

IOOPM 2019

Lars-Henrik Eriksson
Kursansvarig

Välkommen, nu kör vi!



Föreläsning 1

Lars-Henrik Eriksson

Kursansvarig

Föreläsningsbilderna är baserade på bilder gjorda av Tobias Wrigstad.

*Kursintro,
Imperativ programmering och C,
grundläggande datatyper
deklaration, uttryck och satser*



Imperativ- och objektorienterad programmeringsmetodik (IOOPM)

- Du kan ”tänka programmering” — efter PKD

Funktionell programmering

Algoritmer, datastrukturer

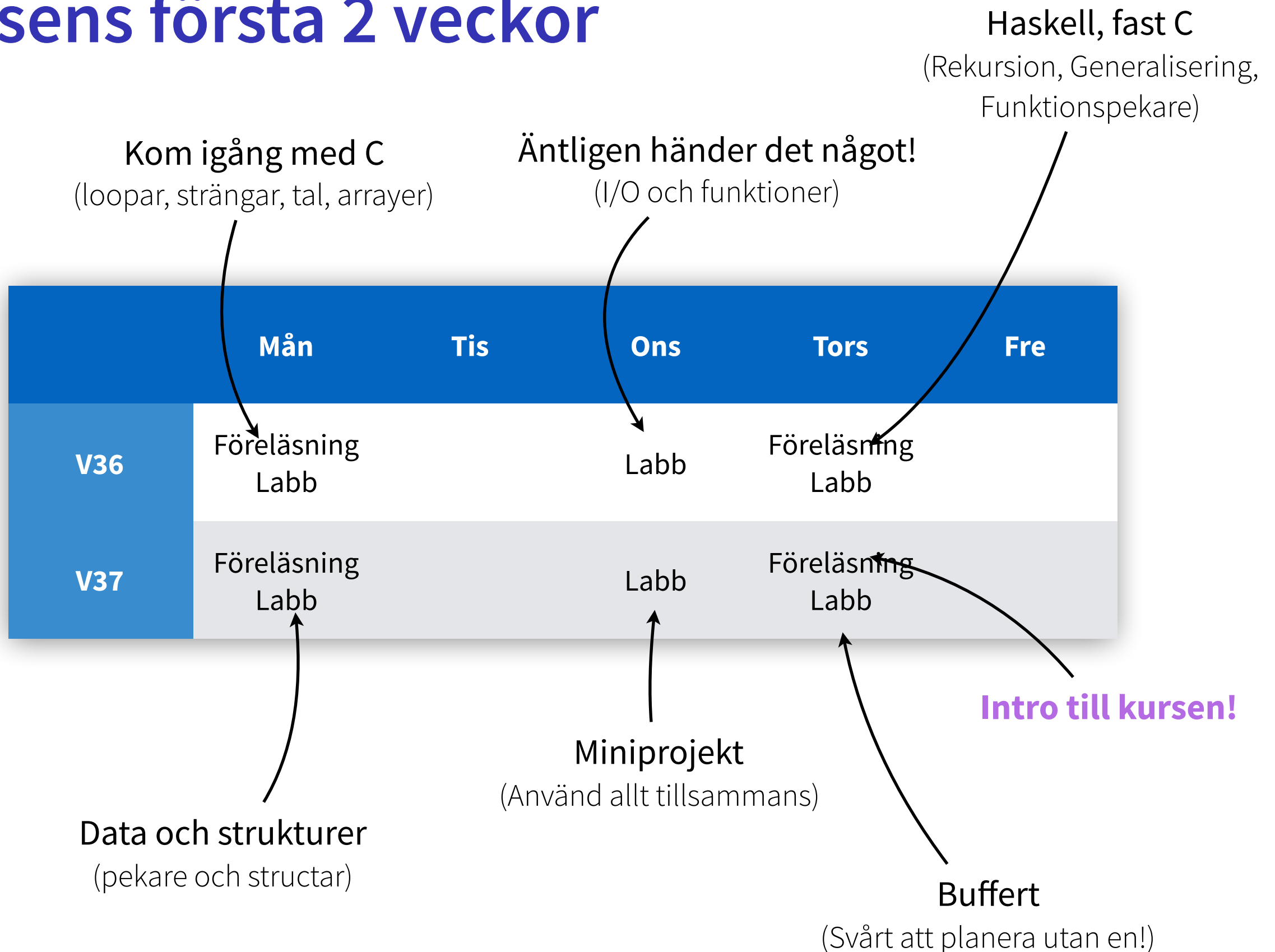
- Denna fortsättningskurs fokuserar på de två vanligaste paradigmen — imperativ & objektorienterad (OOP)

Vi skall lära oss delar av C och Java, skriva ett par 1000 rader kod (förhoppningsvis)

Denna vecka och nästa har vi 5 labbar i C i syfte att du skall komma igång

Mer information om kursuplägget kommer i föreläsning 4

Kursens första 2 veckor



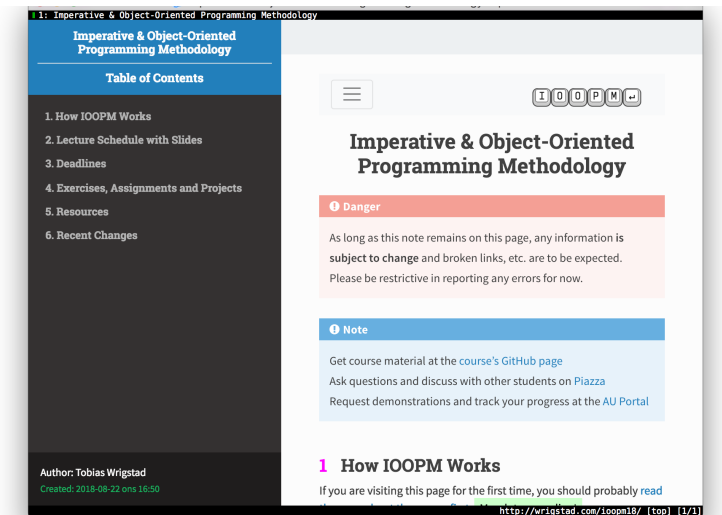
IOOPM i siffror

- ~30 föreläsningar
- ~30 labbar
- 4 inlämningsuppgifter
- 1 projektuppgift
- 2 kodprov
- 30–70 olika delmål som skall redovisas var för sig, ca 2–3 i veckan
- ~18 assistenter
- ~12 timmar schemalagd undervisning varje vecka — **minst** lika mycket till krävs
- etc. etc.

20 HP = 533 arbetstimmar

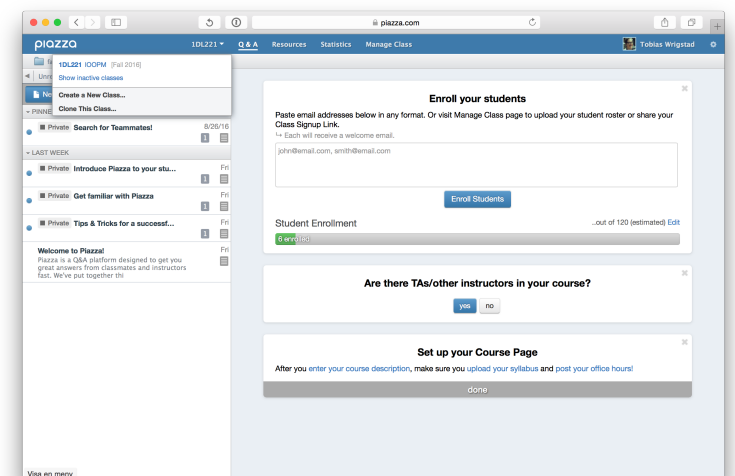
All information, kursmaterial

wrigstad.com/ioopm19



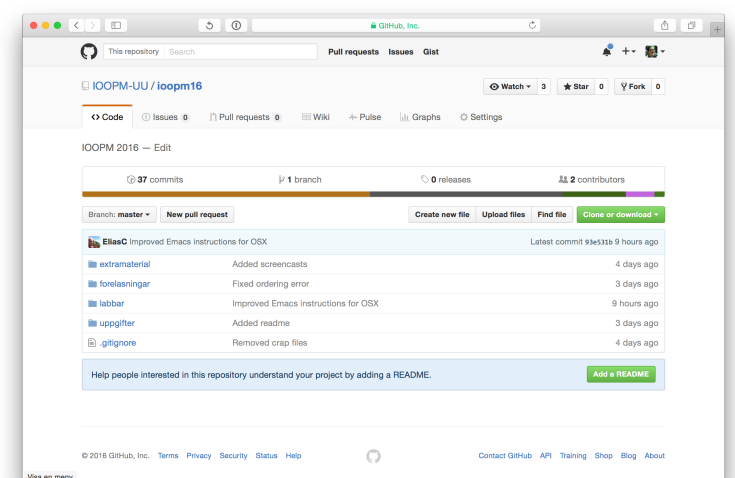
Diskussionsforum, handledning, enkäter, etc.

piazza.com/class/k01yr6gds9m62i



Utdelad kod, andra resurser

github.com/tobiaswrigstad/ioopm19



Vad är imperativ programmering?

Imperativ vs. funktionell programmering

- Funktionell programmering:

Beräkningen av ett uttryck utförs **i ett svep**.

Det spelar i princip **ingen roll i vilken ordning** olika delar av programmet beräknas.

Funktioner får data genom sina argument och returnerar data som i sin tur blir data till andra funktioner – "**dataflöde**" / "**dataflow**".

Data **ändras aldrig**, men nya data kan beräknas och gamla "försvinna".

- Imperativ programmering:

Beräkningen utförs i **diskreta steg** ("satser") som utförs i **en viss ordning**.

Data som beräknas i ett steg **lagras** i variabler eller i datastrukturer för användning i följande steg.

Variabler och datastrukturer kan **ändras**.

Det imperativa programspråket C

- Maskinnära språk (för någon maskin iaf)

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 abstraktioner och komplexitet

(Och för att en del personer tycker att det är underbart och fantastiskt!)



Några skillnader mellan C och Haskell

- C är **imperativt** och **eager** ("ivrigt"), Haskell är funktionellt och lazy
- Språken tillhör olika **syntax**familjer
- C är **manifest typat**: alla variabler måste ges en explicit typ av programmeraren (ingen typhärledning)
- C är **svagt typat**: vissa typomvandlingar görs automatiskt och okontrollerade brutala typomvandlingar tillåts
- C är betydligt mer **låg-nivå**:
 - t.ex. C har ingen list-typ, man måste koda sina listor själv
 - man kan arbeta direkt med minnesadresser (pekare)
 - Minneshanteringen i C måste ofta göras explicit
 - Det görs vanligen **ingen** runtime-**kontroll** när C-program exekverar (vild adressering,

```
#include<stdio.h>

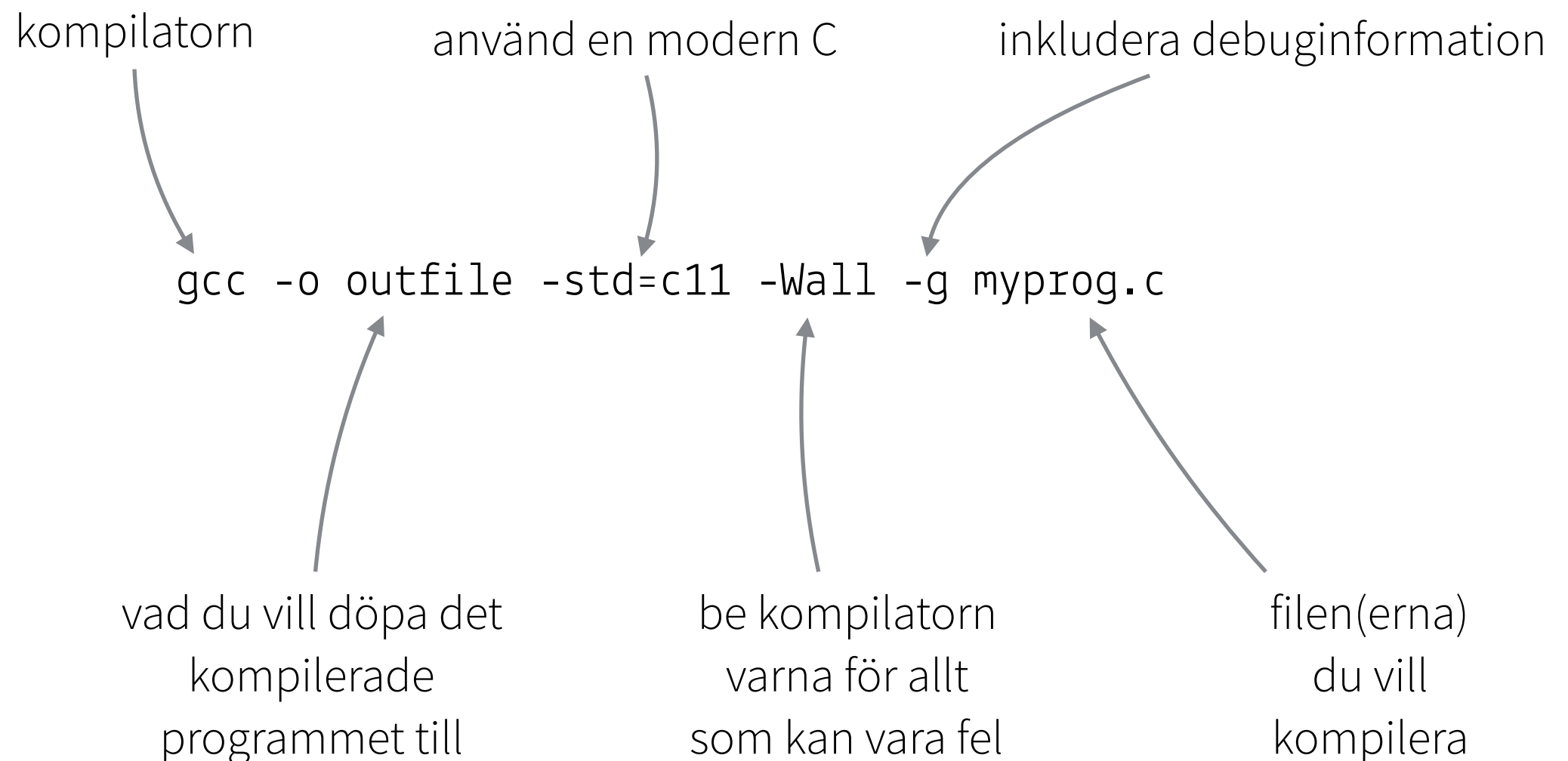
int main(int argc, char *argv[])
{
    puts("Hello, world!");
    return 0;
}
```

hello.c

```
$ gcc hello.c
$ ./a.out
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*** (sidoeffekt)

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
void	Ingenting!	<i>n/a</i>
bool	Sedan C99	[true, false]

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

Kräver biblioteket `stdbool.h`

```

#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

```

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

```



Operatorer

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 (motsvarar **if** i Haskell)

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");
}
```

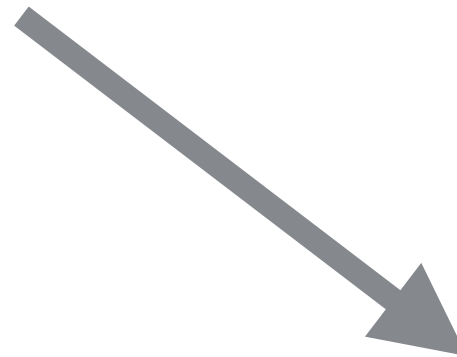
```
if (age > 100) puts("Very old");
```

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


Läsbarhet: 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;
```

<https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-1266>

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:  puts("n=0"); break;  
  case 1:  puts("n=1"); break;  
  default: puts("n<0 or n>1");  
}
```

- Vanlig felkälla — bevisat dålig design

Exempel på en trasig switchsats

```
switch (n)
{
  case 0: puts("n=0");
  case 1: puts("n=1");
  default: puts("n<0 or n>1");
}
```

- 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

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

OK

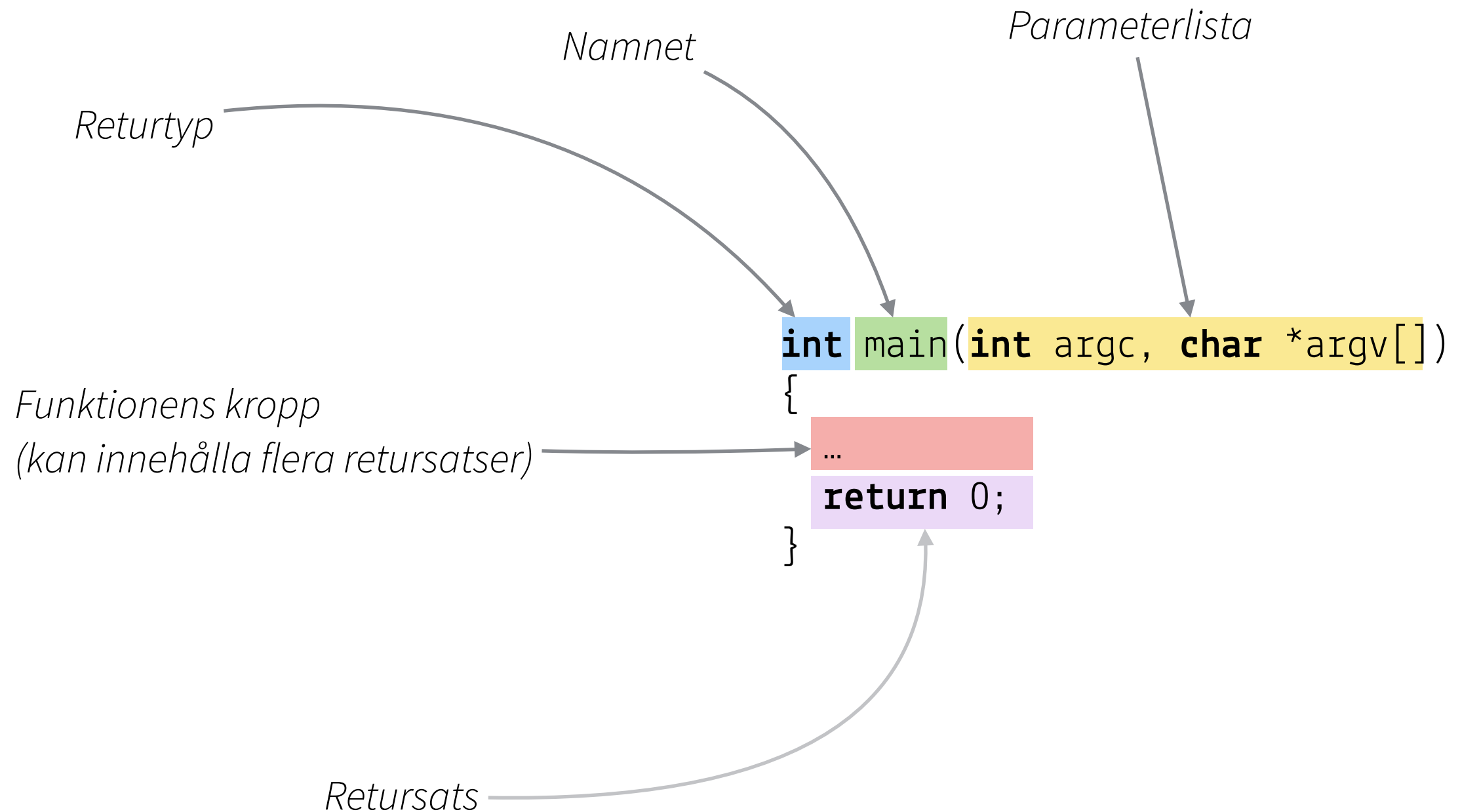
*Antalet kommandorads-
argument*

*De faktiska
kommandorads-
argumenten*

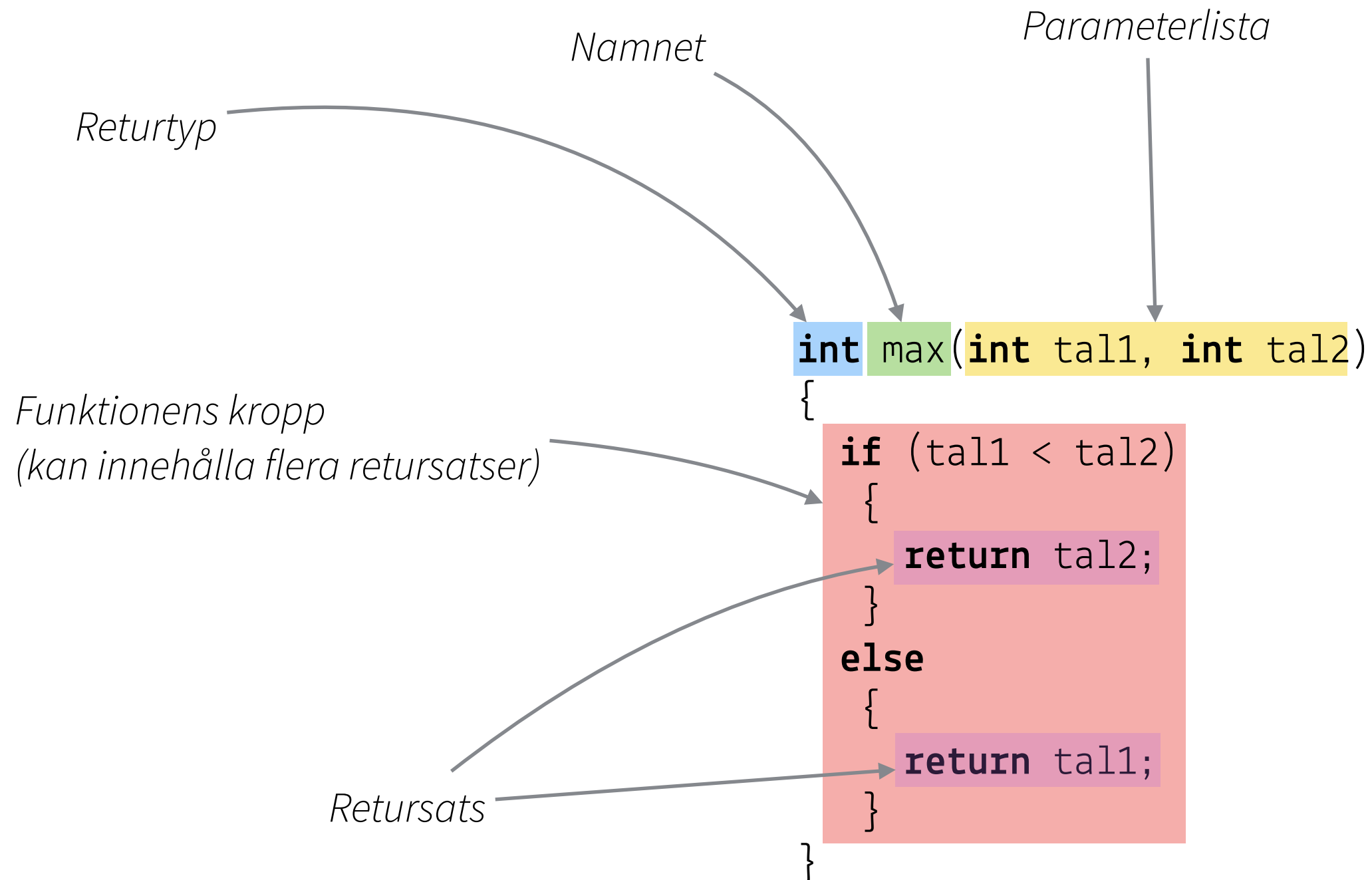
```
int main(int argc, char *argv[])
{
    ...
    return 0;
}
```

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 dina egna filer

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;
}
```

