det senare.

Exempel på olika funktioner

Funktion	Kommentar
void puts(char *)	Skriv ut en sträng på skärmen
int atoi(char *)	Konvertera en sträng till ett heltal (int)
long atol(char *)	Konvertera en sträng till ett heltal (long)
int getchar()	Läs in ett tecken från tangentbordet
FILE *fopen(char *, char *)	Öppna en fil

Läs mer om funktionerna med hjälp av man-kommandot

Läsbarhet [identiska funktioner enligt kompilatorn]

```
int max(int tal1, int tal2)
{
    if (tal1 < tal2)
    {
        return tal2;
    }
    else
    {
        return tal1;
    }
}
```

```
int max(int tal1, int tal2)
{
    return (tal1 < tal2) ? tal2 : tal1;
}
```

```
int max(int tal1, int tal2)
{
    if (tal1 < tal2) return tal2;
    return tal1;
}
```


Returns kontrollflöde [vanlig felkälla]

```
int max(int tal1, int tal2)
{
    if (tal1 < tal2)
    {
        return tal2;
    }
    else
    {
        return tal1;
    }

    puts("Jag skrivs aldrig ut!");
}
```

Program som skriver ut kommandoradsargument

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

int main(int argc, char *argv[])
{
    printf("%d kommandoradsargument\n", argc);
    for (int i = 0; i < argc; ++i)
    {
        printf("Argument %d = %s\n", i, argv[i]);
    }
    return 0;
}
```

cl-args.c



Komplett exempel för n-fakultet

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

int main(int argc, char *argv[])
{
    int nfak = 1;
    int n = atoi(argv[1]);
    for (int i = 1; i <= n; ++i)
    {
        nfak *= i;
    }
    printf("%d! = %d\n", n, nfak);
    return 0;
}
```

nfak.c



Arrayer

`T name[size];` // deklarerar en array av *size* element av typen T

```
int salaries[500];
```

```
long sum = 0;
```

```
for (int i = 0; i < 500; ++i)
```

```
{
```

```
    sum += salaries[i];
```

```
}
```

läs det i:te elementet i arrayen



- Arrayer har en fix storlek – kan inte ändras
- Arrayer indexeras $[0, size)$ – första elementet har index 0, sista $size-1$

Skriv: `myarr[17] = 42;` Läs: `myarr[x]`

- Arrayerna har inget metadata, och C gör ingen indexkontroll

Ingen indexkontroll

```
int salaries[500];  
long sum = 0;  
  
for (int i = 0; i <= 500; ++i)  
{  
    sum += salaries[i];  
}
```

Vad blir resultatet av detta program när det körs?

Ingen indexkontroll

```
int salaries[500];  
long sum = 0;  
  
for (int i = 0; i <= 500; ++i)  
{  
    sum += salaries[i];  
}
```

**UNDEFINED
BEHAVIOUR**

Vad blir resultatet av detta program när det körs?