echo.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;
}
```

n-fakultet.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

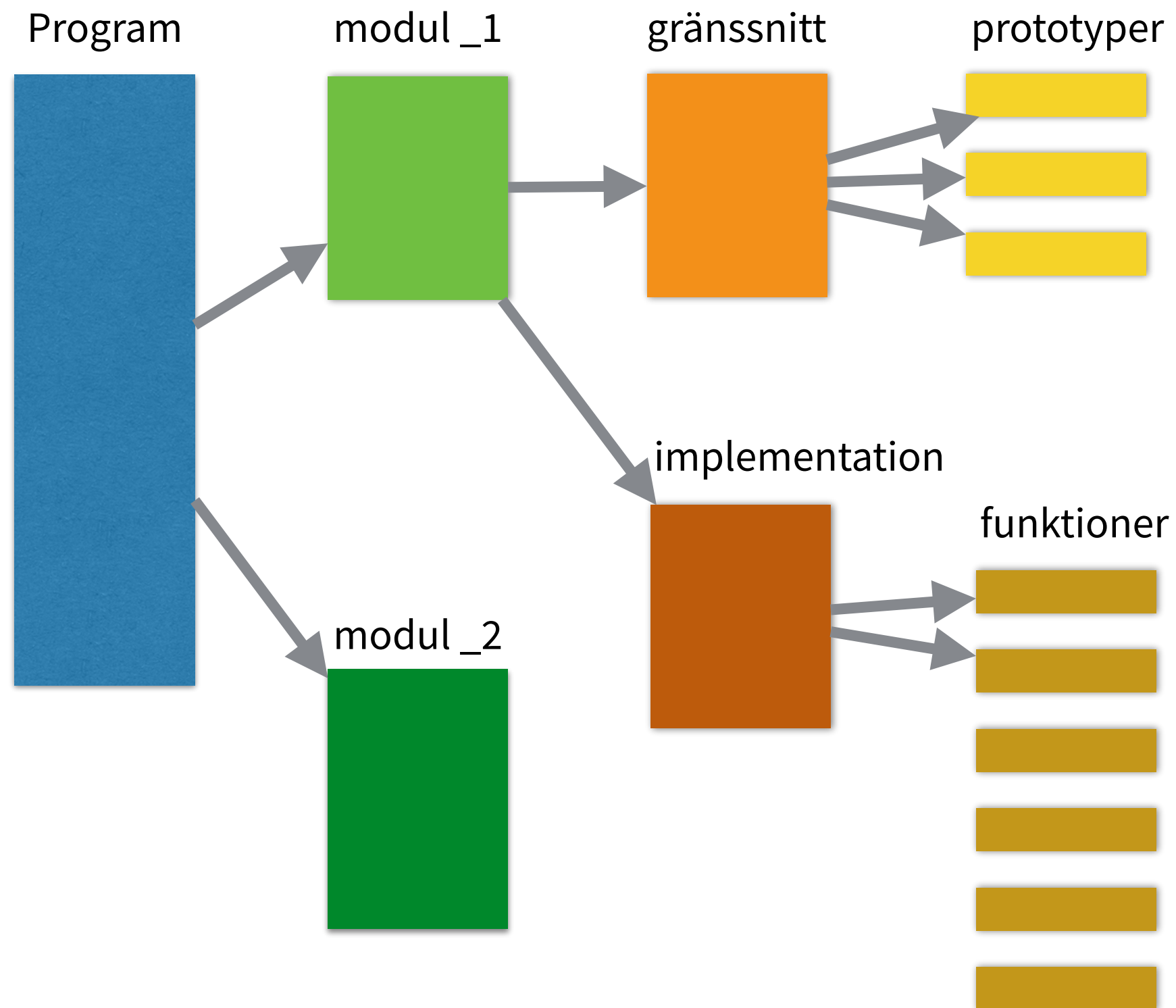
`// Startar kommentar som gäller till radens slut`

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

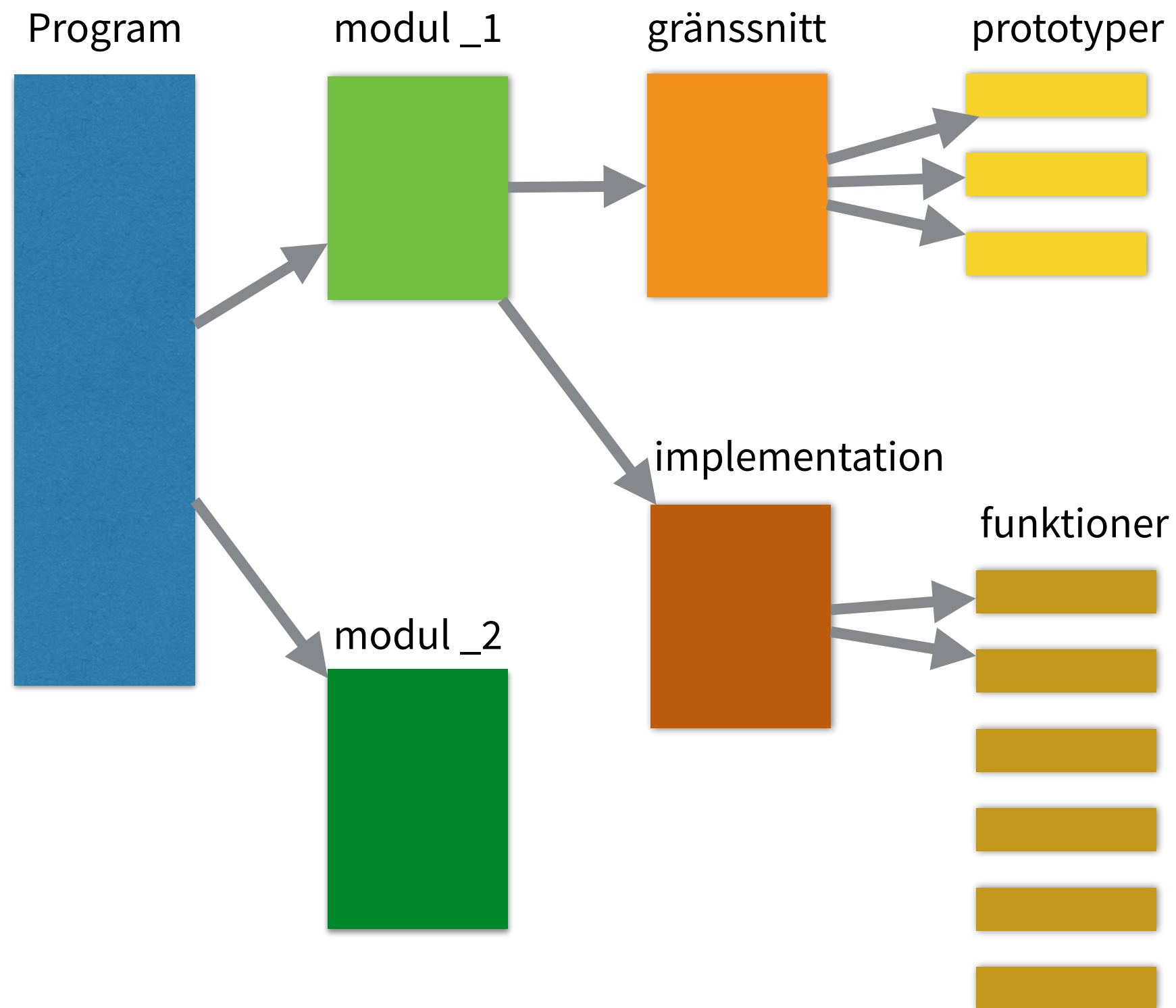
Kommentarer är mest nödvändiga för att förklara varför.

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

Ett programs anatomi

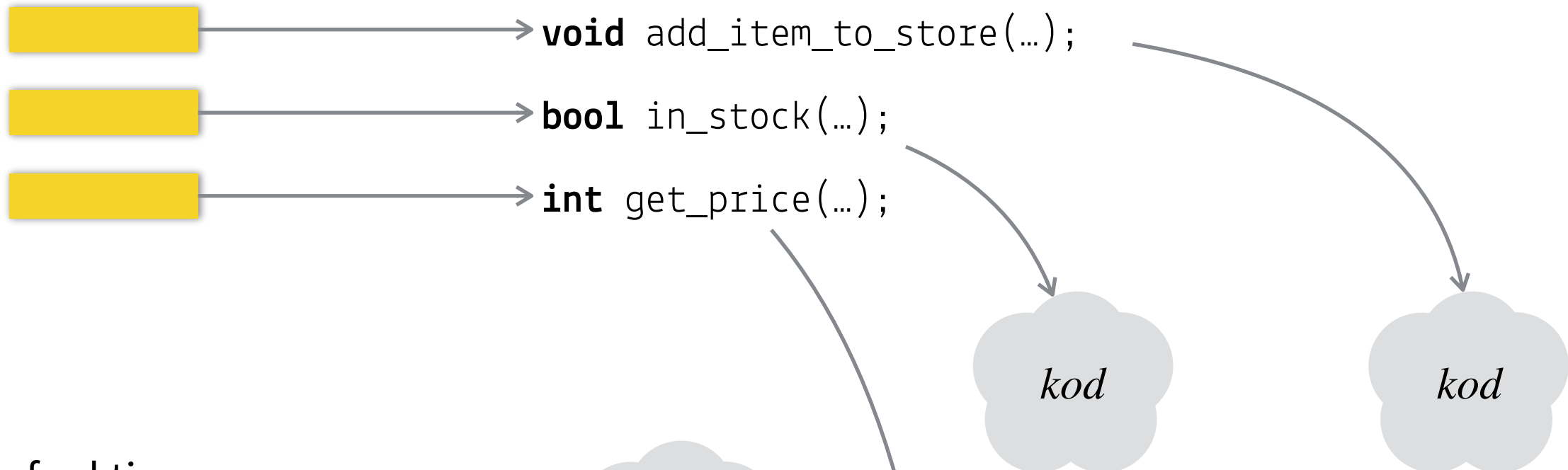


Ett programs anatomi

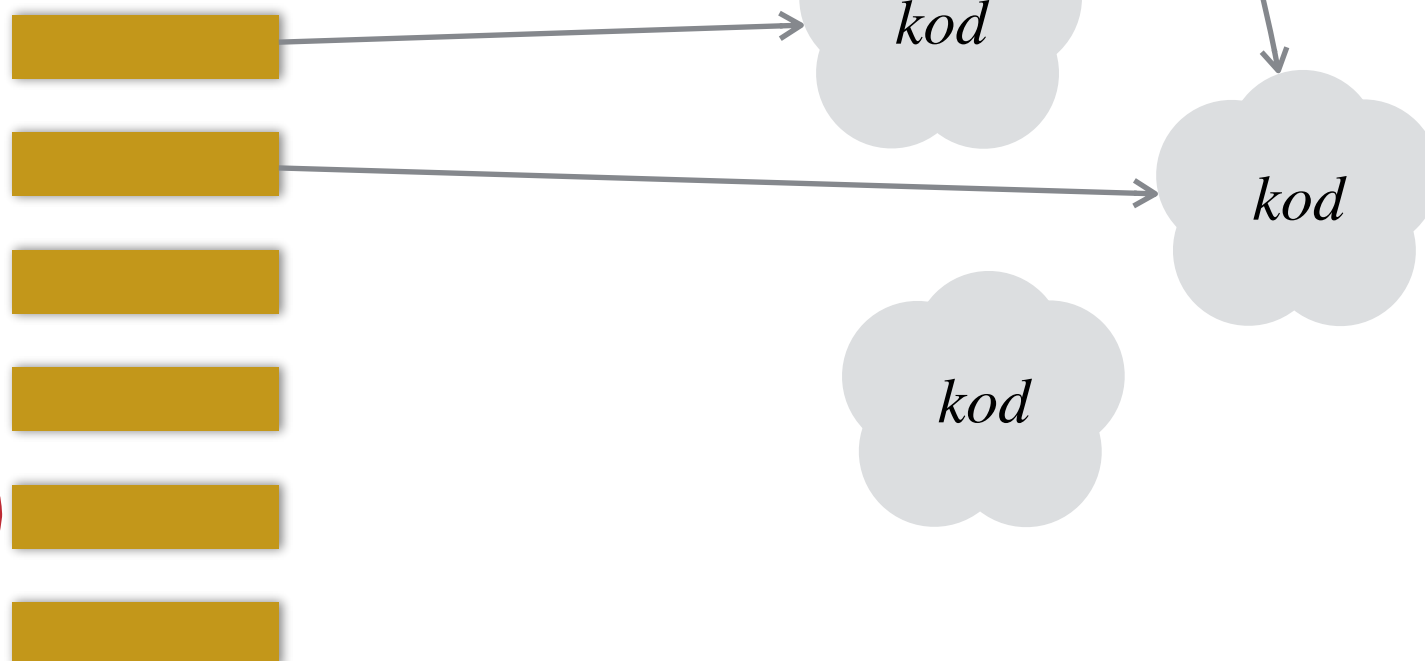


Funktionsabstraktionen

prototyper

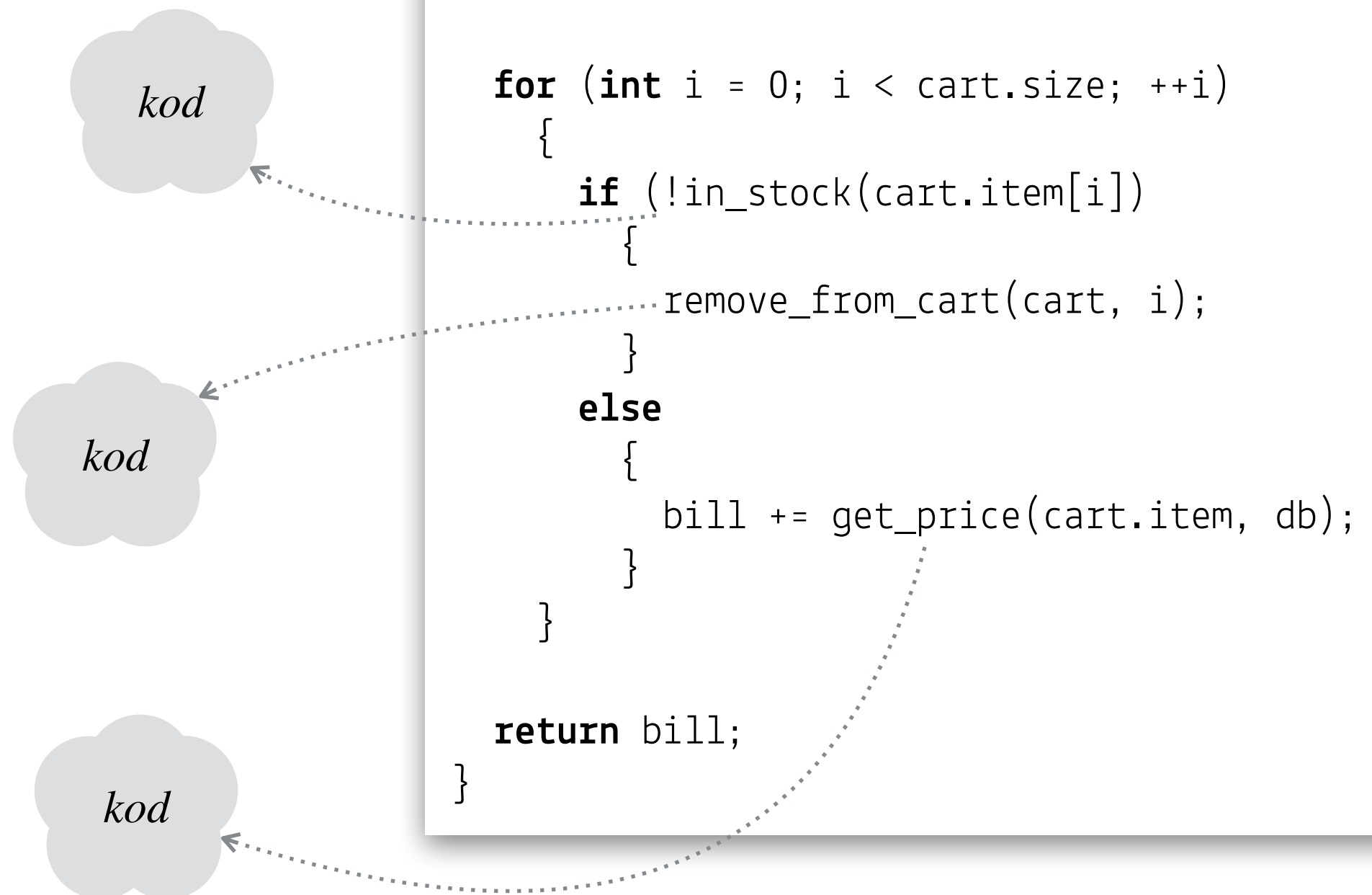


funktioner



Funktionsabstraktionen

Bygga abstraktioner av abstraktioner!



Föreläsning 1 extra

Gås endast igenom kursivt på föreläsningen och i mån av tid — dock bra att ha tillgängligt inför lab 2 & 3

*Se även **extramaterialet** i föreläsningsanteckningarna.*

Introduktion till standard-I/O och grundläggande stränghantering



Hur var det i Haskell?

- Char är en egen typ

`'a' :: Char`

`'a' + 2` — *kompilerar ej*

Hur var det i Haskell?

- Char är en egen typ

`'a' :: Char`

`'a' + 2` — *kompilerar ej*

- **type String** = [Char]

`"Hello" == ['H', 'e', 'l', 'l', 'o']`

`'F' : "oo" ++ "Bar!" == "FooBar!"`

Hur var det i Haskell?

- Char är en egen typ

`'a' :: Char`

`'a' + 2` — *kompilerar ej*

- **type String** = [Char]

`"Hello" == ['H', 'e', 'l', 'l', 'o']`

`'F' : "oo" ++ "Bar!" == "FooBar!"`

- Kan du listor kan du strängar! (mer eller mindre)

Tecken i C

- **char** — ett heltal (på *normalt* en byte)
- Ett tecken är en **char** med motsvarande ASCII-värde

```
char c1 = 'a';
```

```
char c2 = 97;      == c1
```

```
char c3 = 'a' + 2; == 'c'
```

```
char c4 = '2' + '2'; == 50 + 50
```

Tecken i C

- **char** — ett heltal (på *normalt* en byte)
- Ett tecken är en **char** med motsvarande ASCII-värde

```
char c1 = 'a';
```

```
char c2 = 97;      == c1
```

```
char c3 = 'a' + 2; == 'c'
```

```
char c4 = '2' + '2'; == 100
```

Tecken i C

- **char** — ett heltal (på *normalt* en byte)
- Ett tecken är en **char** med motsvarande ASCII-värde

```
char c1 = 'a';
```

```
char c2 = 97;      == c1
```

```
char c3 = 'a' + 2; == 'c'
```

```
char c4 = '2' + '2'; == 'd'
```

Tecken i C

- **char** — ett heltal (på *normalt* en byte)
- Ett tecken är en **char** med motsvarande ASCII-värde

```
char c1 = 'a';
```

```
char c2 = 97;      == c1
```

```
char c3 = 'a' + 2; == 'c'
```

```
char c4 = '2' + '2'; == 'd'
```

- Obs! 'a' och 97 är alltså ekvivalenta

'a' är *syntaktiskt socker* för 97

Tecken i C

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

```
int main(void)  
{
```

```
    for (int c = 0; c < 128; c++)  
    {
```

```
        printf("Nr: %d, Char: %c\n", c, c);  
    }
```

```
    return 0;
```

```
}
```

Är c en int (%d) eller ett tecken (%c)?

ascii.c

Strängar i C

- C har ingen strängtyp!

`char*` — en pekare till en array av chars

`char[]` — en array av chars

I stora stycken är typerna ovan ekvivalenta

Strängar i C

- C har ingen strängtyp!

`char*` — en pekare till en array av chars

`char[]` — en array av chars

I stora stycken är typerna ovan ekvivalenta

- Kan du listor kan du strängar! (mer eller mindre)

`char s[] = "Hello"` är **precis** samma sak som

`char s[] = {'H', 'e', 'l', 'l', 'o', '\0'}` vilket är **precis** samma sak som

`char s[] = {72, 101, 108, 108, 111, 0}`

Strängar i C

- C har ingen strängtyp!

char* — en pekare till en array av chars

char[] — en array av chars

I stora stycken är typerna ovan ekvivalenta

- Kan du listor kan du strängar! (mer eller mindre)

char s[] = "Hello" är **precis** samma sak som

char s[] = {'H', 'e', 'l', 'l', 'o', '\0'} vilket är **precis** samma sak som

char s[] = {72, 101, 108, 108, 111, 0}

s =	'H'	'e'	'l'	'l'	'o'	'\0'
-----	-----	-----	-----	-----	-----	------

Strängar i C

- C har ingen strängtyp!

char* — en pekare till en array av chars

char[] — en array av chars

I stora stycken är typerna ovan ekvivalenta

- Kan du listor kan du strängar! (mer eller mindre)

char s[] = "Hello" är **precis** samma sak som

char s[] = {'H', 'e', 'l', 'l', 'o', '\0'} vilket är **precis** samma sak som

char s[] = {72, 101, 108, 108, 111, 0}

s =

'H'	'e'	'l'	'l'	'o'	'\0'
-----	-----	-----	-----	-----	------

*Hello är en array av **sex** chars!*

Vanliga misstag 1: Strängjämförelse

- Strängjämförelse med `==` avser *identitet*, inte *ekvivalens*

```
#include <stdio.h>

int main (void)
{
    char password[] = "abc123";
    char entered[128];

    puts("Please enter the secret code:");
    scanf("%s", entered); // read input

    if (entered == password)
    {
        puts("You are logged in!");
    }
    else
    {
        puts("Incorrect password!");
    }

    return 0;
}
```

password.c

```
#include <stdio.h>

int main (void)
{
    char password[] = "abc123";
    char *entered;      // kommer att "peka ut" inläst data
    size_t entered_size; // längden på entered

    puts("Please enter the secret code:");
    getline(&entered, &entered_size, stdin); // read input

    if (entered == password)
    {
        puts("You are logged in!");
    }
    else
    {
        puts("Incorrect password!");
    }

    return 0;
}
```

password-bad.c



```
#include <stdio.h>

int main (void)
{
    char password[] = "abc123";
    char *entered;      // kommer att "peka ut" inläst data
    size_t entered_size; // längden på entered

    puts("Please enter the secret code:");
    getline(&entered, &entered_size, stdin); // read input

    if (strncmp(entered, password, entered_size) == 0)
    {
        puts("You are logged in!");
    }
    else
    {
        puts("Incorrect password!");
    }

    return 0;
}
```

password-good.c



Vanliga misstag 2

- Aliasering!

*Inläsning med `getline()`
undviker detta!*

```
#include <stdio.h>

int main (void)
{
    char buffer [128];
    char *first;
    char *last;

    puts("What is your first name?");
    scanf("%s", buffer);
    first = buffer;

    puts("What is your last name?");
    scanf("%s", buffer);
    last = buffer;

    printf("Hello %s %s!\n", first, last);
    return 0;
}
```

Vanliga misstag 2: Aliasering

```
#include<stdio.h>

int main()
{
    char buffer[128];
    char first[128];
    char last[128];

    puts("What is your first name?");
    scanf("%s", buffer);
    first = buffer; // Fel! first och buffer är nu **samma* sträng

    puts("What is your last name?");
    scanf("%s", buffer); // När vi ändrar buffer här ändrar vi samtidigt first!
    last = buffer; // Fel! first, last och buffer är nu **samma* sträng

    printf("Hello %s %s!\n", first, last);
    return 0;
}
```

greeting.c

Vanliga misstag 3

- Strängar är arrayer som slutar med '\0'

```
#include <stdio.h>

void copy(char to[], char from[], int len)
{
    while(len--)
        to[len] = from[len];
}

int main (void)
{
    char *s = "Hello";
    char t[5];

    copy(t, s, 5);

    puts(t);

    return 0;
}
```


Vanliga misstag 3: ...som slutar med '\0'!

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

int main(int argc, char *argv[])
{
    char *s = "Hello";
    char t[5]; // Fel! För att få plats med "Hello" krävs 6 tecken
               // 5 för strängen **plus 1 för '\0'**
    strcpy(t, s);
    puts(t);

    return 0;
}
```

length.c

Standardbiblioteket `string.h`

- Inkluderas med `#include <string.h>`

Funktion	Beskrivning
<code>strlen(s)</code>	längden av <code>s</code> (utan <code>'\0'</code> -tecknet!)
<code>strncpy(s, t, n)</code>	kopiera <code>t</code> till <code>s</code> (upp till <code>n</code> tecken)
<code>strncmp(s, t, n)</code>	jämför <code>s</code> och <code>t</code> (upp till <code>n</code> tecken)
<code>== 0</code>	<code>s</code> och <code>t</code> är samma sträng
<code>> 0</code>	<code>s</code> kommer efter <code>t</code> i bokstavsordning
<code>< 0</code>	<code>s</code> kommer före <code>t</code> i bokstavsordning
<code>strncat(s, t, n)</code>	Lägg <code>t</code> följt av <code>s</code> i <code>t</code>

Funktioner i `stdlib.h` (togs även upp tidigare)

- Inkluderas med `#include <stdlib.h>`

`atoi(s)` — konvertera strängrepresentation av tal till motsvarande heltal

`atol(s)` — med `long` istf `int` som returtyp

Exempel:

```
atoi("123abc") == 123 (som int)
```

OBS!

- Alla funktioner är optimistiska

Förutsätter från att det finns tillräckligt minne i datat som används

Förutsätter att strängar är '`\0`'-terminerade korrekt

- Använd alltid `strncpy` (`strncpy`, ...) över `strcpy` (`strcpy`, ...) etc.

OBS!

- Alla funktioner är optimistiska

Förutsätter från att det finns tillräckligt minne i datat som används

Förutsätter att strängar är '`\0`'-terminerade korrekt

- Använd alltid `strncpy` (`strncpy`, ...) över `strcpy` (`strcpy`, ...) etc.
- Använd alltid funktioner från standardbiblioteken (som `string.h`) hellre än funktioner du skriver själv

OBS!

- Alla funktioner är optimistiska

Förutsätter från att det finns tillräckligt minne i datat som används

Förutsätter att strängar är '`\0`'-terminerade korrekt

- Använd alltid `strncpy` (`strncpy`, ...) över `strcpy` (`strcpy`, ...) etc.
- Använd alltid funktioner från standardbiblioteken (som `string.h`) hellre än funktioner du skriver själv

Även om det är en bra övning att ha implementerat motsvarande funktioner själv någon gång!

Standardbiblioteket `ctype.h`

- Inkluderas med `#include <ctype.h>`

Funktion	Beskrivning
<code>isalpha(c)</code>	Är <code>c</code> en bokstav?
<code>isdigit(c)</code>	Är <code>c</code> en siffra?
<code>islower(c)</code>	Är <code>c</code> en gemen? (liten bokstav)
<code>digittoint(c)</code>	Konvertera från '2' till 2 (som int)
<code>toupper(c)</code>	Från gemen till versal ('a' till 'A')

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;
}

```

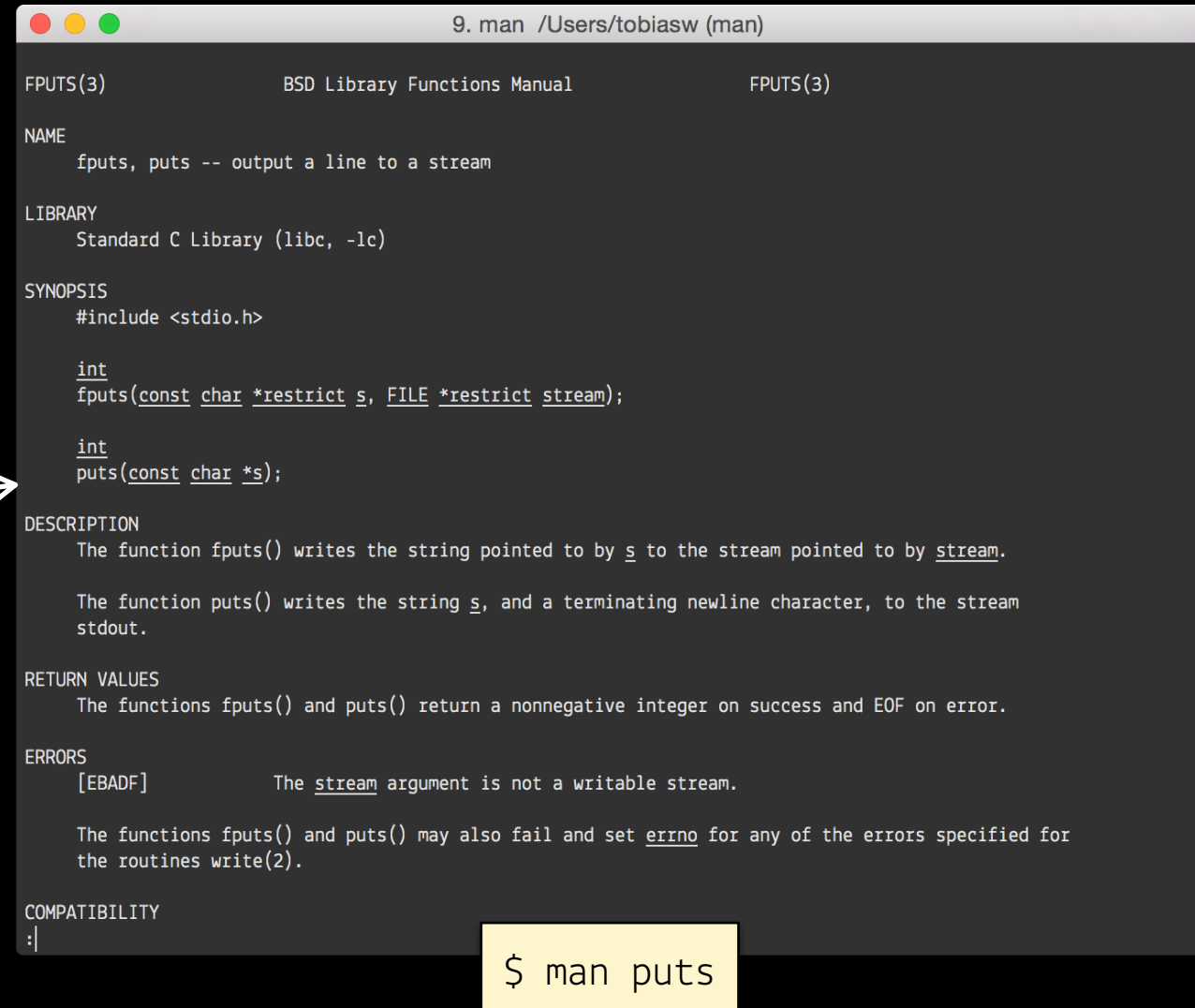
fib-rec-memo.c

man

time

valgrind

...



```
9. man /Users/tobiasw (man)

FPUTS(3)                BSD Library Functions Manual                FPUTS(3)

NAME
    fputs, puts -- output a line to a stream

LIBRARY
    Standard C Library (libc, -lc)

SYNOPSIS
    #include <stdio.h>

    int
    fputs(const char *restrict s, FILE *restrict stream);

    int
    puts(const char *s);

DESCRIPTION
    The function fputs() writes the string pointed to by s to the stream pointed to by stream.

    The function puts() writes the string s, and a terminating newline character, to the stream
    stdout.

RETURN VALUES
    The functions fputs() and puts() return a nonnegative integer on success and EOF on error.

ERRORS
    [EBADF]                The stream argument is not a writable stream.

    The functions fputs() and puts() may also fail and set errno for any of the errors specified for
    the routines write(2).

COMPATIBILITY
    :|

$ man puts
```

Vad skall jag göra härnäst?

1. Läs om kursen på kurswebben (lägg ett schema!)

Titta speciellt på DEADLINES

2. Skaffa ett GitHub-konto (se kurswebben för instruktioner)
3. Registrera dig på Piazza (du skall ha fått ett mail, annars gå via kurswebben)
4. Gör Kom-igång-uppgiften tillsammans med någon eller några (gärna i din grupp)
5. Ta kontakt med resten av din grupp (på Piazza), bestäm ett gruppmöte (se X63)

Föreläsning 2

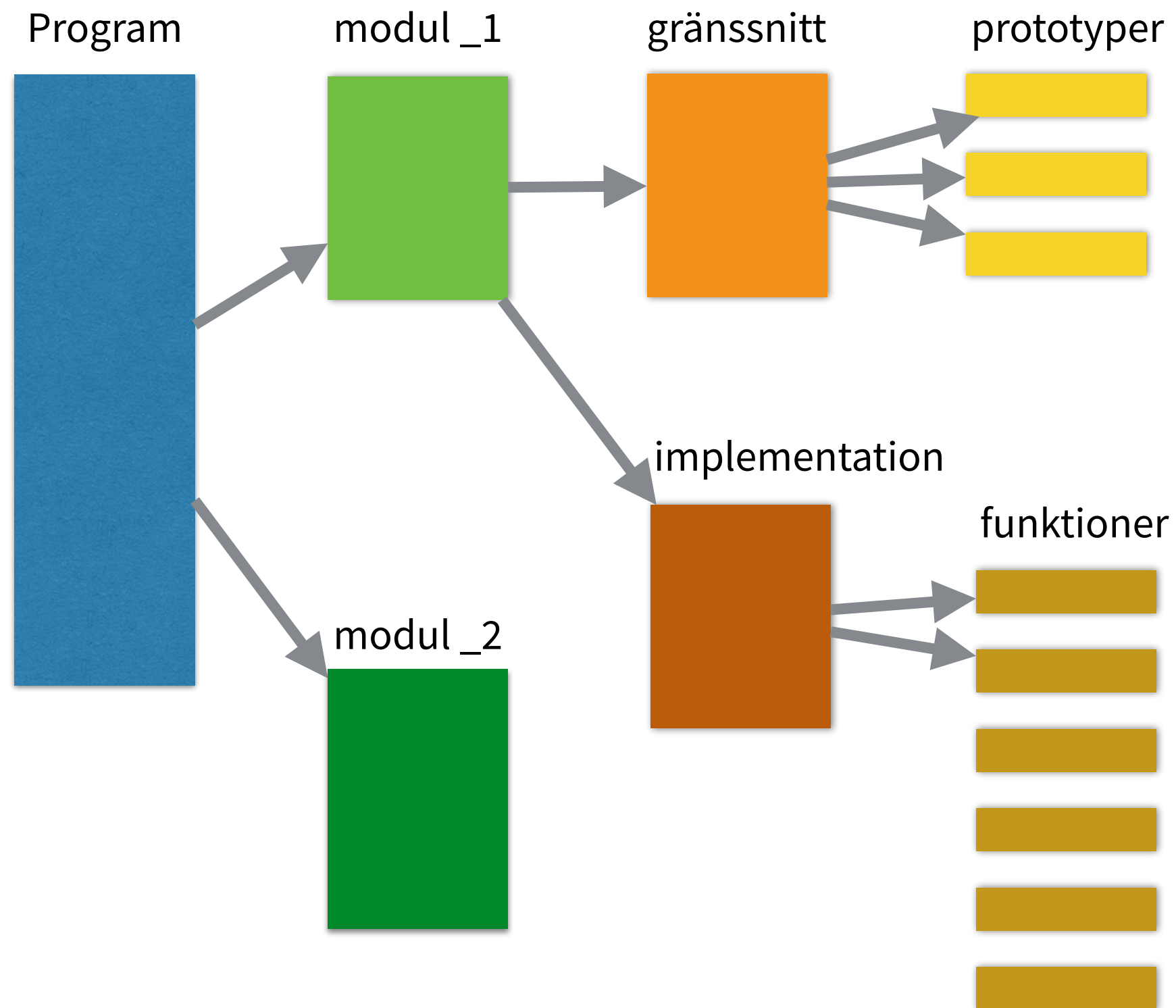
Lars-Henrik Eriksson
Kursansvarig

Föreläsningsbilderna är baserade på bilder gjorda av Tobias Wrigstad.

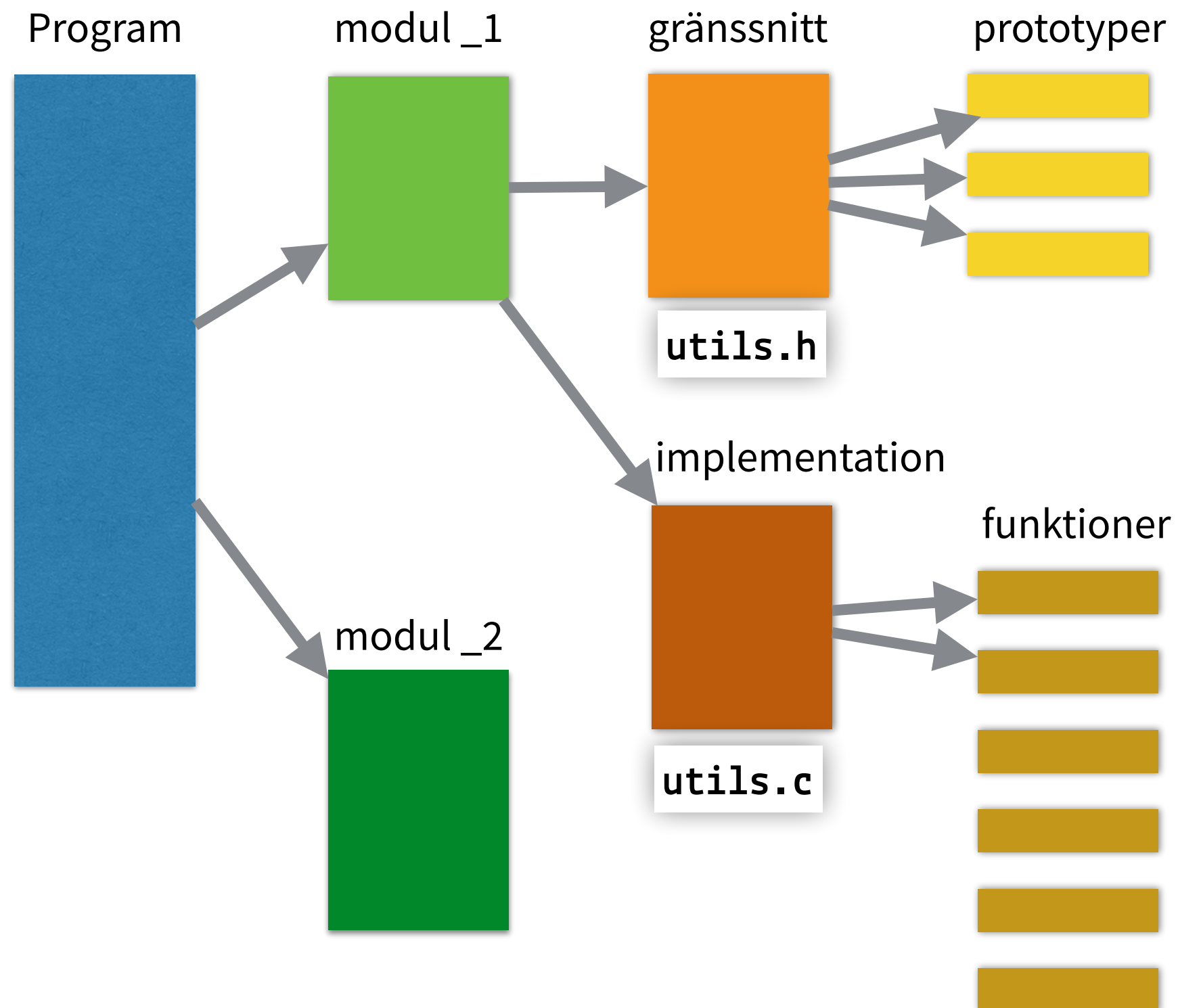
*Poster, unioner
och typedef*



Ett programs anatomi

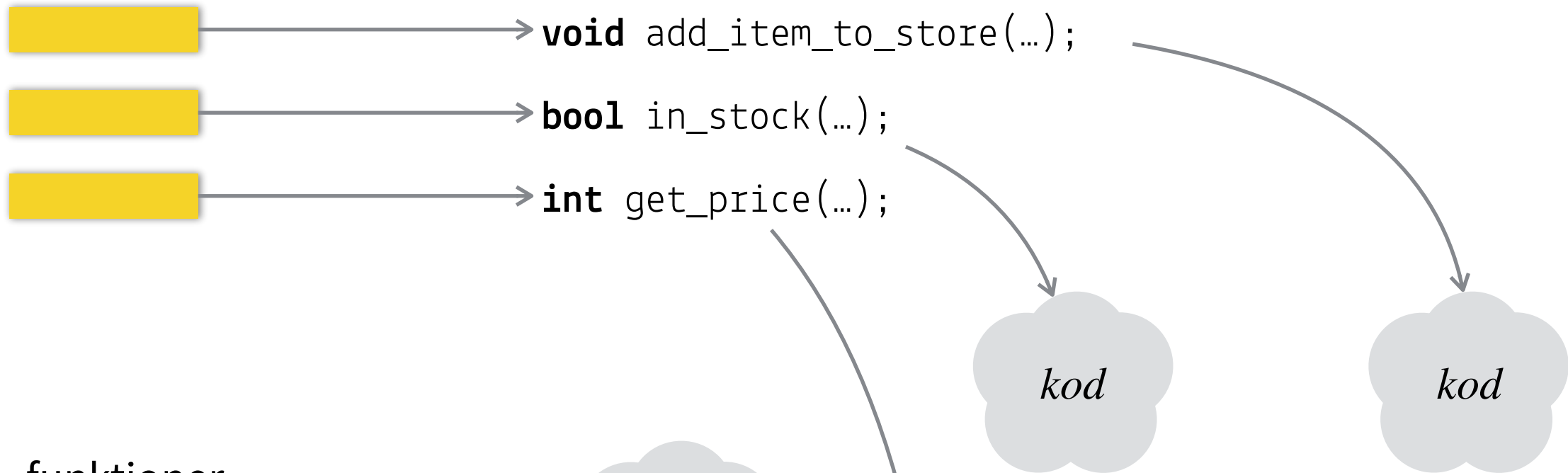


Ett programs anatomi

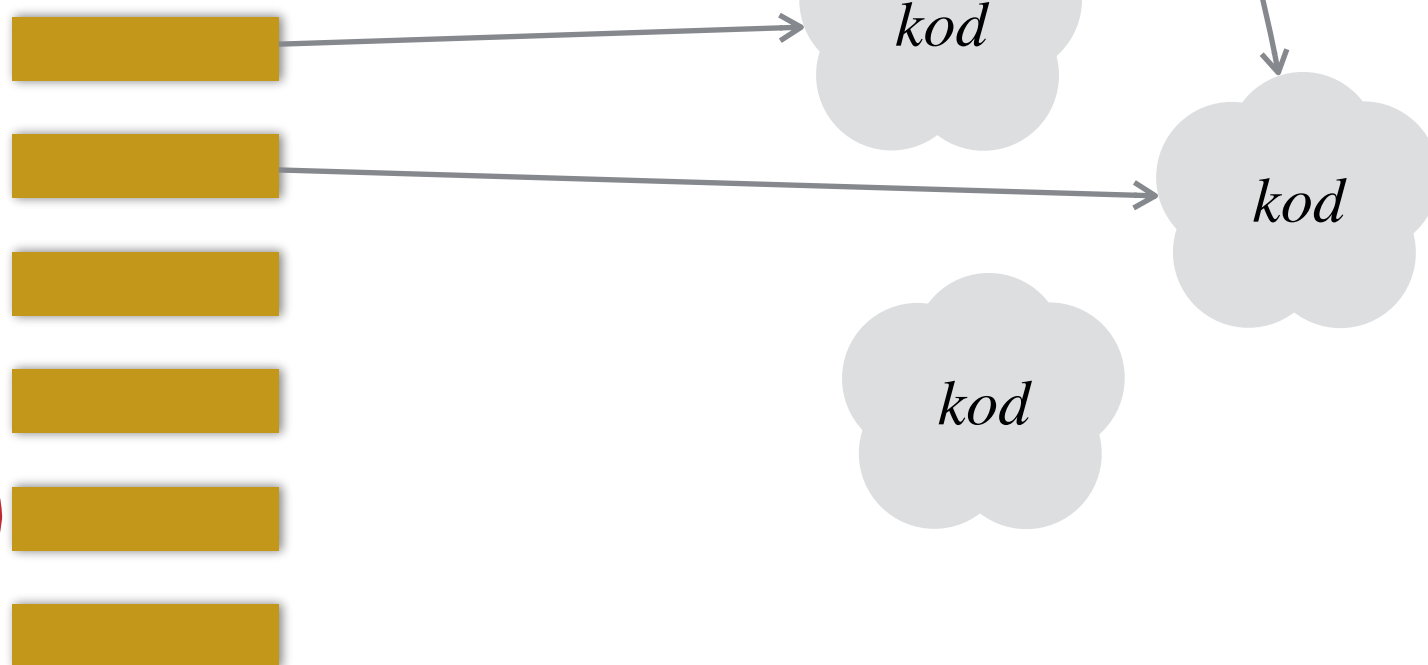


Funktionsabstraktionen

prototyper

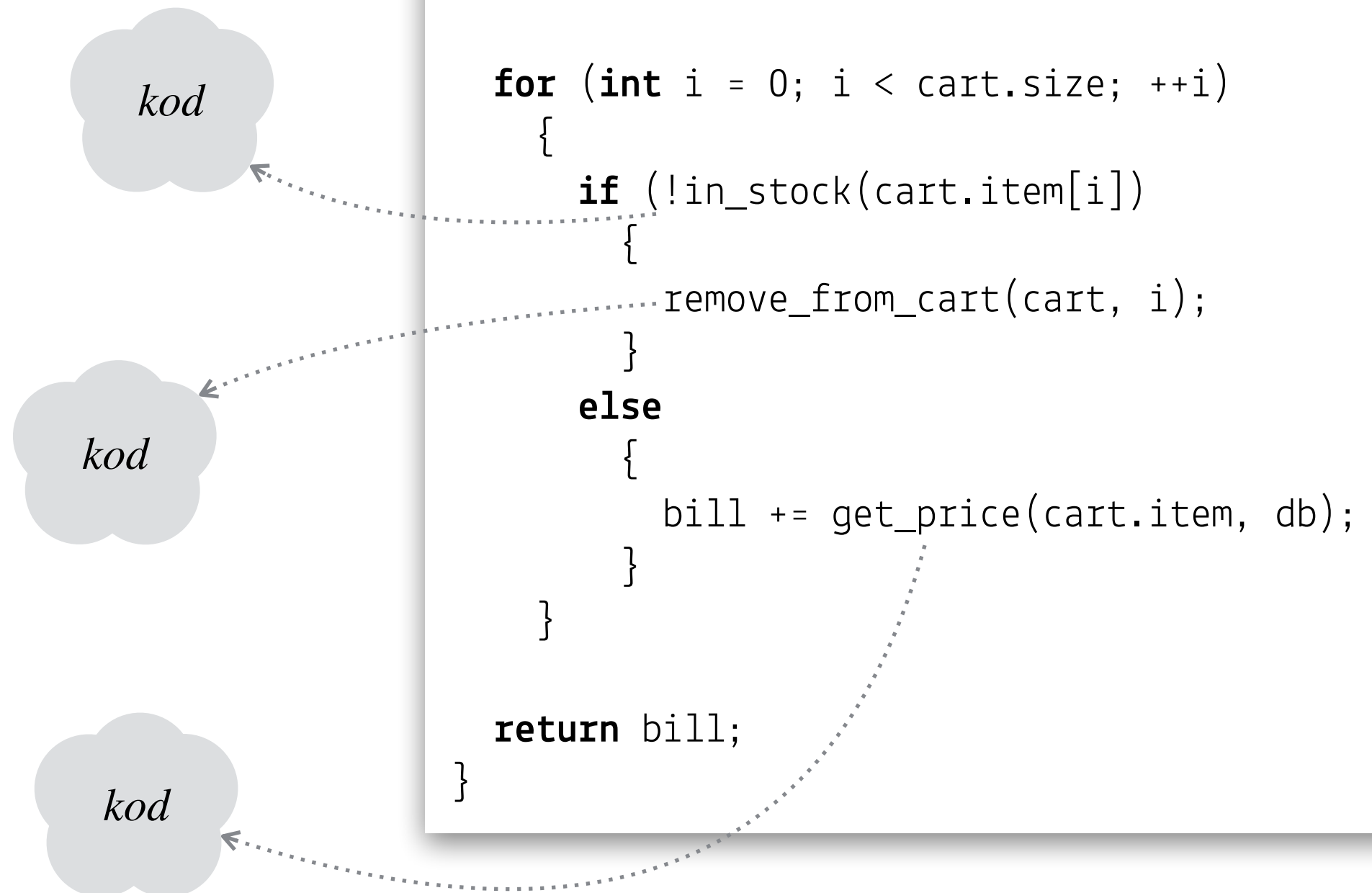


funktioner



Funktionsabstraktionen

Bygga abstraktioner av abstraktioner!



Sammansatta datatyper

- Minns Haskell...

Kraftfullt stöd för matchning, konstruktion och dekonstruktion av värden

Algebraiska datatyper

```
                -- name, year, month, day
data Anniversary = Birthday String Int Int Int
                -- spouse name 1, spouse name 2, year, month, day
                | Wedding String String Int Int Int

smithWedding = Wedding "John Smith" "Jane Smith" 1987 3 4
```

Sammansatta datatyper i C

värdesemantik!

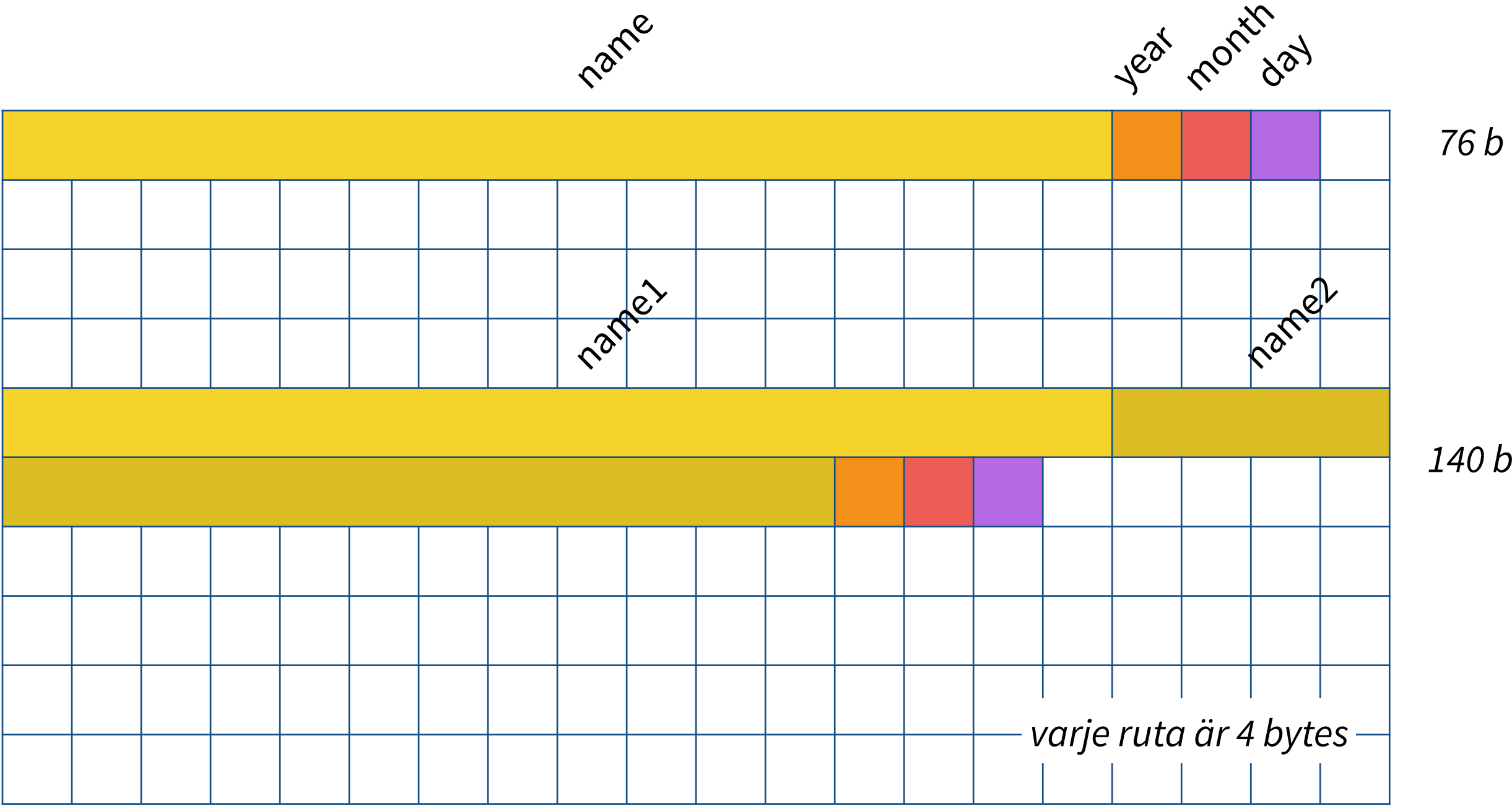
- Ingen pattern matching
- Inget enkelt sätt att konstruera eller dekonstruera
- Definitioner för att skapa ett sammanhängande datablock med olika *namngivna fält* som innehåller *värden av fix storlek*.

Ingen enkel motsvarighet till Anniversary

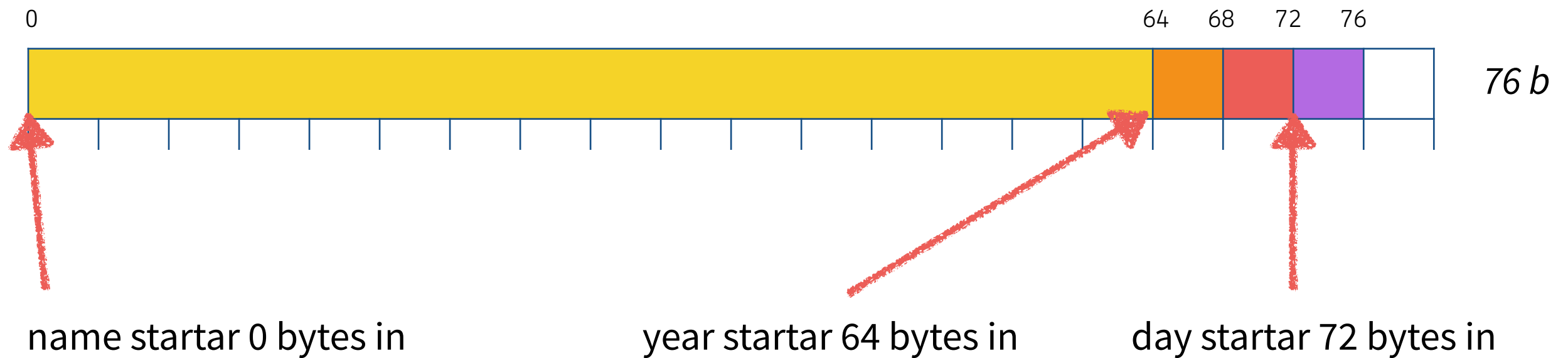
```
struct birthday
{
    char name[64];
    int year;
    int month;
    int day;
};
```

```
struct wedding
{
    char spouse_name1[64];
    char spouse_name2[64];
    int year;
    int month;
    int day;
};
```

En strukt är en "minneskarta" [char=1 byte, int=4 bytes nedan]



En strukt är en "minneskarta" [char=1 byte, int=4 bytes nedan]



```
&bday.year == &bday + sizeof(char[64])
```

```
&bday.month == &bday + sizeof(char[64]) + sizeof(int)
```

Unions — inte riktigt algebraiska datatyper...

- En union tillåter oss att gömma flera typer i en — som *alternativ*

Storleken (**sizeof**) på en union av T_1 och T_2 är $\max(\text{sizeof}(T_1), \text{sizeof}(T_2))$

Metadata för att veta vilken typ ett värde har måste stoppas in för hand

Jmf. kind nedan

```
enum anniversary_kind { WEDDING, BIRTHDAY };

struct anniversary
{
    enum anniversary_kind kind;
    union
    {
        struct birthday b;
        struct wedding w;
    };
};
```

Unions — inte riktigt algebraiska datatyper...

- En union tillåter oss att gömma flera typer i en — som *alternativ*

Storleken (**sizeof**) på en union av T_1 och T_2 är $\max(\text{sizeof}(T_1), \text{sizeof}(T_2))$

Metadata för att veta vilken typ ett värde har måste stoppas in för hand

Jmf. kind nedan

```
void use_anniversary(struct anniversary a)
{
    if (a.kind == BIRTHDAY)
    {
        a.b.name = ...
    }
    else
    {
        a.w.spouse_name1 = ...
    }
}
```


Att deklarera en variabel av strukt-typ

- Den nedre varianten är att föredra

Notera att den **nollställer allt** som inte tilldelas

```
// #1
struct wedding w; // uninitialised wedding
w.year = 1972;
// etc.