Kommentarer

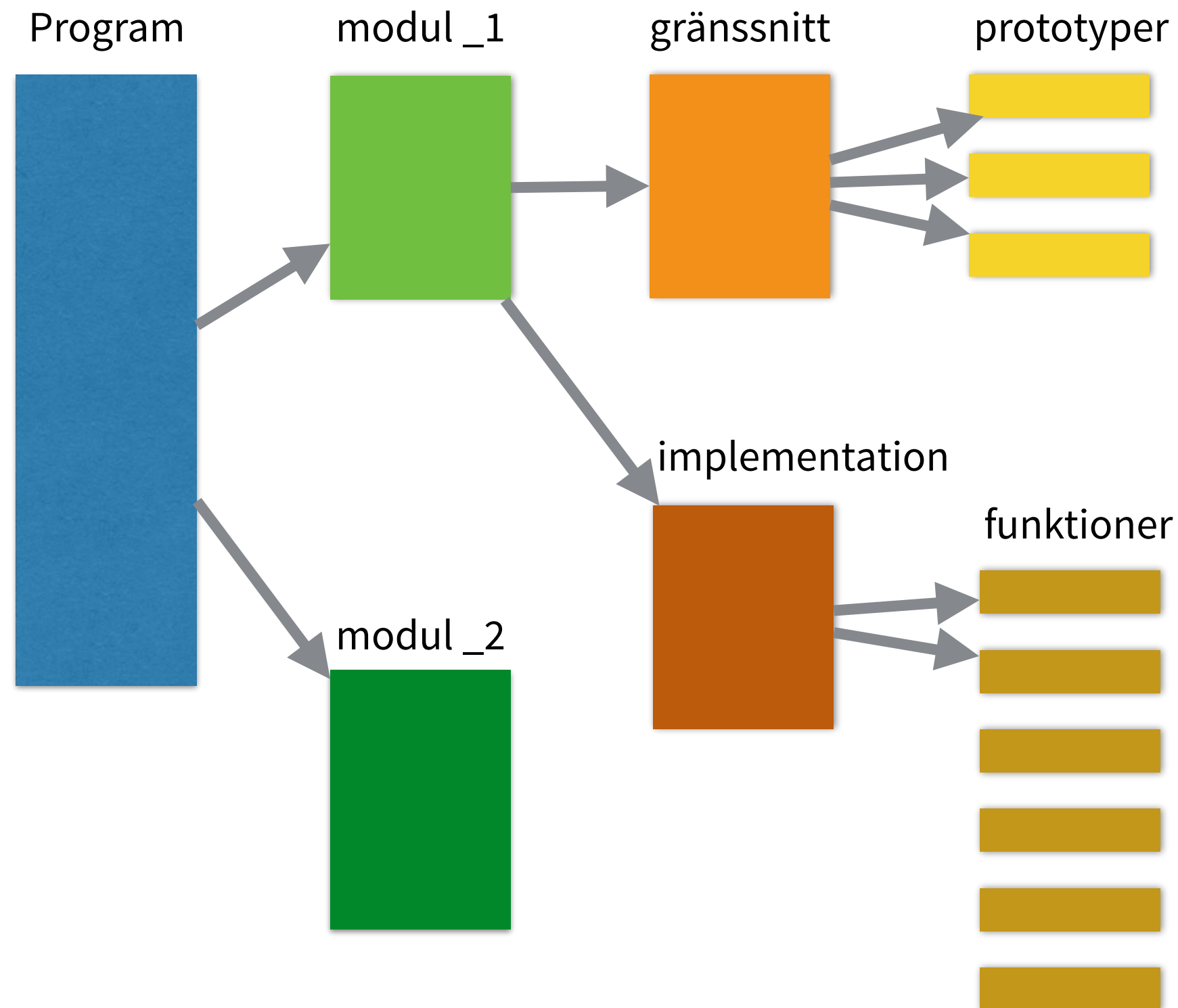
```
// Startar kommentar som gäller till radens slut
```

```
/* Startar kommentarblock som gäller ända till */
```

Kommentarer är mest nödvändiga för att förklara *varför*.

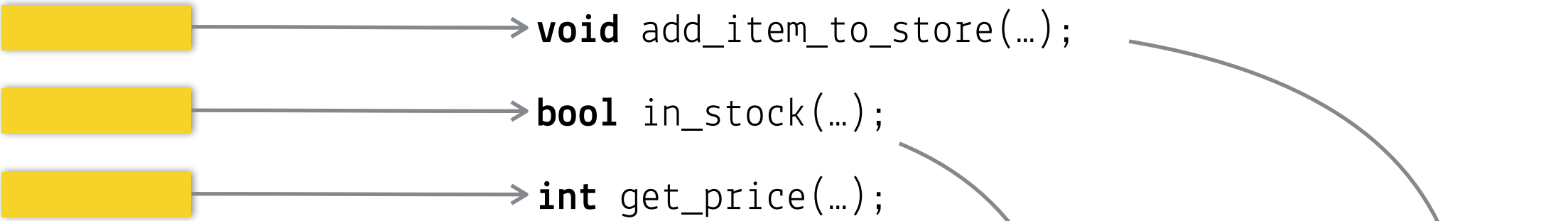
Om du känner att du behöver kommentera en bit kod för att den skall gå att förstå är det 99% chans att koden borde skrivas om istället för kommenteras.