// #2
struct birthday b =
    (struct birthday) { .year = 1972, .month = 5, .day = 21 };
```

(union fungerar likadant)

Att läsa eller tilldela poster i en strukt

- Den nedre varianten är att föredra

Notera att den **nollställer allt** som inte tilldelas

```
struct wedding w;  
// uppdaterar day-posten  
w.day = 30;  
  
// läser (och skriver ur) spouse_name2-posten  
puts(w.spouse_name2);
```

```
struct anniversary a = ...;  
puts(a.w.spouse_name2);
```



"navigera" genom nästlade strukturer/unioner

```
#include <stdio.h>

struct point { int x, y; };

void print_point(struct point p)
{
    printf("(%d,%d)\n", p.x, p.y);
}

void move_point(struct point p, int dx, int dy)
{
    p.x += dx;
    p.y += dy;
}

int main(void) {
    struct point p = { 100, 50 }; // Valitt, men felbenäget!
    print_point(p);
    move_point(p, 10, 10);
    print_point(p);
    return 0;
}
```



pekarsemantik!

```
#include <stdio.h>

struct point { int x, y; };

void print_point(struct point p)
{
    printf("(%d,%d)\n", p.x, p.y);
}

void move_point(struct point *p, int dx, int dy)
{
    p->x += dx;
    p->y += dy;
}

int main(void) {
    struct point p = { 100, 50 }; // Valitt, men felbenäget!
    print_point(p);
    move_point(&p, 10, 10);
    print_point(p);
    return 0;
}
```



Pekarsemantik vs. värdesemantik

- Värdesemantik

Kopierar data — problem med identitet, men inga problem med förändringar

Kostar att kopiera data

- Pekarsemantik

Delar data — identitet bibehålls, men/och förändringar får ”globalt genomslag”

Kan vara dyrt att läsa data som är långt borta (eller negativ cache coherence)

- Vad är rätt i vilket program/sammanhang?

Typedef

*Vi kommer att använda
typedef från och med nu*

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

```
typedef struct point point_t;
```

```
struct point { int x, y; };
```

```
void print_point(point_t p)
{
    printf("(%d,%d)\n", p.x, p.y);
}
```

```
void move_point(point_t *p, int dx, int dy)
{
    p->x += dx;
    p->y += dy;
}
```

```
int main(void) {
    point_t p = (point_t) { .x = 100, .y = 50 };
    print_point(p);
    move_point(p, 10, 10);
    print_point(p);
    return 0;
}
```



Föreläsning 3

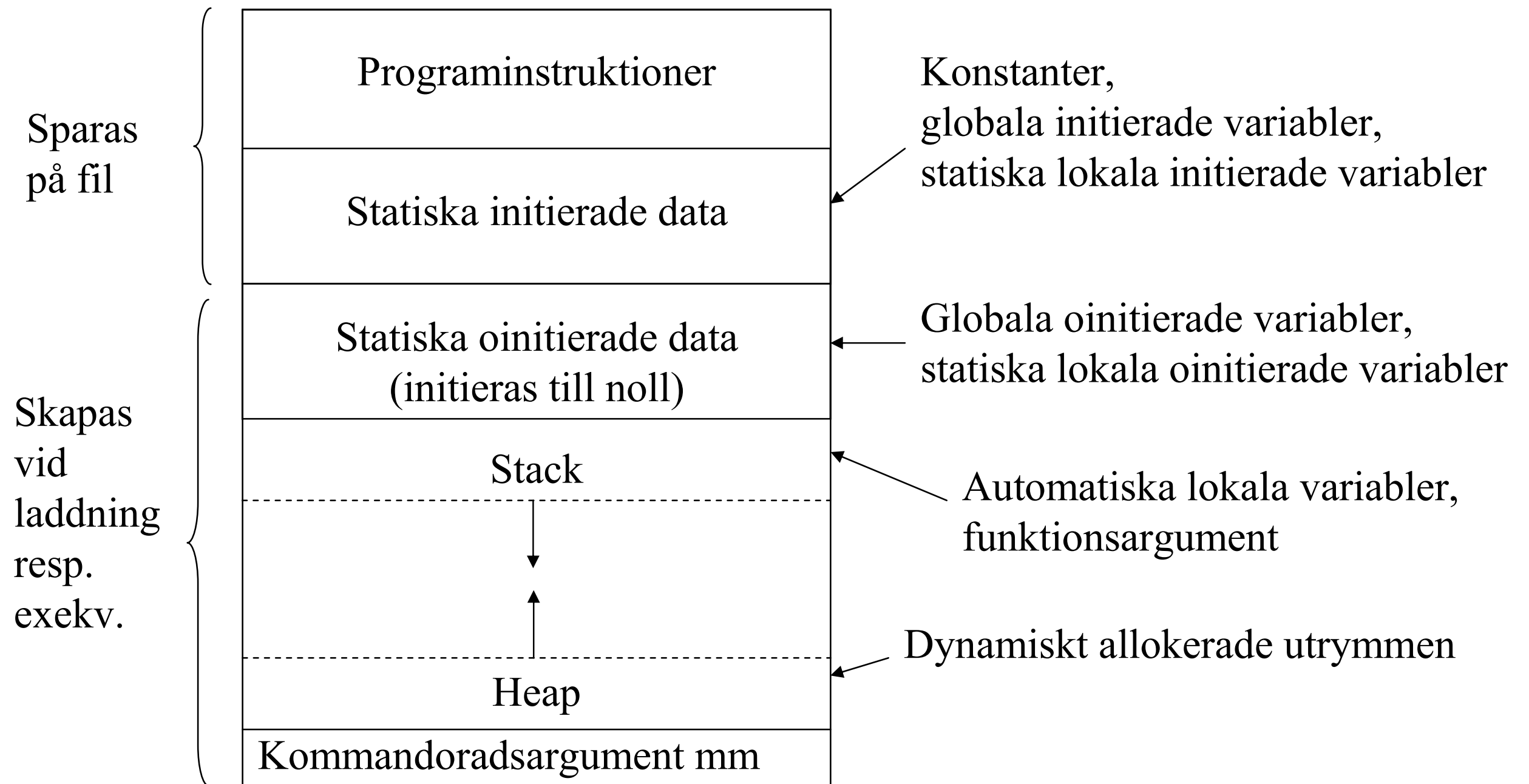
Lars-Henrik Eriksson
Kursansvarig

Föreläsningssbilderna är baserade på bilder gjorda av Tobias Wrigstad.

*Stack & heap, manuell
minneshantering, pekare*



Var lagras data?



Stacken

Läs Addendum om stack och heap

Ett enkelt stackexempel

SYFTET MED det här lilla häftet är att driva hem några poängar kring skillnaderna mellan *stack* och *heap*, adressstagningsoperatorn & och avrefereringsoperatorn *.

Vi börjar med ett enkelt kodexempel:

```
int x = 42;
int *y = &x;
int *z = NULL;
z = y;
(*z)++;
y++;
```

Vilka värden har variablerna x, y och z? Svaret är att värdet på x är 43 men att värdena på y och z är okända¹.

Men låt oss börja från början. För enkelhetens skull kan vi tänka oss att ett C-program har tillgång till två sorters minne: *stackminne* och *heapminne*. Stacken är det minnesutrymme som används för att lagra värdena i lokala variabler i funktioner, så kallade automatiska variabler. Betrakta följande funktion:

```
void stupid(int value) {
    if (value) {
        int smäller = value - 1;
        stupid(smäller);
    }
}
```

Funktionen har två lokala variabler, *value* och *smäller*, båda av typen *int*. Låt oss anta att en *int* är 4 bytes. Det betyder att vi kommer att behöva 8 bytes på stacken för värdena i *value* och *smäller*.²

Funktionen *stupid* är rekursiv; ett anrop *stupid(3)* kommer att leda till att funktionen anropar sig själv ytterligare tre gånger; varje funktion har en egen lokal variabel *value* vars värde är ett mindre än den anropande funktionens *value*-värde. Varje gång *stupid* anropas behövs ytterligare 8 bytes för att hålla värdena i dess *value* och *smäller*-variabler. Vi säger att varje funktionsanrop leder till att ny *stack frame* push:as på stacken som innehåller det minnesutrymmen

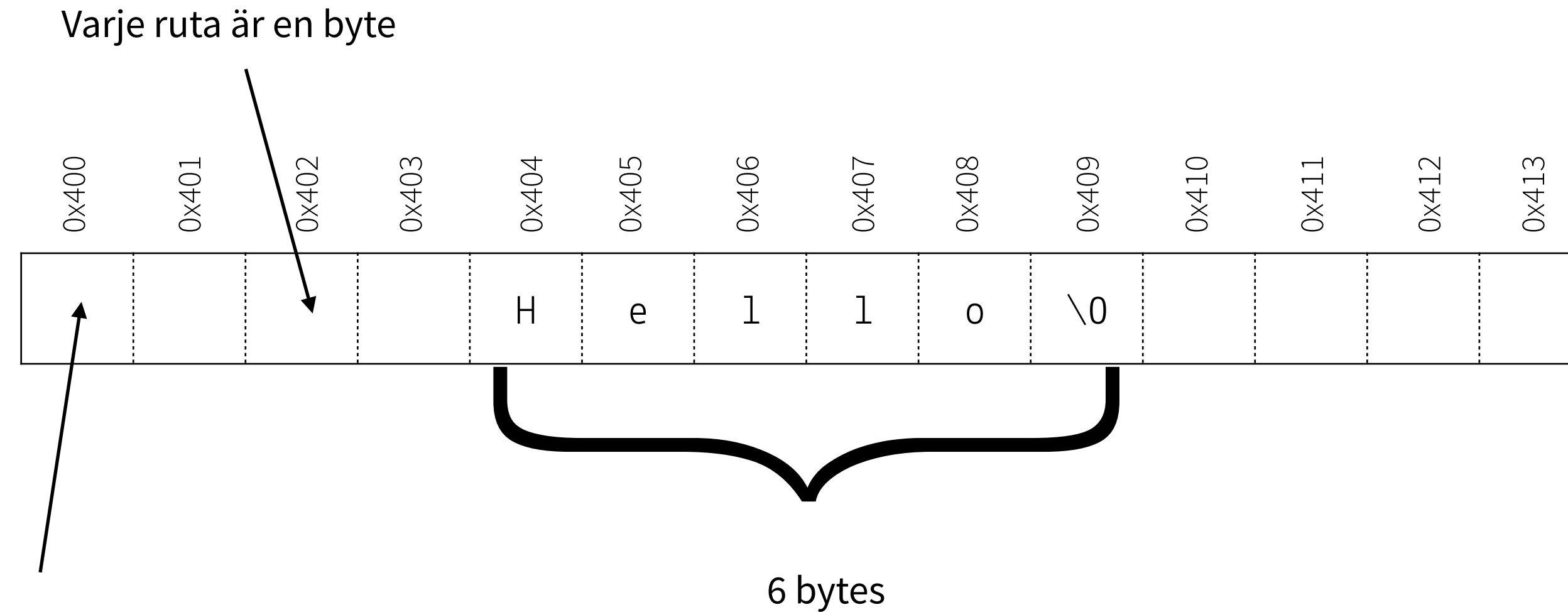
¹ Däremot så vet vi att om värdet på z är heltallet n, så är värdet på y n + 4, givet att storleken på en pekare är 4 bytes.

² Det är en liten förenkling - vissa andra data kan behövas, och smarta kompilatorer gör optimeringar som troligtvis bortser från sånt.

<https://github.com/I00PM-UU/ioopm17/blob/master/extramaterial/C-addendum-om-stack-och-heap.pdf>



C:s minnesmodell



Varje byte i minnet har
en **adress** — dess
avstånd från "starten"

Ett enkelt C-program & dess minnesanvändning

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

uint32_t fakultet(uint8_t f)
{
    if (f > 1)
    {
        return f * fakultet(f - 1);
    }
    else
    {
        return 1;
    }
}

int main(int argc, char *argv[])
{
    uint8_t n = (uint8_t) atol(argv[1]);
    uint32_t resultat = fakultet(n);

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

    return 0;
}
```



4 bytes

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

1 byte

```
uint32_t fakultet(uint8_t f)
{
    if (f > 1)
    {
        return f * fakultet(f - 1);
    }
    else
    {
        return 1;
    }
}
```

? bytes

?? bytes

1 byte

```
int main(int argc, char *argv[])
{
    uint8_t n = (uint8_t) atol(argv[1]);
    uint32_t resultat = fakultet(n);

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

    return 0;
}
```

4 bytes

+binären, etc.



```

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

uint32_t fakultet(uint8_t f)
{
    if (f > 1)
    {
        return f * fakultet(f - 1);
    }
    else
    {
        return 1;
    }
}

int main(int argc, char *argv[])
{
    uint8_t n = (uint8_t) atol(argv[1]);
    uint32_t resultat = fakultet(n);

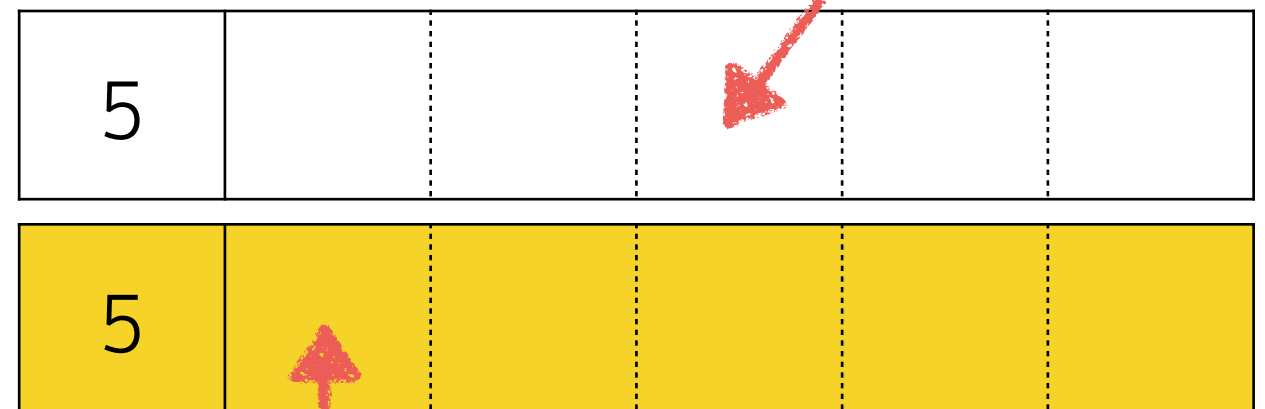
    printf("%d! = %d\n", n, resultat);

    return 0;
}

```

f <return value>

fakultet:s
stackframe



main:s stackframe

```

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

uint32_t fakultet(uint8_t f)
{
    if (f > 1)
    {
        return f * fakultet(f - 1);
    }
    else
    {
        return 1;
    }
}

int main(int argc, char *argv[])
{
    uint8_t n = (uint8_t) atol(argv[1]);
    uint32_t resultat = fakultet(n);

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

    return 0;
}

```

f <return value>

4					
5					
5					

```

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

uint32_t fakultet(uint8_t f)
{
    if (f > 1)
    {
        return f * fakultet(f - 1);
    }
    else
    {
        return 1;
    }
}

int main(int argc, char *argv[])
{
    uint8_t n = (uint8_t) atol(argv[1]);
    uint32_t resultat = fakultet(n);

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

    return 0;
}

```

f <return value>

3					
4					
5					
5					

```

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

uint32_t fakultet(uint8_t f)
{
    if (f > 1)
    {
        return f * fakultet(f - 1);
    }
    else
    {
        return 1;
    }
}

int main(int argc, char *argv[])
{
    uint8_t n = (uint8_t) atol(argv[1]);
    uint32_t resultat = fakultet(n);

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

    return 0;
}

```

f <return value>

2					
3					
4					
5					
5					


```

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

uint32_t fakultet(uint8_t f)
{
    if (f > 1)
    {
        return f * fakultet(f - 1);
    }
    else
    {
        return 1;
    }
}

int main(int argc, char *argv[])
{
    uint8_t n = (uint8_t) atol(argv[1]);
    uint32_t resultat = fakultet(n);

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

    return 0;
}

```

f <return value>

1					1
2					
3					
4					
5					
5					


```

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

uint32_t fakultet(uint8_t f)
{
    if (f > 1)
    {
        return f * fakultet(f - 1);
    }
    else
    {
        return 1;
    }
}

int main(int argc, char *argv[])
{
    uint8_t n = (uint8_t) atol(argv[1]);
    uint32_t resultat = fakultet(n);

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

    return 0;
}

```

f	<return value>				
1					1
2					2
3					
4					
5					
5					

```

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

uint32_t fakultet(uint8_t f)
{
    if (f > 1)
    {
        return f * fakultet(f - 1);
    }
    else
    {
        return 1;
    }
}

int main(int argc, char *argv[])
{
    uint8_t n = (uint8_t) atol(argv[1]);
    uint32_t resultat = fakultet(n);

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

    return 0;
}

```

f	<return value>				
1					1
2					2
3					6
4					
5					
5					

```

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

uint32_t fakultet(uint8_t f)
{
    if (f > 1)
    {
        return f * fakultet(f - 1);
    }
    else
    {
        return 1;
    }
}

int main(int argc, char *argv[])
{
    uint8_t n = (uint8_t) atol(argv[1]);
    uint32_t resultat = fakultet(n);

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

    return 0;
}

```

f	<return value>				
1					1
2					2
3					6
4					24
5					
5					

```

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

uint32_t fakultet(uint8_t f)
{
    if (f > 1)
    {
        return f * fakultet(f - 1);
    }
    else
    {
        return 1;
    }
}

int main(int argc, char *argv[])
{
    uint8_t n = (uint8_t) atol(argv[1]);
    uint32_t resultat = fakultet(n);

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

    return 0;
}

```

f	<return value>				
1					1
2					2
3					6
4					24
5					120
5					

```

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

uint32_t fakultet(uint8_t f)
{
    if (f > 1)
    {
        return f * fakultet(f - 1);
    }
    else
    {
        return 1;
    }
}

int main(int argc, char *argv[])
{
    uint8_t n = (uint8_t) atol(argv[1]);
    uint32_t resultat = fakultet(n);

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

    return 0;
}

```

f	<return value>				
1					1
2					2
3					6
4					24
5					120
5					

```

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

uint32_t fakultet(uint8_t f)
{
    if (f > 1)
    {
        return f * fakultet(f - 1);
    }
    else
    {
        return 1;
    }
}

int main(int argc, char *argv[])
{
    uint8_t n = (uint8_t) atol(argv[1]);
    uint32_t resultat = fakultet(n);

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

    return 0;
}

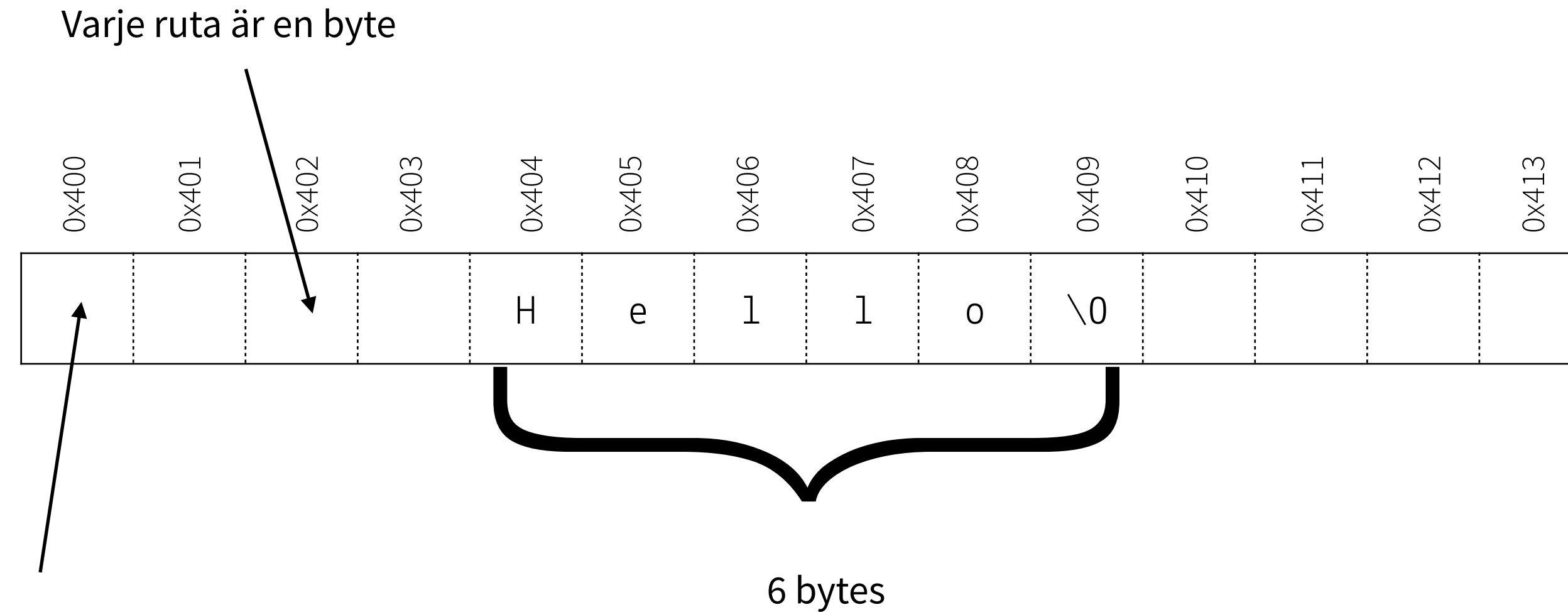
```

f	<return value>				
1					1
2					2
3					6
4					24
5					120
5					

”Pekare”



C:s minnesmodell



Varje byte i minnet har
en **adress** — dess
avstånd från "starten"

Pekare

- Adresser till platser i minnet
- Pekarvariabler
- Två operationer:

Peka om variabeln

Avreferera pekaren för att läsa/skriva minnesplatsen

```
int x = 42;    // x innehåller ett heltal
```



```
int x = 42;    // x innehåller ett heltal
```

```
int *p;        // p innehåller en adress till en  
                // plats i minnet där det finns ett  
                // heltal t.ex. 0x404
```



```
int x = 42;    // x innehåller ett heltal
```

```
int *p;        // p innehåller en adress till en  
                // plats i minnet där det finns ett  
                // heltal
```

```
p = &x;        // uppdatera pekaren p
```



```
int x = 42;    // x innehåller ett heltal
```

```
int *p;        // p innehåller en adress till en  
               // plats i minnet där det finns ett  
               // heltal
```

adresstagnings-
operatorn

→
p = &x;

```
// uppdatera pekaren p
```



```
int x = 42;    // x innehåller ett heltal
```

```
int *p;        // p innehåller en adress till en  
               // plats i minnet där det finns ett  
               // heltal
```

adresstagnings-
operatorn

→
`p = &x;`

```
// uppdatera pekaren p
```

```
*p = 17;      // uppdatera minnesplatsen som p  
              // pekar på  
              // Nu är x == 17
```



```
int x = 42;    // x innehåller ett heltal
```

```
int *p;
```

```
// p innehåller en adress till en  
// plats i minnet där det finns ett  
// heltal
```

adresstagnings-
operatorn

→
`p = &x;`

```
// uppdatera pekaren p
```

↖
`*p = 17;`

avrefererings-
operatorn

```
// uppdatera minnesplatsen som p  
// pekar på  
// Nu är x == 17
```



”Nullpekare”

Syntax: NULL

Semantik: ”pekar inte på någonting”

Representeras som talet noll.

```
int *x = NULL;
```



```
int *x = NULL;
```

```
int y = *x;
```



**NULL POINTER
DEREFERENCE**

```
int *x = NULL;  
int y = *x;
```

- "I call it my billion-dollar mistake... At that time [1965], I was designing the first comprehensive type system for references in an object-oriented language. My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years."

— Tony Hoare



Ny typ: void *

Tas upp mer senare

- En pekare till något av okänd typ

C tillåter **int** att den avrefereras

Användbar för att skapa t.ex. generella lagringsklasser (jmf. inlupp 2, lagerh.)

```
int x;  
int *y = &x;  
void *z = y;
```

```
*y = 42; // ok  
*z = 42; // kompilerar ej
```

```
int a = *y; // ok  
int b = *z; // kompilerar ej
```

```
int *c = (int *)z;  
*c = 42; // ok
```

Pekare: recap

`&var` — ta adressen till innehållet i `var`

`*var` — följ en pekare till en plats i minnet som håller data

`NULL` — en pekare som inte pekar på någonting

Återbesök av stackexemplet med pekare



```

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

void fakultet(uint8_t f, uint32_t *r)
{
    if (f > 1)
    {
        *r *= f; // *r = *r * f;
        fakultet(f - 1, r);
    }
}

int main(int argc, char *argv[])
{
    uint8_t n = (uint8_t) atol(argv[1]);
    uint32_t resultat = 1;
    fakultet(n, &resultat);

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

    return 0;
}

```

f

r



0x400

```

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

void fakultet(uint8_t f, uint32_t *r)
{
    if (f > 1)
    {
        *r *= f; // *r = *r * f;
        fakultet(f - 1, r);
    }
}

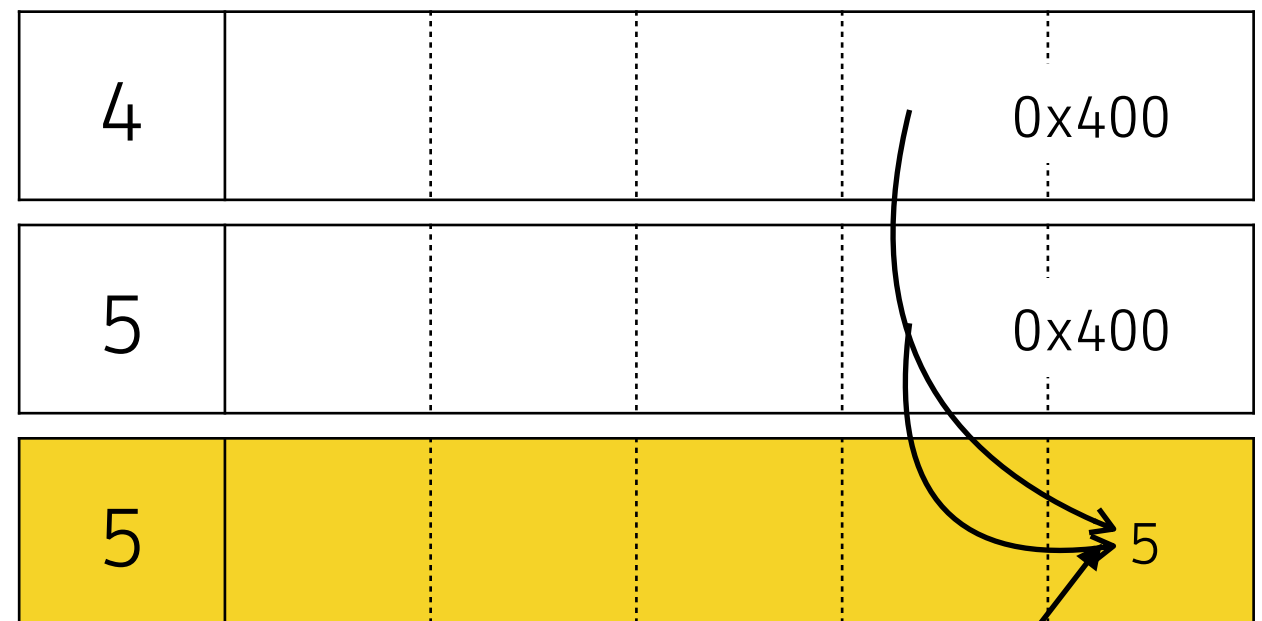
int main(int argc, char *argv[])
{
    uint8_t n = (uint8_t) atol(argv[1]);
    uint32_t resultat = 1;
    fakultet(n, &resultat);

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

    return 0;
}

```

f r



0x400


```

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

void fakultet(uint8_t f, uint32_t *r)
{
    if (f > 1)
    {
        *r *= f; // *r = *r * f;
        fakultet(f - 1, r);
    }
}

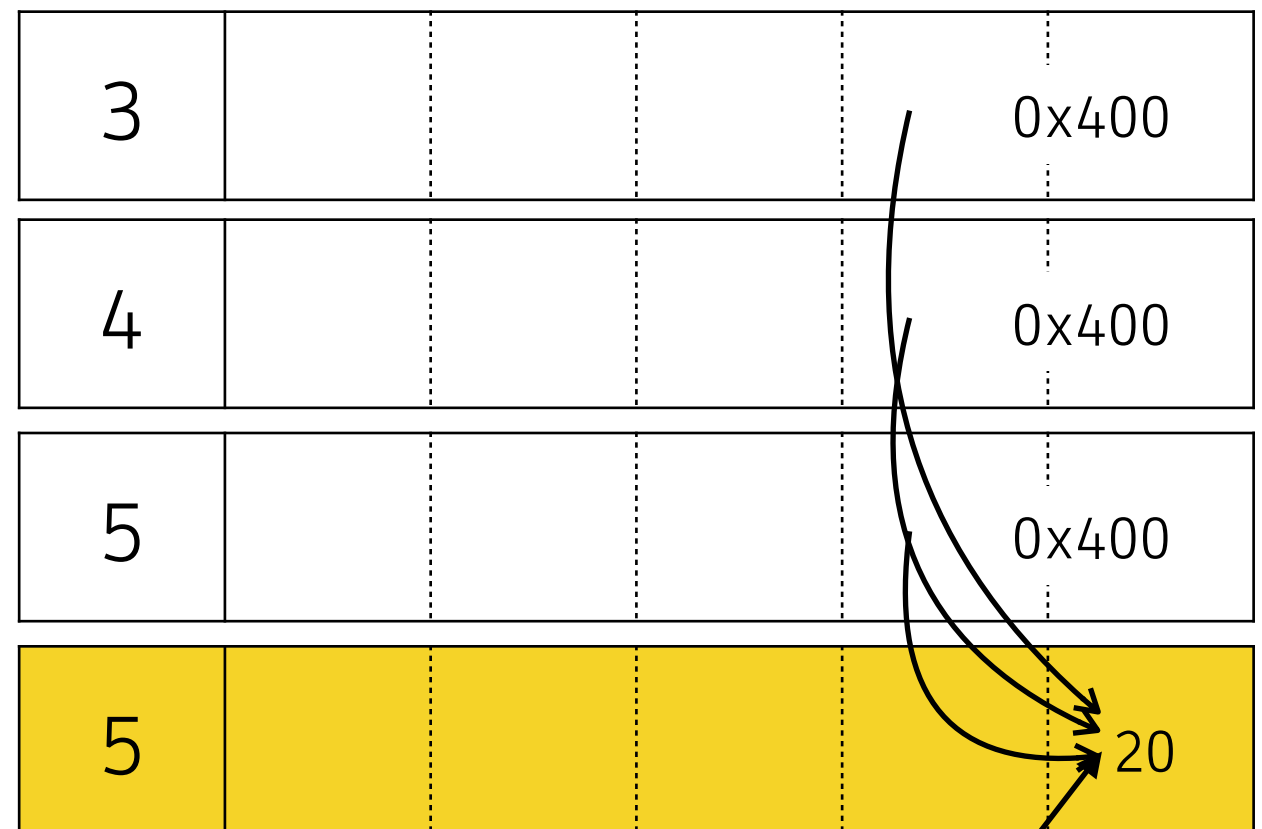
int main(int argc, char *argv[])
{
    uint8_t n = (uint8_t) atol(argv[1]);
    uint32_t resultat = 1;
    fakultet(n, &resultat);

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

    return 0;
}

```

f r



0x400


```

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

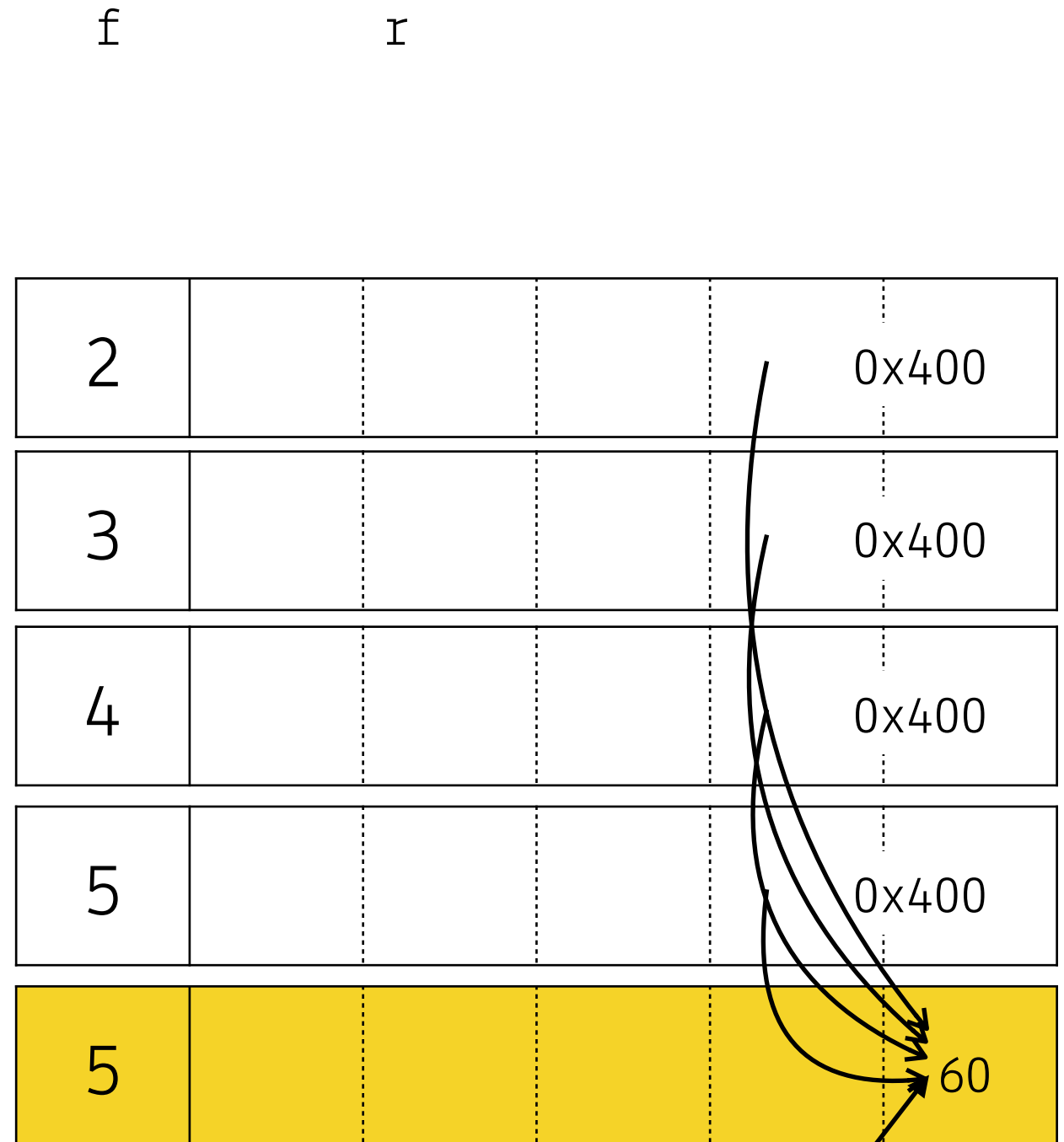
void fakultet(uint8_t f, uint32_t *r)
{
    if (f > 1)
    {
        *r *= f; // *r = *r * f;
        fakultet(f - 1, r);
    }
}

int main(int argc, char *argv[])
{
    uint8_t n = (uint8_t) atol(argv[1]);
    uint32_t resultat = 1;
    fakultet(n, &resultat);

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

    return 0;
}

```



0x400

```

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

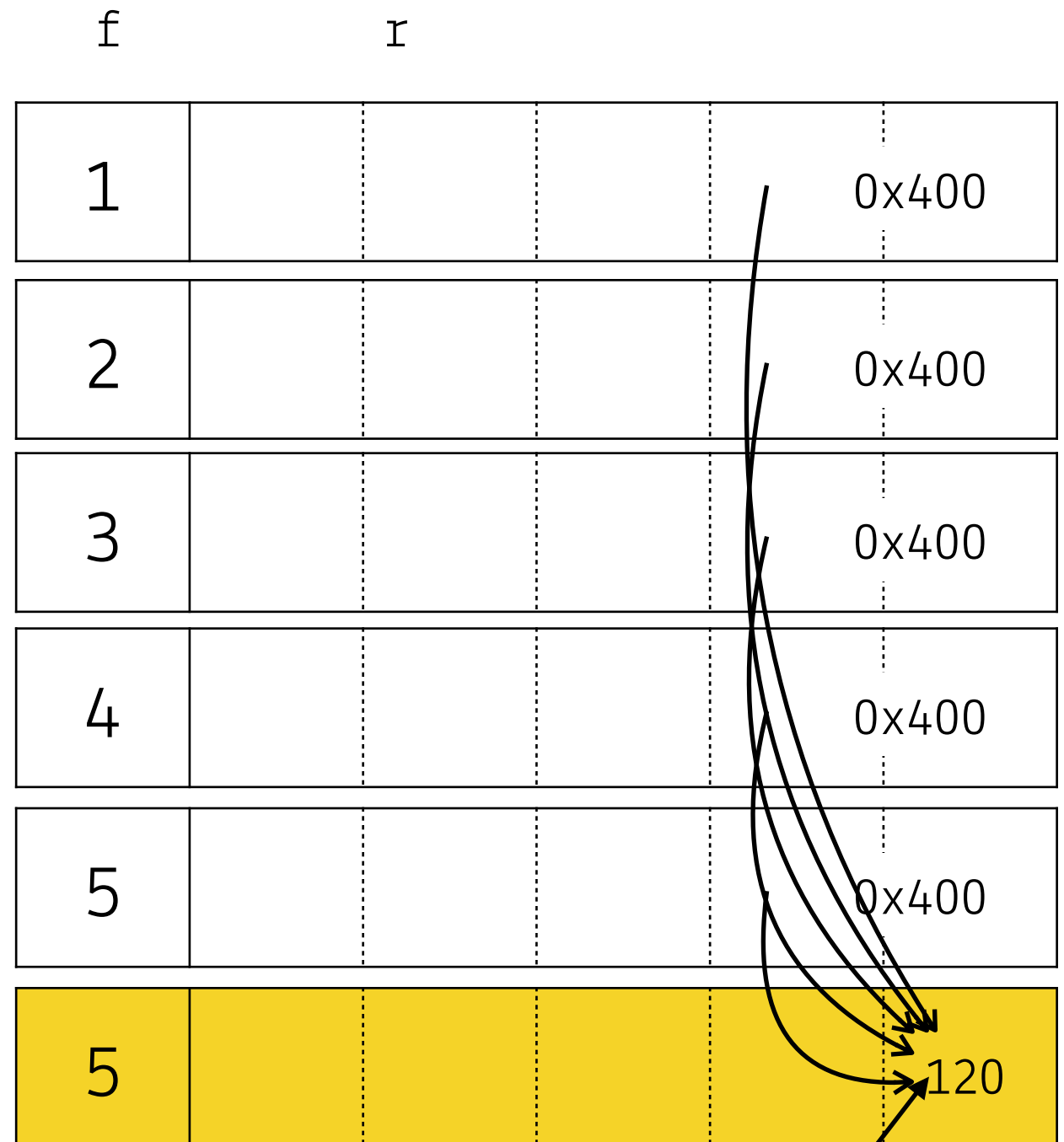
void fakultet(uint8_t f, uint32_t *r)
{
    if (f > 1)
    {
        *r *= f; // *r = *r * f;
        fakultet(f - 1, r);
    }
}

int main(int argc, char *argv[])
{
    uint8_t n = (uint8_t) atol(argv[1]);
    uint32_t resultat = 1;
    fakultet(n, &resultat);

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

    return 0;
}

```



0x400

```

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

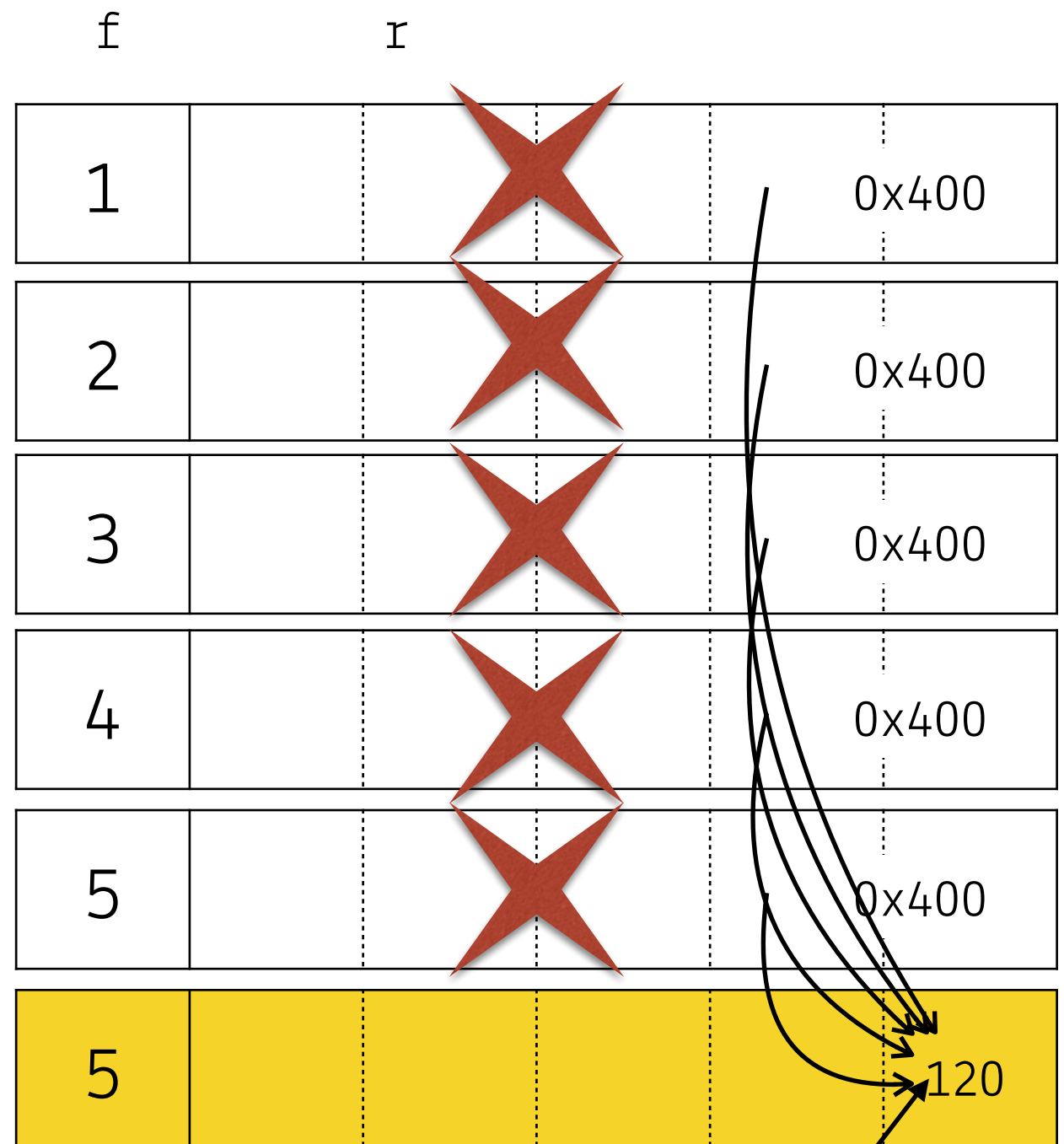
void fakultet(uint8_t f, uint32_t *r)
{
    if (f > 1)
    {
        *r *= f; // *r = *r * f;
        fakultet(f - 1, r);
    }
}

int main(int argc, char *argv[])
{
    uint8_t n = (uint8_t) atol(argv[1]);
    uint32_t resultat = 1;
    fakultet(n, &resultat);

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

    return 0;
}

```



0x400

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

void fakultet(uint8_t f, uint32_t *r)
{
    for (int i = 0; i < f; ++i)
    {
        *r *= f-i;
    }
}

int main(int argc, char *argv[])
{
    uint8_t n = (uint8_t) atol(argv[1]);
    uint32_t resultat = 1;
    fakultet(n, &resultat);

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

    return 0;
}
```

Kör i konstant minne

(varför?)



Stacken: recap

- Enda minnet som C:s ”exekveringsmiljö” hanterar automatiskt

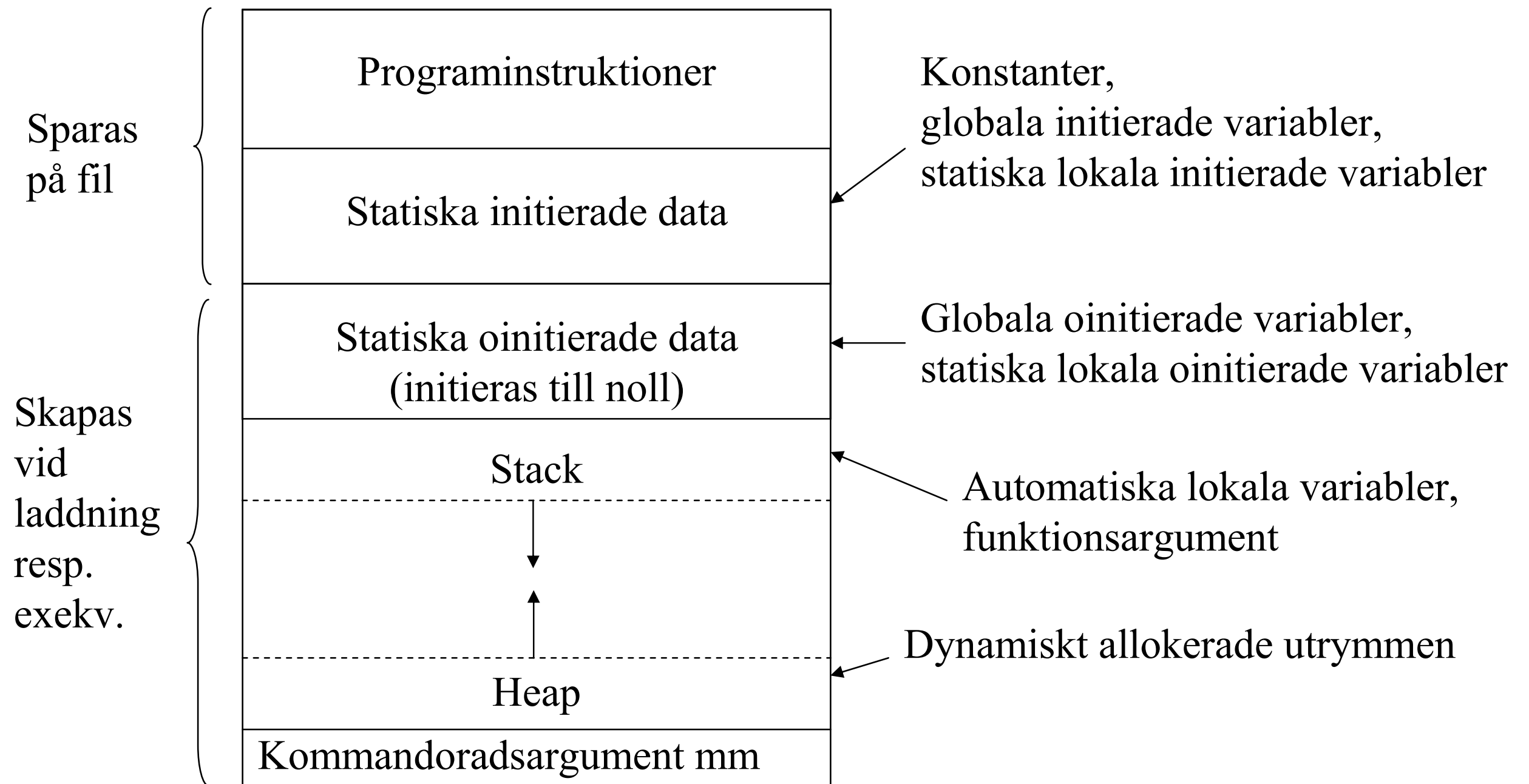
Håller ”kortlivat data” (knutet till funktionens livslängd)

Varje funktionsanrop ger upphov till en ny ”stack frame”
(not. parameteröverföring sker i regel mha register)

Allokeras och avallokeras automatiskt

- Extremt effektiv allokeringssmetod (”bump pointer”)
- Stacken är en del av minnet — alla variabler på stacken har en adress och kan pekars ut av pekare

Var lagras data?



Heapen

Läs Addendum om stack och heap



Ett enkelt stackexempel

SYFTET MED det här lilla häftet är att driva hem några poänger kring skillnaderna mellan *stack* och *heap*, adressstagningsoperatörn & och avrefereringsoperatörn `*`.

Vi börjar med ett enkelt kodexempel:

```
int x = 42;
int *y = &x;
int *z = NULL;
z = y;
(*z)++;
y++;
```

Vilka värden har variablerna `x`, `y` och `z`? Svaret är att värdet på `x` är 43 men att värdena på `y` och `z` är okända¹.

Men låt oss börja från början. För enkelhetens skull kan vi tänka oss att ett C-program har tillgång till två sorters minne: *stackminne* och *heapminne*. Stacken är det minnesutrymme som används för att lagra värdena i lokala variabler i funktioner, så kallade automatiska variabler. Betrakta följande funktion:

```
void stupid(int value) {
    if (value) {
        int smäller = value - 1;
        stupid(smäller);
    }
}
```

Funktionen har två lokala variabler, `value` och `smäller`, båda av typen `int`. Låt oss anta att en `int` är 4 bytes. Det betyder att vi kommer att behöva 8 bytes på stacken för värdena i `value` och `smäller`.²

Funktionen `stupid` är rekursiv; ett anrop `stupid(3)` kommer att leda till att funktionen anropar sig själv ytterligare tre gånger; varje funktion har en egen lokal variabel `value` vars värde är ett mindre än den anropande funktionens `value`-värde. Varje gång `stupid` anropas behövs ytterligare 8 bytes för att hålla värdena i dess `value` och `smäller`-variabler. Vi säger att varje funktionsanrop leder till att ny *stack frame* push:as på stacken som innehåller det minnesutrymmen

¹ Däremot så vet vi att om värdet på `z` är heltallet `n`, så är värdet på `y` `n + 4`, givet att storleken på en pekare är 4 bytes.

² Det är en liten förenkling - vissa andra data kan behövas, och smarta kompilatorer gör optimeringar som trolrar bort variabler som inte behövs, som tex. `smäller` här. Vi bortser från sånt.

<https://github.com/I00PM-UU/ioopm15/blob/master/extramaterial/C-addendum-om-stack-och-heap.pdf>



Heapen

- Till för lagring av ”långlivat” data
- Hanteras manuellt

För varje data som skall lagras på heapen måste man explicit be om ett motsvarande utrymme mha `malloc` (även `realloc` och `calloc`)

När man är färdig med data måste man explicit returnera det, annars läcker programmet minne mha funktionen `free`

- Heapen bara tillgänglig via pekare

Exempel: funktionen strdup()