Ett programs anatomi

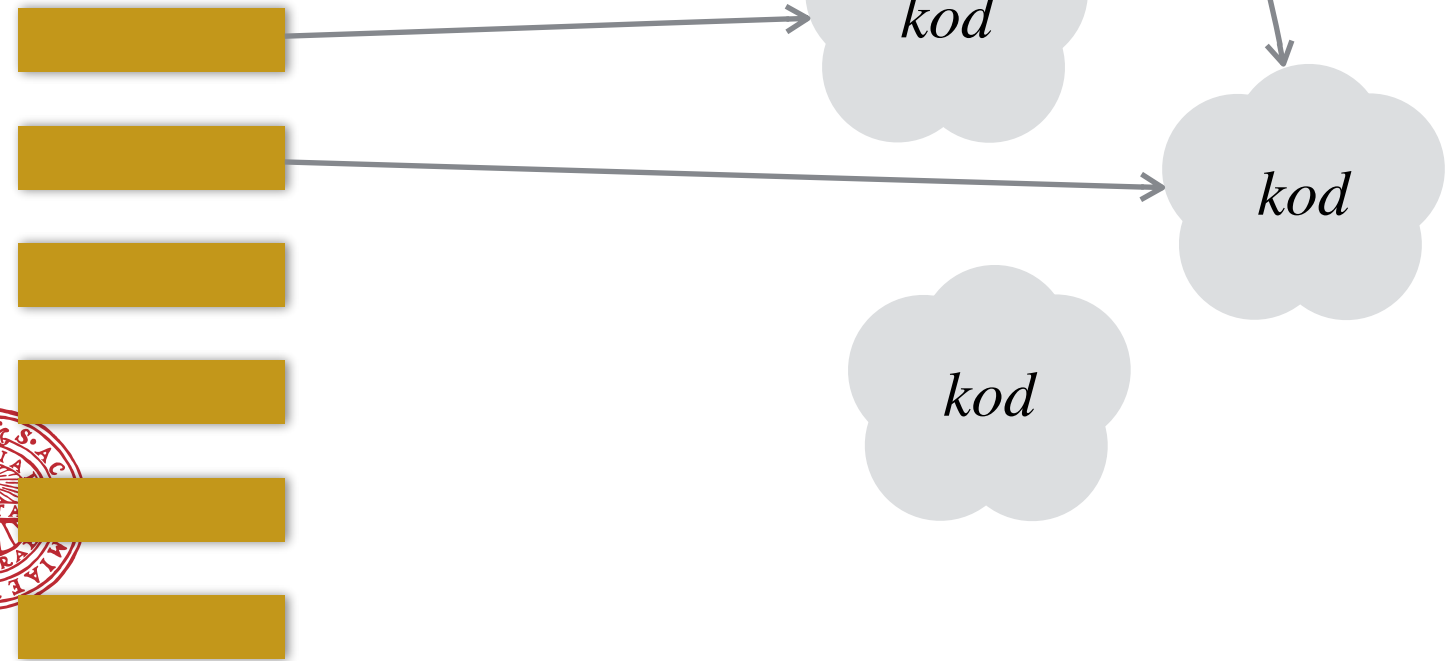


Funktionsabstraktionen

prototyper

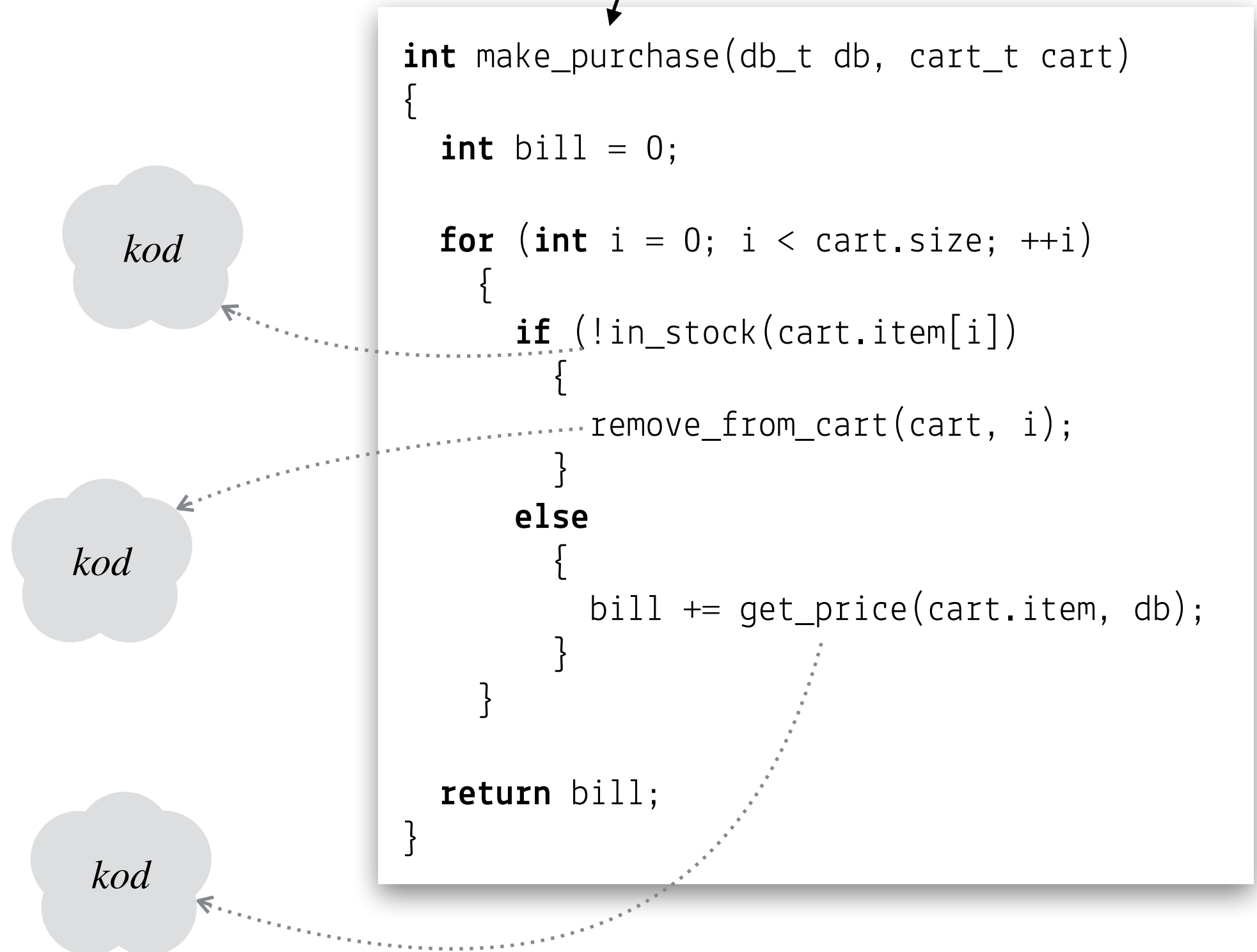


funktioner



Funktionsabstraktionen

Bygga abstraktioner av abstraktioner!



Demonstration: Fibonacci

- Rekursiv vs. imperativ implementation
- Jämförelse av körtid

Utökning med memoisering

- Liten verktygspresentation

gcc

man

valgrind

time

(cflow)

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

uint64_t fib(const int n)
{
    switch (n)
    {
        case 0: return 0;
        case 1: return 1;
        default: return fib(n - 1) + fib(n - 2);
    }
}

int main(int argc, char *argv[])
{
    if (argc < 2)
    {
        puts("Usage: ./fib <n>");
    }
    else
    {
        const int n = atoi(argv[1]);
        printf("fib(%d) = %zd\n", n, fib(n));
    }

    return 0;
}
```

fib-rec.c



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

uint64_t fib(const int n)
{
    uint64_t acc1, acc2, temp;

    acc1 = 0;
    acc2 = 1;

    for (int i = 0; i < n; ++i)
    {
        temp = acc2;
        acc2 += acc1;
        acc1 = temp;
    }

    return acc1;
}
```

```
int main(int argc, char *argv[])
{
    if (argc < 2)
    {
        puts("Usage: ./fib <n>");
    }
    else
    {
        const int n = atol(argv[1]);
        printf("fib(%d) = %zd\n", n, fib(n));
    }

    return 0;
}
```

fib-iter.c



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

int64_t fib(const int n, int64_t *memo)
{
    if (n == 0)    return 0;
    if (n == 1)    return 1;

    if (!memo[n])
    {
        memo[n] = fib(n - 1, memo) + fib(n - 2, memo);
    }

    return memo[n];
}

int main(int argc, char *argv[])
{
    if (argc < 2)
    {
        puts("Usage: ./fib <n>");
    }
    else
    {
        const int n = atoi(argv[1]);
        int64_t memo[128];
        assert(0 <= n && n < 128);

        printf("fib(%d) = %zd\n", n, fib(n, memo));
    }

    return 0;
}
```

fib-rec-memo.c