```
char *strdup(const char *str)
{
    char *dup = malloc(strlen(str)+1);
    char *cursor = dup;

    while (*str)
    {
        *cursor = *str;
        ++str;
        ++cursor;
    }
    *cursor = '\0';

    return dup;
}
```

resultatet måste frigöras — någonstans! (Var och när?)

Dynamiska strukturer

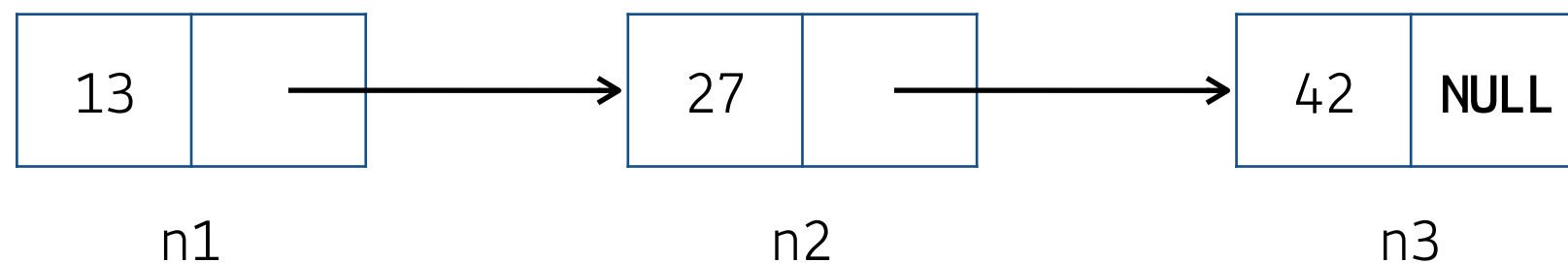
- Heapen används för att lagra stora data och data vars storlek växer dynamiskt, t.ex.

Binära träd, listor, köer, databaser, filbuffrar, etc.

- Vi använder malloc för att ”reservera en del av heapen” — denna kan vara ”var som helst”

Pekare **nödvändiga** för att *länka samman* datastrukturer som byggs upp av **fler än ett** anrop till malloc!

Pekare och länkade strukturer [På stacken]



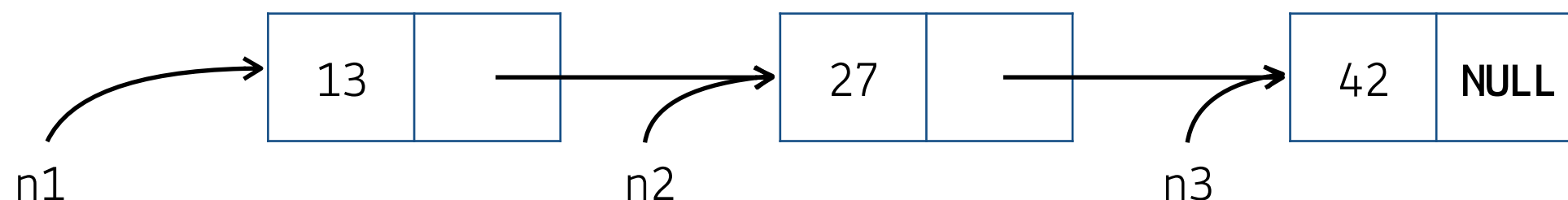
```
struct node n1;  
struct node n2;  
struct node n3;
```

```
n1 = (struct node) { .number = 13, .next = &n2 };  
n2 = (struct node) { .number = 27, .next = &n3 };  
n3 = (struct node) { .number = 42, .next = NULL };
```

```
struct node  
{  
    int number;  
    struct node *next;  
};
```



Pekare och länkade strukturer [På heapen]



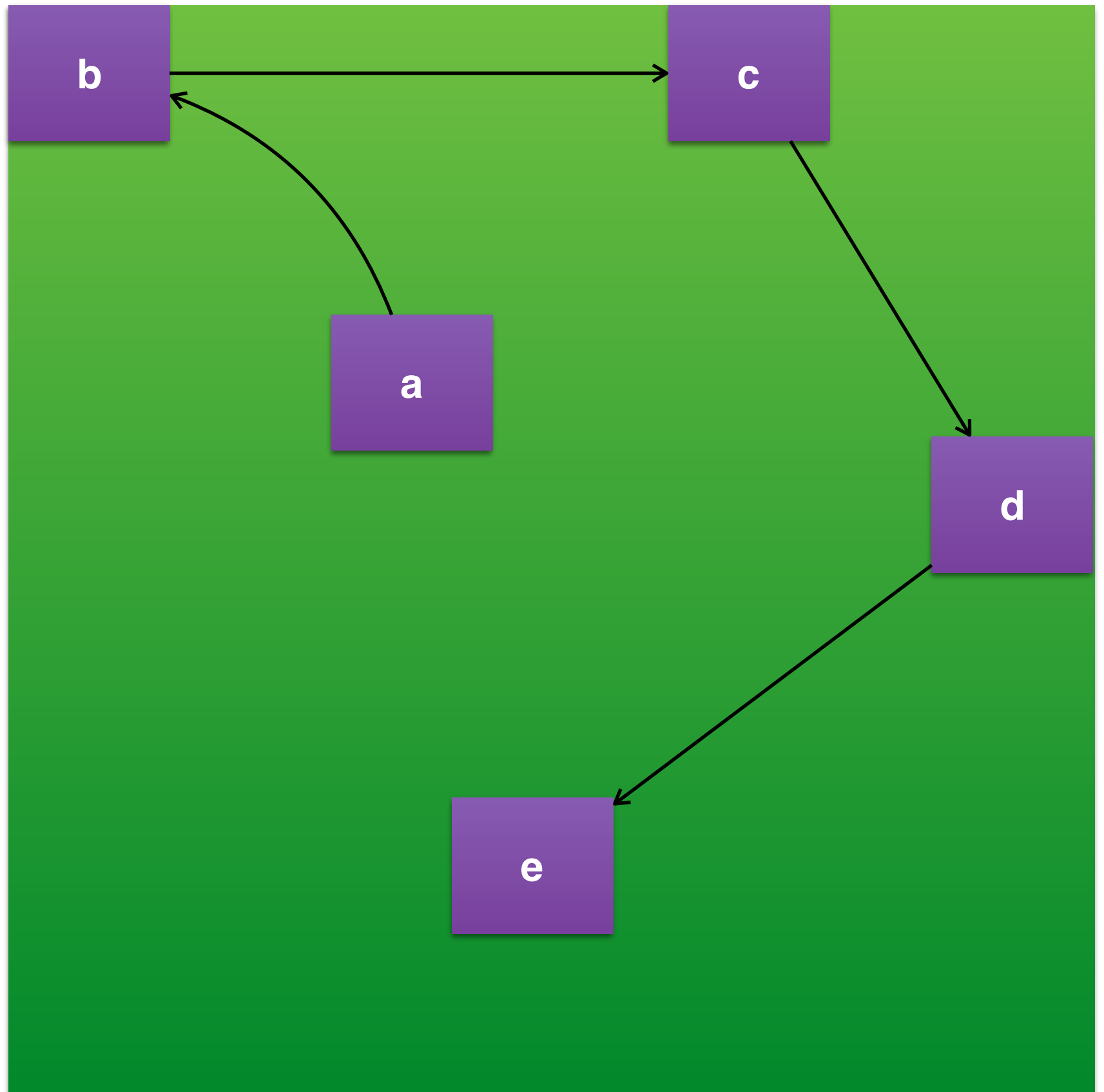
```
struct node *n1 = malloc(sizeof(struct node));
struct node *n2 = malloc(sizeof(struct node));
struct node *n3 = malloc(sizeof(struct node));

*n1 = (struct node) { .number = 13, .next = n2 };
*n2 = (struct node) { .number = 27, .next = n3 };
*n3 = (struct node) { .number = 42, .next = NULL };
```



En "länkad" lista

```
struct node
{
    int number;
    struct node *next;
};
```



Avreferera pekare

`(*node).next = node->next`



Följ pekaren



Läs next-posten
i strukten



Båda i samma
operator



Att skapa noder i ett träd

```
struct tree_node
{
    node_t *left;
    node_t *right;
    int key;
    data_t *data;
};
```

```
node_t node_new(int key, data_t *data)
{
    node_t *n = malloc(sizeof(node_t));
    if (n)
    {
        n->left = NULL;
        n->right = NULL;
        n->key = key;
        n->data = data;
    }
    return n;
}
```



Ta bort noder ur ett träd

```
void node_deallocate1(node_t *n)
{
    if (n->left)  free(n->left);
    if (n->right) free(n->right);
    free(n);
}
```

```
void node_deallocate2(node_t *n)
{
    if (n->left)  free(n->left);
    if (n->right) free(n->right);
    free(n->data);
    free(n);
}
```

```
data_t *node_deallocate3(node_t *n)
{
    if (n->left)  free(n->left);
    if (n->right) free(n->right);
    free(n);
    return n->data; // BOOOOM!!!!
}
```

```
data_t *node_deallocate4(node_t *n)
{
    if (n->left)  free(n->left);
    if (n->right) free(n->right);
    data_t *tmp = n->data;
    free(n);
    return tmp;
}
```



Minneshantering

- Allokera minne explicit med ngn rutin (t.ex. `malloc`)
- Frigör minne explicit med ngn rutin (t.ex. `free`)
- Vem för bok över vilket minne som är ledigt resp. använt?
- Hur hittar vi ett lämpligt ledigt utrymme?
- Vad betyder lämpligt?

```
a = malloc(A);
```

```
b = malloc(B);
```

```
c = malloc(C);
```

```
d = malloc(D);
```

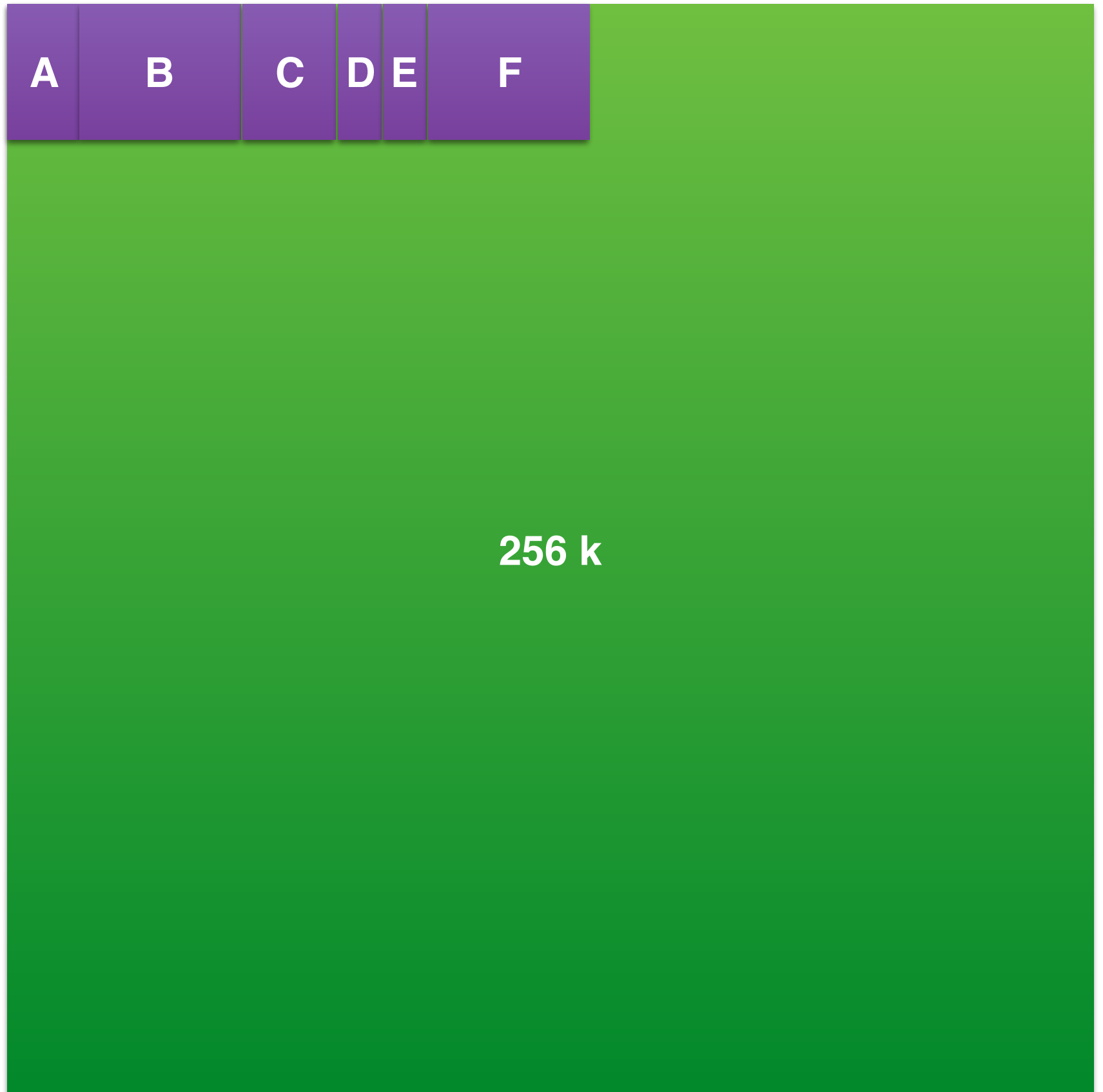
```
e = malloc(E);
```

```
f = malloc(F);
```

```
free(a);
```

```
free(c);
```

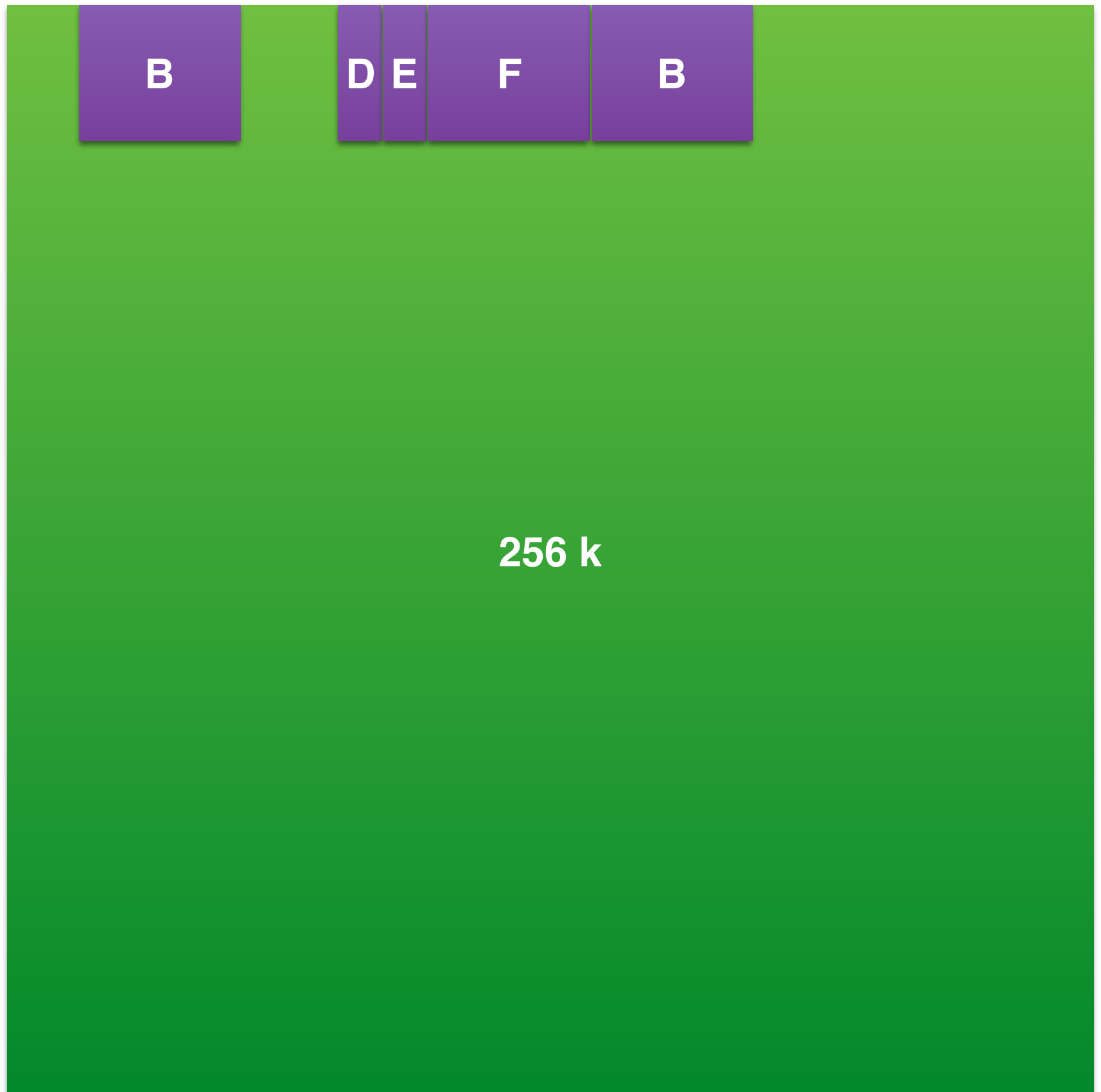
```
g = malloc(B);
```



B



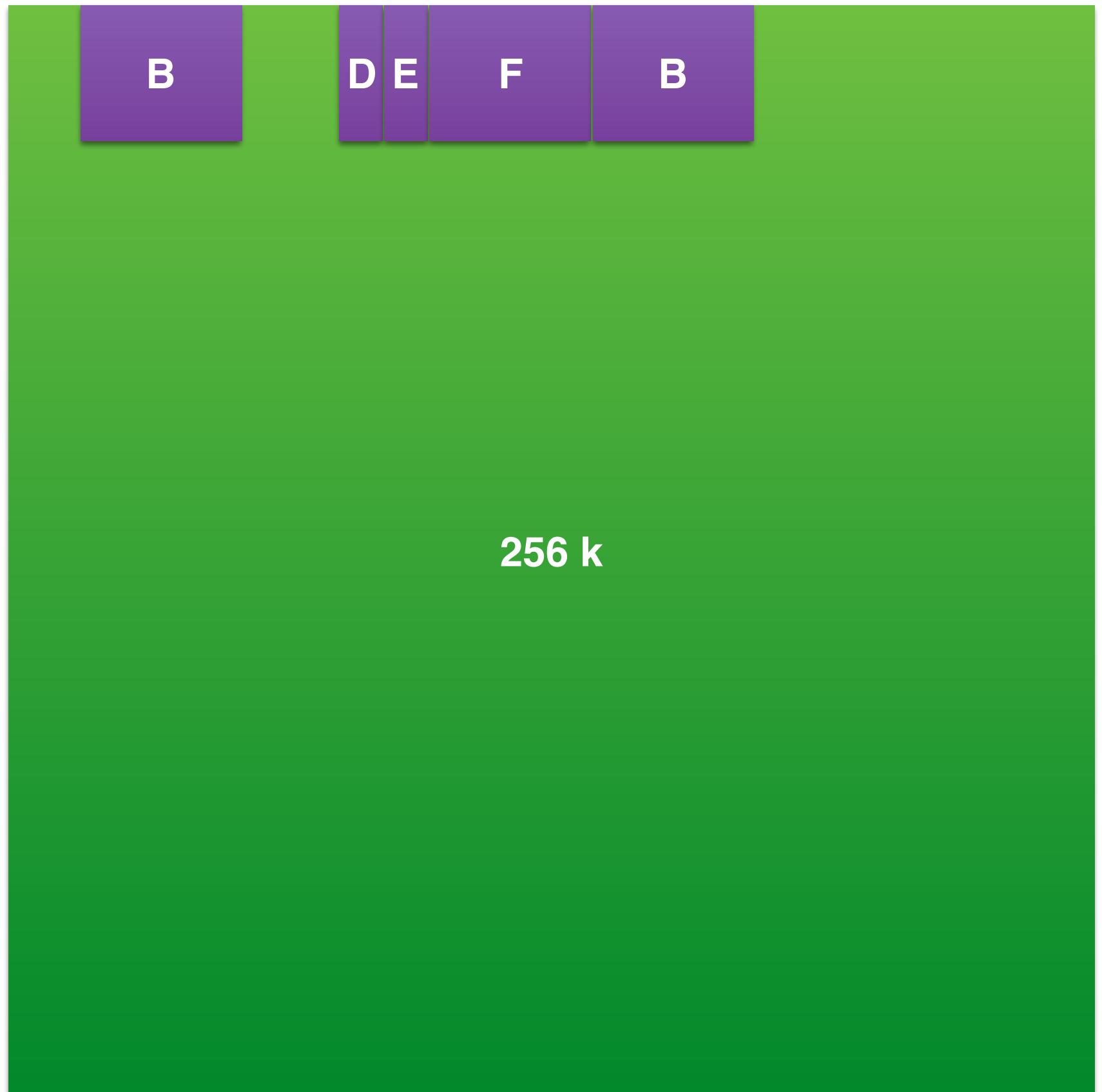
```
a = malloc(A);  
b = malloc(B);  
c = malloc(C);  
d = malloc(D);  
e = malloc(E);  
f = malloc(F);  
free(a);  
free(c);  
g = malloc(B);
```



Fragmentering

*Vi fick inte rum med B' i
det lediga minnet från
A och C eftersom de
inte var samman-
hängande (konsekutiva;
contiguous)*

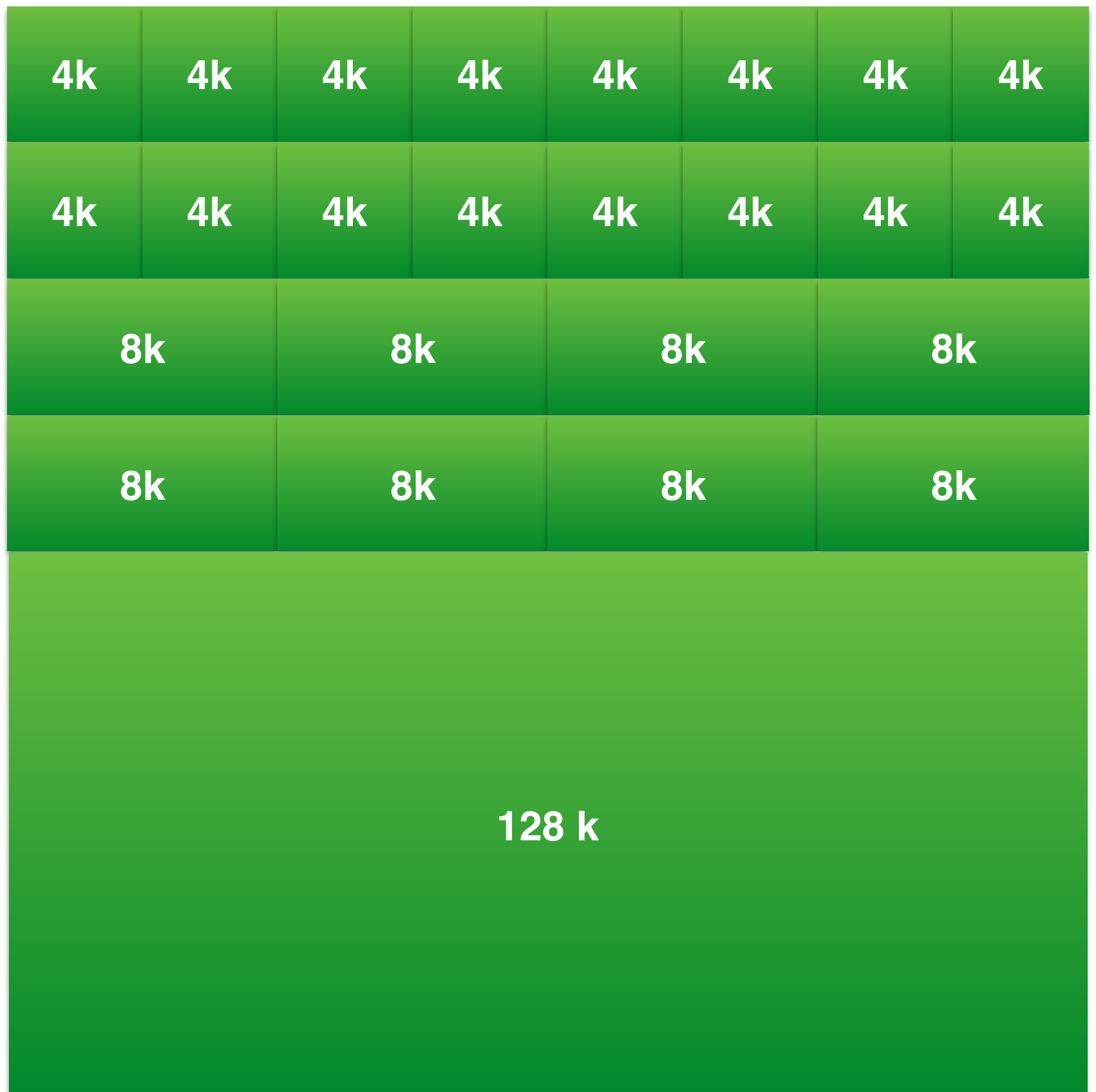
*Kan leda till att minnet
effektivt tar slut fast det
finns gott om ledigt
minne*



Bucket Allocation

Dela in minnet i många bitar av olika storlekar för snabbare allokering av objekt

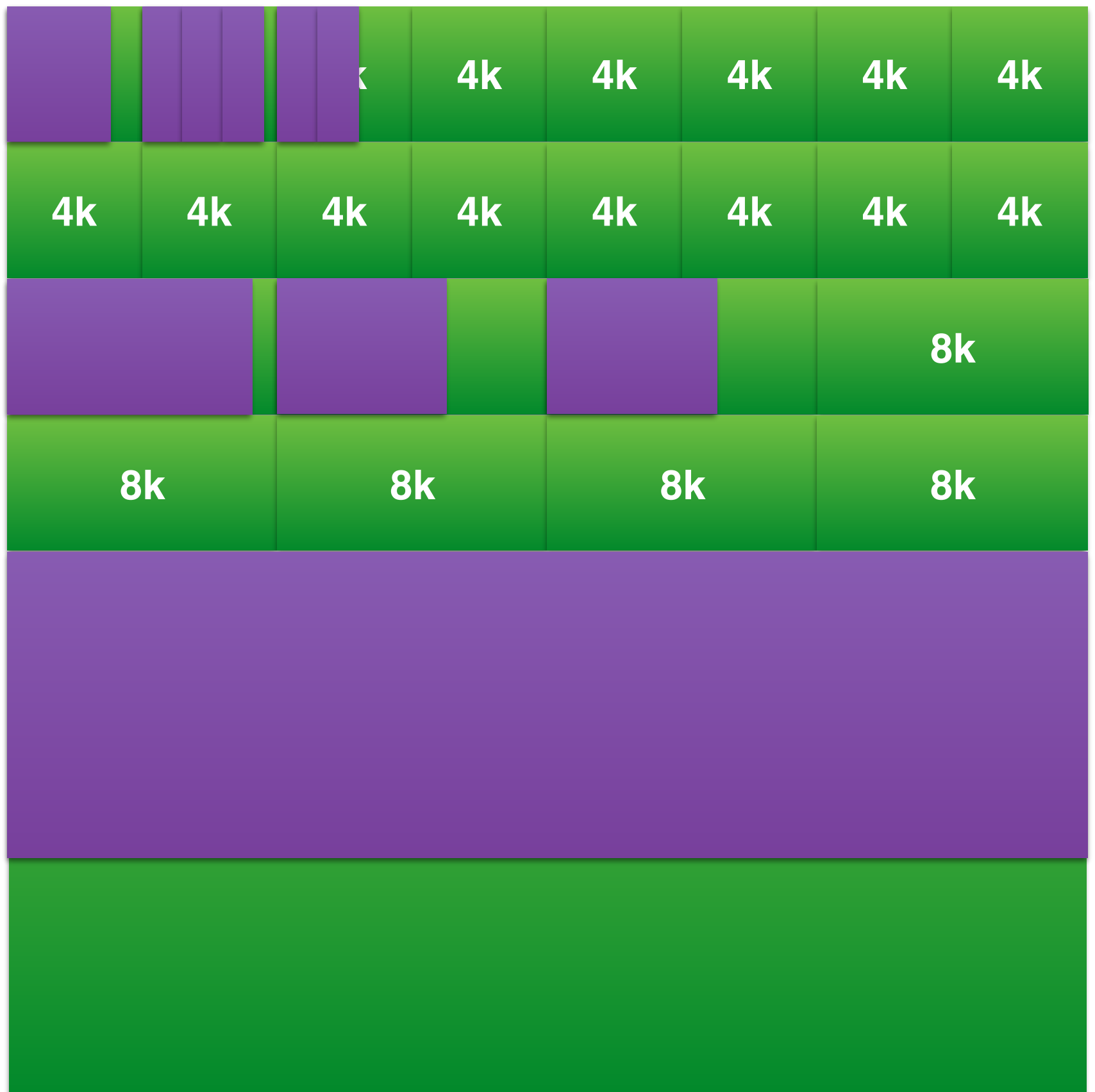
*Vad får detta för effekt
map fragmentering?*



Bucket Allocation

Dela in minnet i många bitar av olika storlekar för snabbare allokering av objekt

*Vad får detta för effekt
map fragmentering?*



Minnet i C

- Datastrukturer av dynamisk storlek bor på heapen
 - I regel länkade strukturer (men även `realloc`)
- Inget skydd för överskrivning av data i ett program
- Måste se till att allokera nog med yta
- Måste själv anropa `free` i rätt tid
- Se valgrind för verktygsstöd för att hantera minne
- "malloc är inte magisk"

Sammanfattning

Manuell minneshantering är felbenäget

Vem ansvarar för att avallokera?

Hur vet jag om "*jag*" är ansvarig?

Hur vet jag var minnet *går* att avallokera?

Typiska fel

Dubbel avallokering (double deallocation)

Skjutna pekare (dangling pointers)

Tappa bort pekaren till startadressen

Föreläsning 3 extra

Gås endast igenom kursivt på föreläsningen och i mån av tid

Hitta felen!



Hitta felet!

```
int8_t *add(int32_t count, int8_t *fst, int8_t *snd)
{
    int8_t *result = malloc(count * sizeof(int8_t));
    int8_t *cursor = result;
    while (count--) *cursor++ = *fst++ + *snd++;
    free(fst);
    free(snd);
    return result;
}
```



2. Därför måste vi "spara undan"
pekaren till startadressen

```
int8_t *add(int32_t count, int8_t *fst, int8_t *snd)
{
    int8_t *result = malloc(count * sizeof(int8_t));
    int8_t *cursor = result;
    while (count--) *cursor++ = *fst++ + *snd++;
    free(fst);
    free(snd);
    return result;
}
```

1. Vi flyttar pekaren i minnet

3. Vi glömde det mönstret
för fst och snd!



Hitta felet!

```
int8_t *add(int32_t count, int8_t *fst, int8_t *snd)
{
    int8_t *result = malloc(count * sizeof(int8_t));
    while (count--) result[count] = fst[count] + snd[count];
    free(fst);
    free(snd);
    return result;
}
```



2. Men vad händer om `fst == snd`?

```
int8_t *add(int32_t count, int8_t *fst, int8_t *snd)
{
    int8_t *result = malloc(count * sizeof(int8_t));
    while (count--) result[count] = fst[count] + snd[count];
    free(fst);
    free(snd);
    return result;
}
```

1. Kod med arrayindex är tydligare



Aliasering

- ”Två eller fler variabler avser samma objekt”

Förändring via en väg synlig genom en annan

- Kraftfullt!
- Livsfarligt!
- Fundamentalt problem i programspråksfältet

```
*x = 1;  
*y = 0;  
printf("%d\n", *x);
```



Vad skrivs ut av
detta program?

OBS! Ej tillräcklig information
på denna bild för att kunna
besvara den frågan!

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

int8_t *add(int32_t count, int8_t *fst, int8_t *snd)
{
    int8_t *result = malloc(count * sizeof(int8_t));
    while (count--) result[count] = fst[count] + snd[count];
    free(fst);
    free(snd);
    return result;
}

int main(void)
{
    int8_t even[] = { 2, 4, 6, 8 };
    int8_t odd[] = { 1, 3, 5, 7 };

    int8_t *sum = add(4, odd, even);

    for (i = 0; i < 4; ++i) printf("%d ", sum[i]);

    free(sum);
    return 0;
}
```

Hitta felet!


```

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

int8_t *add(int32_t count, int8_t *fst, int8_t *snd)
{
    int8_t *result = malloc(count * sizeof(int8_t));
    while (count--) result[count] = fst[count] + snd[count];
    free(fst);
    free(snd);
    return result;
}

```

1. Samma kod
som "sist"

```

int main(void)
{
    int8_t even[] = { 2, 4, 6, 8 };
    int8_t odd[] = { 1, 3, 5, 7 };

    int8_t *sum = add(4, odd, even);

    for (i = 0; i < 4; ++i) printf("%d ", sum[i]);

    free(sum);
    return 0;
}

```

2. even och odd
är allokerade på
stacken!

*Hitta
felet!*

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

int8_t *add(int32_t count, int8_t *fst, int8_t *snd)
{
    int8_t *result = malloc(count * sizeof(int8_t));
    while (count--) result[count] = fst[count] + snd[count];
    free(fst);
    free(snd);
    return result;
}

int main(void)
{
    int8_t *ns = malloc(4 * sizeof(int8_t));
    ns[0] = 1; ns[1] = 3; ns[2] = 5; ns[3] = 7;

    int8_t *sum = add(4, ns, ns);

    for (i = 0; i < 4; ++i)
    {
        printf("%d ", sum[i]);
    }

    free(sum);
    return 0;
}
```



add
anropas
"med
samma
array"

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

int8_t *add(int32_t count, int8_t *fst, int8_t *snd)
{
    int8_t *result = malloc(count * sizeof(int8_t));
    while (count--) result[count] = fst[count] + snd[count];
    free(fst);
    free(snd);
    return result;
}

int main(void)
{
    int8_t *ns = malloc(4 * sizeof(int8_t));
    ns[0] = 1; ns[1] = 3; ns[2] = 5; ns[3] = 7;

    int8_t *sum = add(4, ns, ns);

    for (i = 0; i < 4; ++i)
    {
        printf("%d ", sum[i]);
    }

    free(sum);
    return 0;
}
```



Föreläsning 4

Lars-Henrik Eriksson
Kursansvarig

Föreläsningssbilderna är baserade på bilder gjorda av Tobias Wrigstad.

Resten av kursen...



Så här långt borde du ha...

1. Gått på 3 föreläsningar
2. Gjort klart 5 labbar
3. Ha tittat runt på kursens webbsida
4. Ha skaffat ett GitHub-konto och registrerat det hos oss
5. Ha loggat in på Piazza minst en gång
6. ...

Från och med nästa vecka: Labbar på
100PM avser tid att redovisa och få
hjälp med programmering.

**Den mesta programmeringen sker
utanför schemalagd tid!**



Din uppgift

**Att lära dig så
mycket som
möjligt om
imperativ- och
objektorienterad
programmering,
verktyg, program-
meringsmetodik
och process**

Vår uppgift

**Att hjälpa dig i din
inlärningsprocess.**

**Att se till att
det leder till att du
blir godkänd på
kursen.**



IOOPM 2019



Imperativ och ObjektOrienterad ProgrammeringsMetodik



Imperativ och ObjektOrienterad ProgrammeringsMetodik

Del 1 & 2



Imperativ och ObjektOrienterad ProgrammeringsMetodik

Del 2



Imperativ och **O**bjekt**O**rienterad **P**rogrammerings**M**etodik

Del 1, 2 & 3



Efter godkänd kurs ska studenten kunna [1/2]

- förklara hur program **exekverar** och beskriva hur program **lagrar och hanterar information**.
- förklara och exemplifiera **kvalitativa** och **kvantitativa aspekter** av ett programs **design** och **implementation**.
- förklara skillnaden mellan **manuell och automatisk minneshantering** samt grundläggande relaterade begrepp (som stack, heap, statisk minnesarea) och använda **verktyg** för felsökning av minnesfel.
- förklara hur **större programuppgifter** kan lösas & resonera om olika lösningsalternativ.
- redogöra för hur **enkla parallelliserbara problem kan lösas** effektivt med relevanta hjälpmedel.
- **designa, koda, granska, testa, felsöka och dokumentera** egna program, med hjälp av lämpliga **verktyg**.
- **läsa, förstå och modifiera** icke-trivial **kod** som studenten själv inte har skrivit samt **integrera nyskriven kod med existerande**.



Efter godkänd kurs ska studenten kunna [2/2]

- beskriva – **oberoende av programkoden** – den uppgift ett program skall lösa och de förutsättningar som krävs för att det skall kunna arbeta. Motivera varför programmet under dessa förutsättningar är korrekt, **samt specificera och konstruera testfall och köra tester för att verifiera detta.**
- skriva lämplig **dokumentation för programmering** och **testhantering.**
- tillämpa specifika **utsnitt av kända utvecklingsprocesser** och -metodiker (ex. **agil utveckling, parprogrammering** och testdriven utveckling).
- beskriva olika former av **testning** och deras vikt i olika skeden av **utvecklingsprocessen.**
- bidra till ett **konstruktivt samarbete i programmeringsprojekt.**
- **presentera** och **diskutera** kursens innehåll **muntligt** och **skriftligt** med för utbildningsnivån lämplig färdighet.



Mål (achievements)



Achievements Group Dashboard Lars-henrik's Profile Sign out

Welcome to Achievement Unlocked for IOOPM 2019

This is where you find the list of [achievements](#) and register demonstrations during lab hours. Send an email to ioopm@it.uu.se if you cannot sign in.

All other information can be found on [the course homepage](#)

Klicka här!

Strängar i C är
Bara arrayer av chars
som slutar med '\0'



Mål

A. Abstraktion

	Name	Short desc	Grade level	Assessment
	A1	Procedurell abstraktion	3	L
	A2	Objektorienterad abstraktion	3	L
	A3	Informationsgömning	4	L
	A8	Gränssnitt	4	L

B. Arv

	Name	Short desc	Grade level	Assessment
	B4	Arv och subtypspolymorfism	3	L
				

ID

Ingress/namn

Färdig (grön) eller ofärdig (röd)

Nivå

3 L = Demonstrated during the labs.
4 S = See the achievement specification.
5 R = Project report

Hur målet kan redovisas



Mål

A. Abstraktion

	Name	Short desc
	A1	Procedurell abstraktion
	A2	Objektorienterad abstraktion
	A3	Informationsgömning
	A8	Gränssnitt

B. Arv

	Name	Short desc
	B4	Arv och subtypspolymorfism

2.1 A1: Procedurell abstraktion

Lab

G3

Tillämpa procedurell abstraktion på ett konsekvent sätt för att öka läsbarheten och undvika upprepningar.

Abstraktion är en av de viktigaste programmeringsprinciperna. Vi vet att djupt, djupt nere under huden är allt bara ettor och nollor (redan detta är en abstraktion!), men ovanpå dessa har vi byggt lager på lager av abstraktioner som låter oss tala om program t.ex. i termer av strukturer och procedurer.

Procedurell abstraktion handlar om att separera användande från implementation. Varje procedur utför – idealiskt – en enda, väl-specifierad funktion, t.ex. en viss beräkning, en förändring av programmets tillstånd, uppdaterar en datastruktur, stänger en fil, etc.

En väldigt tydlig procedur (funktion) t.ex. `open_file_for_reading()` ger en tydlig beskrivning av *vad* den gör, men *hur* den gör det – dess implementation – är inkapslad och inte synlig utifrån. Använder den systemanropet `fopen()` eller någon annan funktion? **Det skall vi strunta i!** Det är inte^[1] något vi skall behöva bry oss om, och dessutom något som kan komma att ändras, eller skilja sig beroende på vilken dator jag kör programmet på.

Ta som tumregel att varje procedur skall kunna beskrivas enkelt och kortfattat. En procedur vars beteende inte går att beskrivas enkelt och kortfattat skall förmodligen brytas upp i flera mindre procedurer.

I ett nötskal handlar procedurell abstraktion alltså om att *kapsla in* alla “conceptual units of behaviour” i en *procedur*. En procedur är ungefär detsamma som en *funktion* och många programspråk (t.ex. C) gör ingen skillnad på dem.

2.1.1 Exempel

Proceduren `ritaEnCirkel(int radie, koordinat center)` utför beräkningar och tänder individuella pixlar på en skärm, men i och med att dessa rutiner kapslats in i en procedur med ett vettigt namn, där indata är uttryckt i termer av koordinater och radie har vi abstraherat bort dessa detaljer, och det blir möjligt att rita cirklar tills korna kommer hem utan att förstå hur själva implementationen ser ut.

Väl utförd abstraktion döljer detaljer och låter oss fokusera på färre koncept i taget.

2.1.2 Redovisning

Du bör ha en klar uppfattning om bland annat:

- Varför det är vettigt att identifiera liknande mönster i koden och extrahera dem och kapsla in dem i en enda procedur som kan anropas istället för upprepningarna?
- Abstraktioner kan “läcka”. Vad betyder det och vad får det för konsekvenser?
- Vad är skillnaderna mellan “control abstraction” (ex. if-satsen är en abstraktion) och “data abstraction” (ex. en lista är en abstraktion)? Du kan läsa om dessa koncept på t.ex. [http://en.wikipedia.org/wiki/Abstraction_\(computer_science\)](http://en.wikipedia.org/wiki/Abstraction_(computer_science)).
- Ge exempel på procedurell abstraktion i ditt program! Kritisera den! Kan den förbättras?
- Vad är den kortfattade beskrivningen av vad funktionerna gör?
- Ger namnen på funktionerna en bra ledning om vad funktionerna gör?
- Finns det exempel på **läckande abstraktioner**, dvs. där den som anropar funktionen måste känna till hur funktionen faktiskt är implementerad för att fungera, eller förutsätter en viss implementation?
- Låt f_1, f_2 och f_3 vara funktioner. f_1 och f_2 är delar av samma bibliotek och f_2 använder f_3 i sin implementation. Skiljer sig nivån av abstraktion mellan dessa på något sätt? Hur?
- Är abstraktion möjlig utan inkapsling? (Bonusfråga)

Redovisning

- **Din uppgift:** att övertyga examinatorn om att du uppfyller målet

Ge **exempel** från den kod (etc.) du har skrivit

Inga **core dumps**

Ha en **story** för hur allting hänger ihop

- Försök att alltid redovisa **> 1 mål åt gången**

Mindre arbete, mindre väntetid

Synergi!

- **Demo av redovisningssystemet sker på föreläsning innan första redovisningstillfället**



Var finns information om kursen?

Imperative & Object-Oriented Programming Methodology

Table of Contents

1. How IOOPM Works
2. Lecture Schedule with Slides
3. Deadlines
4. Exercises, Assignments and Projects
5. Resources
6. Recent Changes

Imperative & Object-Oriented Programming Methodology

Danger

As long as this note remains on this page, any information is subject to change and broken links, etc. are to be expected. Please be restrictive in reporting any errors for now.

Note

Get course material at the [course's GitHub page](#)
Ask questions and discuss with other students on [Piazza](#)
Request demonstrations and track your progress at the [AU Portal](#)

1 How IOOPM Works

If you are visiting this page for the first time, you should probably [read the page about the course first](#). **Mandatory reading!**

Author: Tobias Wrigstad
Created: 2018-08-22

wrigstad.com/ioopm19

Hur interagerar jag med er?

Class at a Glance Updated Just now. Reload

1 unread posts 145 total posts
1 unanswered questions 650 total contributions
8 unresolved followups 108 instructors' responses
35 students' responses
135 min avg. response time

Student Enrollment ...out of 25 (estimated) Edit
149 enrolled

Dear Professors:

Last Fall, we launched a new service called Piazza Careers. And through this service, we have connected students with other students studying similar subjects, with recent alumni now in industry, and with potential employers. Many of them have successfully secured internships and jobs through our service!

Just as Piazza was born out of *my deeply personal struggle* to get help when I was stuck as a shy student (one of few women in my Computer Science class), Piazza Careers is born out of the challenges I faced grappling with what to do with my life - how and where to pursue my passions. We at Piazza have been at the first problem now for over 4 years and are just starting to embark on the second.

We have a vision for Piazza Careers, and like with Piazza Q&A we can't get there without your feedback. We are eager to make a real impact on students' lives, helping enhance their experience in their classes as well as their careers. It will require a lot of iteration and hard work. We'll inevitably make a few mistakes along the way and we ask for your support and forgiveness in those moments.

The Piazza Careers service is essentially a 'third tab' that students can choose to access when desired. Students can remove this additional tab at any time. Our foremost priority is to protect students' privacy and keep their learning distraction-free, and we will:

- Remain FERPA compliant and provide students with complete control over what information is shared with companies using Piazza Careers. By default, no information is shared.
- Not expose any email addresses, contact information, enrollment information, or questions and answers posted to Piazza classes to companies using Piazza Careers.
- Keep the integration of Piazza Careers within the current Piazza Q&A as minimal as possible to prevent any type of distraction.

Piazza + Labbar

Hur lämnar jag in en uppgift? (z-mål)

IOOPM-UU / ioopm15

Course repository for IOOPM, 2015 — Edit

11 commits 1 branch 0 releases 1 contributor

Branch: master ioopm15 / +

Mer Emacs

- extramaterial/lathundar Experimentar med relativa länkar 2 hours ago
- uppgifter/fas1 Mer Emacs 27 minutes ago
- gitignore Lade till .gitignore för emacs temp-filer an hour ago
- README.md Initial commit a day ago

ioopm15

Course repository for IOOPM, 2015

GitHub + Redovisning

Hur redovisar jag?

IOOPM Achievements Sign in

Unlockable Achievements (aka kursmål/kunskapsmål)

Assessment Legend

L = Demonstrated during the labs.
S = Demonstrated in a way described in the achievement specification.
R = Demonstrated as part of the project report (plus deliverable and demonstration).

Filter: ☒ Collapse ☒ Grade 3 ☒ Grade 4 ☒ Grade 5

Inlämningsmål

Z. Inläppar

Name	Short desc	Grade level	Assessment
Z100	Inläpp 1	3	L
Z101	Inläpp 2	3	L
Z103	Inläpp 3	3	L
Z104	Inläpp 4	3	L

AU Portal + Labbar

Du börjar här:

Imperative & Object-Oriented Programming Methodology - qutebrowser

1: Imperative & Object-Oriented Programming Methodology

Imperative & Object-Oriented Programming Methodology

Table of Contents

- 1. How IOOPM Works
- 2. Lecture Schedule with Slides
- 3. Deadlines
- 4. Exercises, Assignments and Projects
- 5. Resources
- 6. Recent Changes

1 How IOOPM Works

If you are visiting this page for the first time, you should probably [read the page about the course first](#). **Mandatory reading!**

1.1 What You Should be Doing the First Week

Get course material at the [course's GitHub page](#)
Ask questions and discuss with other students on [Piazza](#)
Request demonstrations and track your progress at the [AU Portal](#)

Author: Tobias Wrigstad
Created: 2018-08-22 ons 16:50

<http://wrigstad.com/ioopm18/> [top] [1/1]

wrigstad.com/ioopm19

Översikt

C-bootstrap (2v, 2/9 – 13/9)

/ Sprint 1 = Inlupp 1 (3v, 16/9 – 4/10)
Fas 1 (C) < (+ 15 nivå-3 mål för godkänt)
\ Sprint 2 = Inlupp 2 (2v, 7/10 – 18/10)

Java-bootstrap (1v, 21/10 – 25/10)

Kodprov 1 (i första hand C, 23/10)

/ Sprint 1 = Inlupp 3 (3v, 28/10 – 15/11)
Fas 2 (Java) < (+ 15 nivå-3 mål för godkänt)
\ Sprint 2 = Inlupp 4 (2v, 18/11 – 29/11)

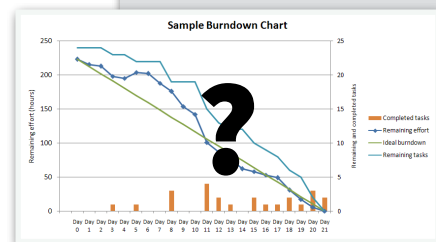
/ Sprint 1 (Er egen indelning under projektet, 4-5v, 2/12 – 16/1)
Fas 3 (Projekt) < ... (+ 4 nivå-3 mål för godkänt)
\ Sprint n

Kodprov 2 (i första hand Java, 10/12)

Kodprov 3 (i första hand omprov, 8/1)

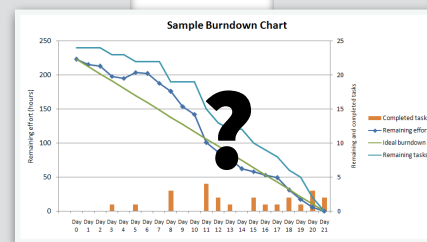


Fas [kursen har 3, var och en med flera sprintar]



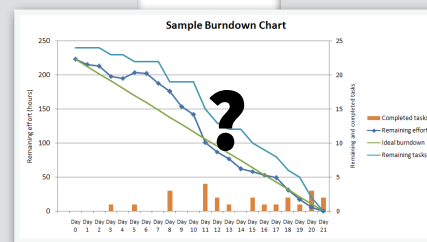
Gör en plan

Revidera!



Gör en plan

Revidera!



Gör en plan

UM!

sprint

UM!

sprint



Sprint [varje fas har flera]

Uppgift

Enkelt lagerhanteringssystem

Denna uppgift går ut på att skriva ett enkelt lagerhanteringssystem. Det kommer att ta många timmar att skriva färdigt. **Målet med dagens övning är att komma igång.**

Systemet skall hålla reda på en "databas" med varor. Programmet skall ha en huvudloop där användaren får ett antal alternativ:

- Lägg till en vara
- Ta bort en vara
- Redigera en vara
- Ängra senaste ändringen
- Lista hela varukatalogen
- Avsluta

Huvudloopen skriver ut alternativen och frågar vad användaren vill göra. De olika valen tar användaren ned i olika funktioner (se nedan) med eget betyende. Efter att sådana funktioner har avslutats återvänder de till huvudloopen igen. Detta förflöende fortsätter tills användaren väljer att avsluta programmet. Exempel:

```

-----
Välkommen till lagerhantering 2.0
-----
1) Lägg till en vara
2) Ta bort en vara
3) Redigera en vara
4) Ängra senaste ändringen
5) Lista hela varukatalogen
6) Avsluta

Vad vill du göra idag? _
-----
aktuella platsen.

```

Implementera

```

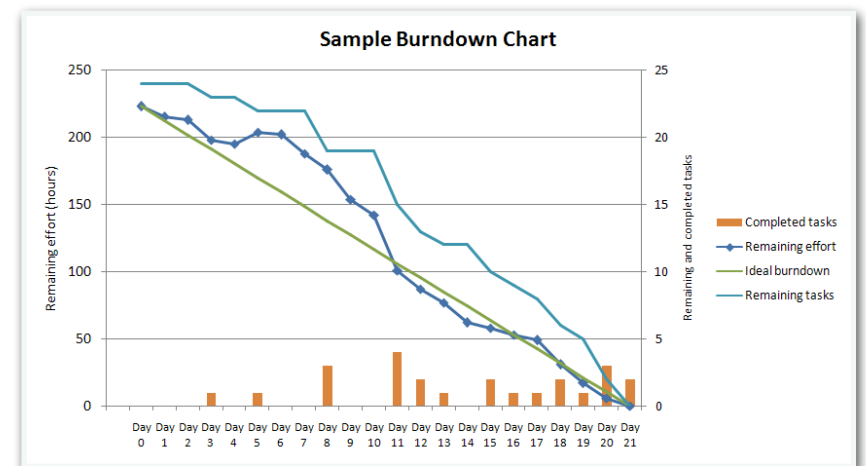
embed void
this->nodes = calloc(k, sizeof(node_t));
end

def insert(v:int) : bool {
  embed void
  node_t **n = (node_t**) &this->root;
  while (*n)
  {
    node_t *c = *n;
    if (c->value == v) { return false; }
    n = (c->value < v) ? &(c->left) : &(c->right);
  }
  ((node_t*)this->nodes)[this->size++] = (node_t *)
  { .value = v };
  *n = &((node_t*)this->nodes)[this->size-1];
end;
true;
}

def reset() : void
  paxx.enc 5% (34,0) [[Encore +2 yas]] 10:37AM 1.70[50%]

```

För bok över dina framsteg



Unlockable Achievements (aka kursmål/kunskapsmål)

Name	Short desc	Grade level	Assessment	Show	Edit	Destroy
A1	Konkret tillämpad procedurabstraktion för att lösa lösbarheten och undvika	3	L	Show	Edit	Destroy
A2	Tillämpa objektorienterad abstraktion för att lösa implementationsdetaljer bakom	3	L	Show	Edit	Destroy
A3	Designera förläggare för designprincipen informationsgenomgång i ett C-program	4	L, G	Show	Edit	Destroy
A4	Med hjälp av c och j-diller	3	L	Show	Edit	Destroy
A5	Ärskilda av, metodspecifikation och superenhet i ett program som visar att	3	L	Show	Edit	Destroy
A6	Ärskilda både överlagrade och specialiserade konstruktioner på lämpligt sätt i	3	L	Show	Edit	Destroy
A7	Förklara hur arvsläggning har använts i ett program för att separera	4	G, T	Show	Edit	Destroy
A8	Förklara begreppet designmönster samt visa hur minst ett designmönster har	4	G	Show	Edit	Destroy
A9	Vad har man kan separera gränssnitt från implementation med hjälp av Java	4	L, G	Show	Edit	Destroy
A10	Designera ett objektmodell med hjälp av Java	3	L	Show	Edit	Destroy
A11	Ärskilda parametriska polymorfism för ökad "vanliga typskickar" vid interaktion	3	L	Show	Edit	Destroy
A12	Designa med parametrisk polymorfism för att göra relevanta delar av ett program	4	L, G	Show	Edit	Destroy
A13	Översätta mellan rekursiva och iterativa lösningar av samma problem samt	3	L	Show	Edit	Destroy
A14	Förklara skillnaderna mellan deklarativ och imperativ; diskutera dessa respektive	5	G, T	Show	Edit	Destroy
A15	Ärskilda kunskapen om förläggare för att undvika informationsproblem i ett C- eller	3	L	Show	Edit	Destroy
A16	Ärskilda kunskapen om förläggare för att undvika informationsproblem i ett C- eller	3	L	Show	Edit	Destroy
A17	Ärskilda kunskapen om förläggare för att undvika informationsproblem i ett C- eller	3	L	Show	Edit	Destroy
A18	Ärskilda kunskapen om förläggare för att undvika informationsproblem i ett C- eller	3	L	Show	Edit	Destroy
A19	Ärskilda kunskapen om förläggare för att undvika informationsproblem i ett C- eller	3	L	Show	Edit	Destroy
A20	Ärskilda kunskapen om förläggare för att undvika informationsproblem i ett C- eller	3	L	Show	Edit	Destroy
A21	Ärskilda kunskapen om förläggare för att undvika informationsproblem i ett C- eller	3	L	Show	Edit	Destroy
A22	Ärskilda kunskapen om förläggare för att undvika informationsproblem i ett C- eller	3	L	Show	Edit	Destroy
A23	Ärskilda kunskapen om förläggare för att undvika informationsproblem i ett C- eller	3	L	Show	Edit	Destroy
A24	Ärskilda kunskapen om förläggare för att undvika informationsproblem i ett C- eller	3	L	Show	Edit	Destroy
A25	Ärskilda kunskapen om förläggare för att undvika informationsproblem i ett C- eller	3	L	Show	Edit	Destroy

Välj mål



Redovisa



Uppföljningsmöte

- **Sex möten under kursen**

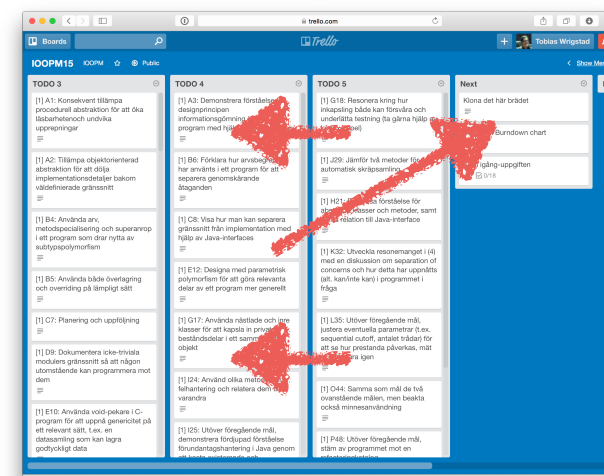
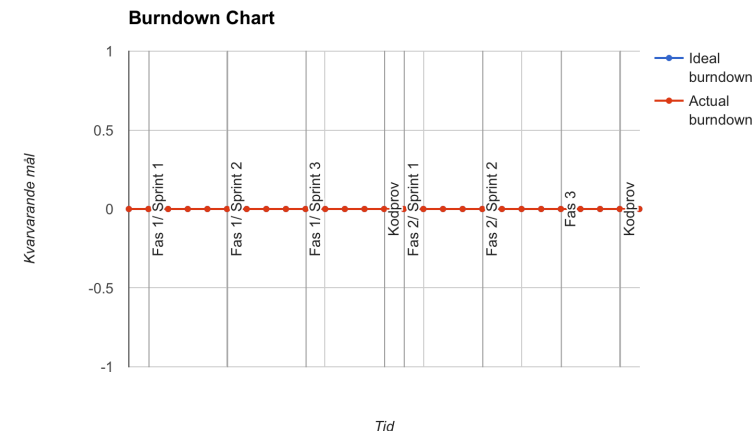
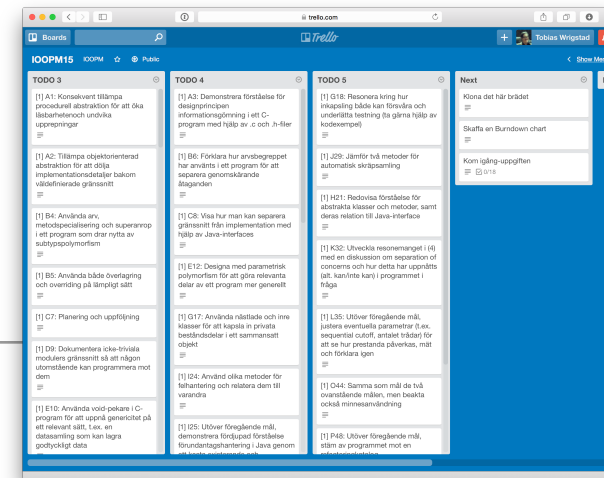
Diskussion kring och utifrån burndown chart

Tillsammans med andra studenter

I början av kursen är det fokus på planering

Vi går mot fokus på uppföljning

- Coachen kallar / har kallat till första möte "nu"



Taylor

trello.com

+

Tobias Wrigstad

IOOPM16

IOOPM

☆

Publik

TODO 3

[1] A1: Procedurell abstraktion

[1] A2: Objektorienterad abstraktion

[1] B4: Arv och subtypspolymorfism

[1] B5: Överlagring och overriding

[1] C7: Planering och uppföljning

[1] D9: Dokumentation

[1] E10: Implementera genericitet genom void-pekare

[1] E11: Parametrisk polymorfism och typsäkerhet

[1] F13: Iteration vs. rekursion

[1] G15: Aliasering

[1] G16: Namn-baserad inkapsling

[1] H19: Skillnaden mellan identitet och ekvivalens

[1] H20: Värdeöverföring

TODO 4

[1] A3: Informationsgömning

[1] B6: Genomskärande åtaganden och arv

[1] C8: Visa hur man kan separera gränssnitt från implementation med hjälp av Java-interfaces

[1] E12: Designa med parametrisk polymorfism

[1] F14: Svansrekursion

[1] G17: Nästlade och inre klasser

[1] I24: Olika metoder för felhantering

[1] I25: Egendefinierade undantag

[1] J28: Manuell vs. automatisk minneshantering

[1] K31: Coupling och cohesion

[1] M39: Pekare till pekare

[1] N41: Bindning

TODO 5

[1] G18: Resonera kring hur inkapsling både kan försvåra och underlätta testning (ta gärna hjälp av kodexempel)

[1] H21: Redovisa förståelse för abstrakta klasser och metoder, samt deras relation till Java-interface

[1] J29: Jämför två metoder för automatisk skräpsamling

[1] K32: Separation of concerns

[1] O44: Profilerings och optimering 3/3

[1] P48: Stäm av ett program mot en refactoringkatalog

[1] Q51: Andra verktyg för felsökning

[1] X62: Kommunikation 1:1

TODO Projektet

[1] Y64: Scrum/Kanban

[1] Y65: Konsekvent tillämpa en kodstandard

[1] Y66: Tillämpa kodgranskning löpande

[1] Y67: Parprogrammering

[1] Y68: Redovisa en fungerande projektuppgift

[1] Y69: Tillämpa regressionstestning under projektet

Next

Klona det här brädet

Skaffa en Burndown chart

Meny

Gå med i tavla

Gruppmedlemmar kan gå med i denna tavla.

Filtrera kort

Mer

Aktivitet

Elias Castegren gjorde denna tavla synlig för publika för 11 minuter sedan

Elias Castegren tog bort kortet #1 från Next I måndags 16:30

Elias Castegren flyttade [1] Y69: Tillämpa regressionstestning under projektet från TODO 3 till Projektet I måndags 16:25

Elias Castegren flyttade [1] Y68: Redovisa en fungerande projektuppgift från TODO 3 till Projektet I måndags 16:25

Elias Castegren flyttade [1] Y67: Parprogrammering från TODO 3 till Projektet I måndags 16:25

Elias Castegren flyttade [1] Y66: Tillämpa kodgranskning löpande från TODO 3 till Projektet I måndags 16:25

Elias Castegren flyttade [1] Y65: Konsekvent tillämpa en kodstandard från TODO 3 till Projektet I måndags 16:24

Elias Castegren flyttade [1] Y64: Scrum/Kanban från TODO 3 till Projektet I måndags 16:24

Elias Castegren la till Projektet på denna tavla I måndags 16:24

trello.com

Uppgifter | Imperativ och objektorient...

Faser | Imperativ och objektorienterad...

Futures: improvements on UpScale...

Creating cards by email - Trello Help

Boards

UpScale

Doing

Integration with nev

Implement depende

Basic data structur

Module system

Add a card...

Futures: improvements

in list **Next**

Members

AN   +

Description [Edit](#)

Subsumes the following (archived) cards:

-  [Await & Suspend](#)
-  [Future chaining](#)
-  [Futures](#)
-  [Coroutines](#)
-  [Suspendable/blocking actors \(was Futures etc.\)](#)

☒

Checklist

Delete...

0%

☐ Merge "children" and "responsibility" in the future struct

☐ Trace functions for futures and future type struct

☐ Remove stupid limitations (like 16 responsibilities max) on futures

☐ Add comprehensive testing for non-deterministic behaviour on futures, chaining, await and suspend

Add an item...

Activity

 Write a comment...

Add

Members

Labels

☒ Checklist

 Due Date

 Attachment

Actions

→ Move

 Copy

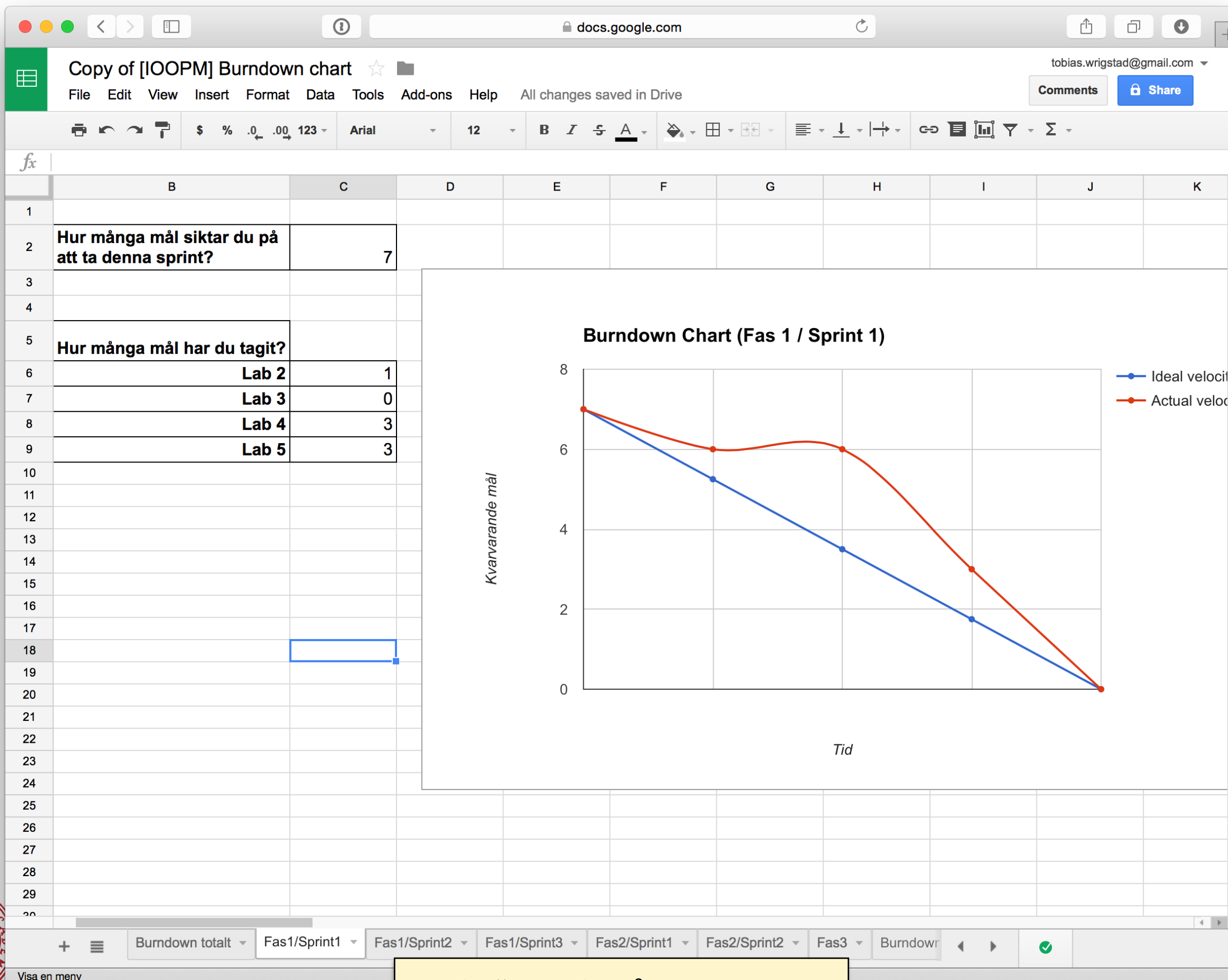
 **Subscribe** 

 Archive

[Share and more...](#)

Visa en meny

trello.com



Se "Länkar" på kurswebben

Övning i skriftlig färdighet

- På nivå 4 och 5 måste du redovisa ett mål som en essä
(För att nå nivå 3 räcker det med projektrapporten)

- Instruktioner finns på kurswebben

Omfattning: 7500 tecken

Deadline: **se kurswebben**

- Lämnas in via GitHub

ng till handledning online (t.ex. via epost)	11	13.4%
Återkoppling/feedback på inlämningar	24	29.3%
(utveckla gärna i kommentarerna nedan)	2	2.4%

Jämförelse mellan två skräpsamlingsalgoritmer Mark and Sweep mot Reference counting

Uppsala universitet
January 16, 2015

Två skräpsamlingsmetoder

Något som blir vanligare och vanligare är användningen av skräpsamlare, även känt som garbage collectors. En skräpsamlares främsta uppgift är att lämna tillbaka minne du inte använder. Så den frigör minne som du använt och inte använder längre, på ett automatiskt sätt. Denna process brukar kallas för skräpsamling. Anledningen att användningen av skräpsamlare har blivit stort skulle kunna bero på språk såsom Java, javascript, python som alla använder sig av någon typ av skräpsamlare. Varför beslutade sig folk för att använda skräpsamlare? Förmodligen för att det är svårt att hantera minnet manuellt. Om inte programmeraren har skickligheten som krävs och är på sin vakt hela tiden kan det uppstå minnesläckor eller andra problem vid manuell hantering av minne. Med en skräpsamlare behöver inte programmeraren oroa sig över sådana saker och kan ägna mer tid till andra saker.

Jag är ganska övertygad att du någon gång kommer använda dig av någon skräpsamlare om du ängar dig åt programmering. Hur mycket du tänker på det eller inte är en annan fråga. Jag kommer beskriva hur två metoder för skräpsamling går till och göra en jämförelse mellan dem.

I detta dokument ska jag beskriva två vanliga algoritmer för skräpsamling och deras skillnader. Metoderna jag kommer fokusera på är Mark and Sweep och Reference counting.

Om ett objekts referensräknare når noll så finns det inte längre några referenser till det objektet. Objektet är därmed skräp och kommer att frigöras direkt när referensräknaren når noll. Referensräknande skräpsamlare är därmed deterministiska,³ vilket innebär att vi vet exakt när ett objekt tas bort. Detta är en av de stora fördelarna med referensräkning. Detta innebär att referensräkning är kompatibelt med RAII⁴ och kan därmed användas tillsammans med destrukturer, kod som körs automatiskt när objektet förstörs.

En annan fördel med referensräkning är att arbetet för att ta reda på vad som är skräp är fördelat över hela programmet körtid, vilket inte är fallet med spårande skräpsamlare. Detta är samtidigt en av svagheterna med referensräkning eftersom det krävs en hel del extra arbete bara för att uppdatera objektens referensräknare. Att varje gång en referens skapas eller ändras så måste objektet letas upp i minnet och dess referensräknare uppdateras. Det kan medföra stora mängder missar när objektet letas upp i cacheminnet, vilket påverkar prestandan negativt.

Det kanske största problemet med referensräknande skräpsamlare är att naiva implementationer inte kan hantera cirkulära strukturer. Cirkulära strukturer innebär objekt som direkt eller indirekt refererar till sig själv. Vissa implementationer löser detta genom att använda svaga referenser som helt enkelt inte ökar objekts referensräknare. Det finns också mer komplexa algoritmer som kan hantera detta, men dessa kräver ofta stora mängder extra arbete.

2.2 Mark and sweep

Mark and sweep tillhör kategorin av tracing eller spårande skräpsamlare. Denna typ av skräpsamlare angriper problemet från motsatt håll. Istället för att hålla reda på och frigöra skräp direkt när det uppstår så kommer skräpsamling ske vid olika tidpunkter. Ofta sker detta när mängden ledigt minne tar slut eller faller under en viss nivå.

Mark and sweep har två steg som inte helt oväntat kallas för "mark" och "sweep". Första steget "mark", går ut på att hitta och markera alla levande objekt. Själva markeringen sker genom att en flagga som sparas tillsammans med varje objekt sätts. Innan "mark" steget påbörjas nollställs samtliga flaggor.

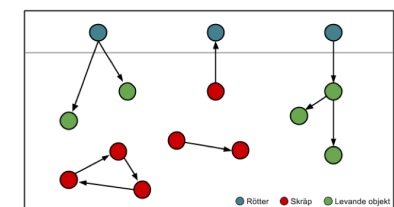


Figure 1: Cirklar representerar objekt och pilarna referenser.

För att hitta alla levande objekt så utgår skräpsamlaren från ett antal rötter. Rötterna är alla referenser eller pekare som vi vet att vi har tillgång till. I programspråket C kan dessa vara alla pekare som ligger på stacken. Skräpsamlaren går över rötterna och följer deras referenser. Varje objekt som hittas markeras och för objektets samtliga referenser så upprepas samma process som för rötterna. Alltså vi följer referenser och alla objekt som hittas markeras och deras referenser följs. Detta visualiseras i figur 1.

³Detta är inte helt sant, då i flerstegs program kan uppdateringen av referensräknare kan leda till race condition[5]

Mål och betyg (reservationer för off-by-one errors)

	3	4	5
"Vanliga" mål (A-V)	34	17	8
Varav utanför labb	4	1	3
Inluppar (Z-mål)	4 (samma för alla)		
Projekt (Y-mål)	6 (samma för alla)		



Fas 3: Projektarbete

- Arbeta 4–6 personer (vi tar fram grupperna under november)
- Uppgiften TBA

Börjar i december

Slutar i januari (se kurswebben för detaljer)

- Lämnas in via GitHub
- Presentation och verkstad med annan grupp
- Grupperna lägger själva upp sprintar
- 1–2 KLOC, plus tester

3

Uppgiften

Uppgiften går ut på att utveckla ett bibliotek, för enkelhets skull kallat "gc", för minneshantering i form av en kompakterande skräpsamlare. Med funktionen `h_init` kan en användare reservera en egen "heap" – ett konsekutivt minnesblock¹ i vilket man sedan kan allokera minne med hjälp av biblioteksfunktioner. Detta minne skall sedan hanteras automatiskt – när minnet tar slut skall skräpsamling automatiskt triggas, och alla objekt i detta minne som inte är nåbart via någon rot i systemet tas bort².

Av pedagogiska skäl beskriver vi först skräpsamling med hjälp av mark-sweep, som vi *inte* skall använda innan vi går in på den kompakterande skräpsamlaren som använder en liknande algoritm.

3.1 Skräpsamling med mark-sweep

Skräpsamling med mark-sweep vandrar genom (traverserar) den graf som heapen utgör för att identifiera objekt som säkert kan deallokeras utan att programmet kraschar. Vi går igenom algoritmen steg-för-steg nedan.

Vi kan tänka oss att varje objekt innehåller en extra bit³, den s.k. *mark-biten*. När denna bit är satt (1) anses objektet vara "vid liv". Annars är objektet skräp som kan tas bort.

Vid skräpsamling sker följande (logiskt sett):

- Steg 1 Iterera över samtliga objekt på heapen och sätter mark-biten till 0. Detta innebär att alla objekt anses vara skräp initialt.
- Steg 2 Sök igenom stacken efter pekare till objekt på heapen⁴, och med utgångspunkt från dessa objekt, traversera heapen och markera alla objekt som påträffas genom att mark-biten sätts till 1.
- Steg 3 Iterera över listan över samtliga objekt på heapen och frigör alla objekt vars mark-bit fortfarande är 0.

Steg 2 kallas för "mark-fasen" och steg 3 för "sweep-fasen", härav algoritmens namn, *mark-sweep*.

OBSERVERA

Denna del av specifikationen är ett *levande dokument* som kan komma att uppdateras och förändras under projektets gång.

¹ T.ex. med hjälp av `malloc` i `stdlib.h`, eller `mmap` i `sys/mman.h`.

² Vi gör en förenkling och utgår från att programmen är enkeltrådade och att endast en heap skapas per program.

³ Tekniskt kan det också vara en bit om man har en över. Ibland kan man packa in bitar i annat data – vi skall se exempel på det senare i denna text!

⁴ Dessa pekare kallar vi också för "rötter".



Kodprovet (2x2,5 HP)

- Två frågor — en C, en Java

Individuellt prov i datorsal, 3 timmar — ingen tillgång till Internet

- Syftet: att tvinga alla att sitta i framsätet vid parprogrammering

Examinerar inte kursmål!

- Går att ta i steg (klara en fråga på varje prov)
- Minst tre provtillfällen under kursen

23/10 och 10/12 och 8/1

- Anmälan annonseras i Piazza



Högskolepoäng

	HP	Deadline
Fas 1	5	V44
Kodprov C	2,5	* Oktober
Fas 2	5	V50
Kodprov Java	2,5	* December
Fas 3	5	V3

Ungefärliga tidpunkter! Se kurswebben för detaljer.



Simple [minimal sammanfattning]

1. Läs specifikationen och leta specifikt efter **verb** (funktioner/beteende), eller **substantiv** (data/objekt/struktur) — gör en work breakdown structure
2. Skriv kod för att pröva om du tänkt rätt (vad är rätt – hur man kollar det?)
3. Ha alltid ett fungerande program
4. Kompilera efter varje förändring
5. Kör programmet hela tiden för att hitta fel (eller ännu bättre — kör testen!)
6. Dela upp alla problem i delproblem, gå till 7. först när något är enkelt
7. Dela upp alla delproblem i mindre steg — gör de enklaste först
8. **Fuska** (cheat) varje gång du riskerar att fastna
9. **Skarva** (dodge) för att förenkla specifikationer och skapa fler enklare delsteg
10. Växla mellan att: tänka, koda och ibland refaktorera (speciellt **fusk** och **skarvar**)



Simple

- Om du inte redan är en programmerare måste du använda SIMPLE under kursen
- Finns detaljerad beskrivning på kurswebben
Använder Lab 4, 5 och inlupp 1 som löpande exempel
- Det kan vara svårt att ta in allt direkt, så börja med det som verkar enkelt
Försök inte göra rätt, utan det som känns rätt — gå tillbaka till texten när det behövs



Att använda en texteditor

- Under kursen kommer vi att använda **Emacs**

Under fas 2 är de tillåtet att använda IDE:er,
men inte rekommenderat

- Emacs kan också betyda vim men **inte**

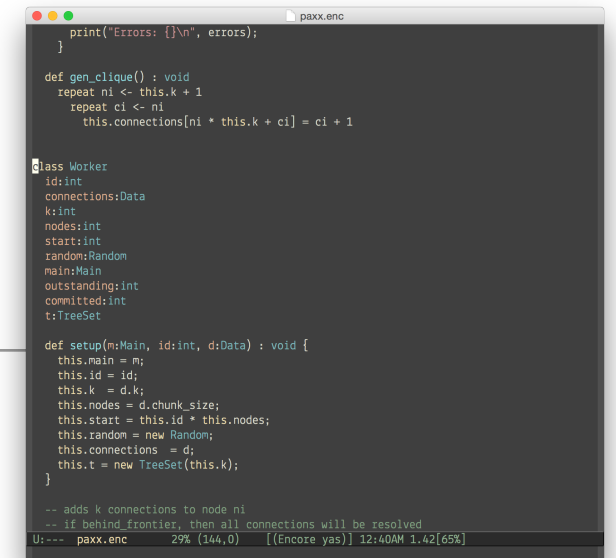
Gedit

Nano

Notepad++

Sublime

...



```
print("Errors: {}\n", errors);
}

def gen_clique() : void
repeat ni <- this.k + 1
repeat ci <- ni
this.connections[ni * this.k + ci] = ci + 1

class Worker
id:int
connections:Data
k:int
nodes:int
start:int
random:Random
main:Main
outstanding:int
committed:int
t:TreeSet

def setup(m:Main, id:int, d:Data) : void {
this.main = m;
this.id = id;
this.k = d.k;
this.nodes = d.chunk_size;
this.start = this.id * this.nodes;
this.random = new Random;
this.connections = d;
this.t = new TreeSet(this.k);
}

-- adds k connections to node ni
-- if behind frontier, then all connections will be resolved
U:--- paxx.enc 29% (144,0) [[Encore yas]] 12:40AM 1.42[65%]
```



Sammanfattningsvis

- Från och med nästa vecka skall du jobba med inlämningsuppgifterna
Jobba i ett programmeringspar
- Labbarna är till för redovisning och hjälp och är **inte obligatoriska**
- Kursen kretsar till 75% kring redovisning av **mål** som finns beskrivna på kursens webbsida
- Alla mål redovisas i par men betygsätts individuellt
- Du förväntas själv göra kopplingen mellan mål och uppgifter
Viss handledning finns i form av tips i uppgiftstexterna
- **Implementera först, redovisa sedan**