

**Är labbarna bara till
för redovisning?**



NEJ



**Måste jag göra
övningarna som delas
ut till varje
föreläsning?**



NEJ



Du kan redovisa
mer än dubbelt
så många mål som
behövs för G på de
labbar som är kvar!



BRA



IOOPM 2015

IMPERATIV ... OCH SÅ VIDARE

[DEADLINES](#)[OM KURSEN](#)[PROCESS](#)[MÅL](#)[UPPGIFTER](#)[LÄNKAR](#)[KONTAKT](#)[HJÄLP!](#)

KURSDELAR

FAS 01

Introduktion till imperativ
programmering i C. 31:a augusti till
23:e oktober.

IMPERATIV OCH OBJEKTORIENTERAD PROGRAMMERINGS- METODIK

Välkommen till sidorna för kursen IOOPM -- imperativ och objektorienterad
programmeringsmetodik! Dessa sidor är **nu** uppdaterade för HT2015.

LÄNKAR

Föreläsningsschema

AU PÖ

GitHub

Piazza

Fas 1 (Imperativ)

Fas 2 (Objektorientering)

Fas 3 (Projekt)

Skriftlig framställning

Presentationsteknik

FAS 1: IMPERATIV PROGRAMMERING I C

Denna fas består av tre sprintar och därmed tre inlämningsuppgifter. Ämnet för fasen är imperativ programmering i C. Vi kommer att lära oss C från grunden, verktyg för C-programmering som [valgrind](#), [gdb](#), m.fl., defensiv programmering, manuell minneshantering, testning, och hur man skriver läsbar kod.

Föreläsningarnas (planerade) innehåll

Typ	Innehåll
F1	Introduktion till kursen . Skillnad mellan funktionell och imperativ programmering. SIMPLE . Det agila upplägget . Huvudmål: C7 (F13, F14)
F2	Grundläggande datatyper (inkl. arrayer), deklaration, uttryck och satser. Huvudmål: A1 (F13) <i>Lämpliga övningar (exercises) från Shaws bok: 0, 1, (2), 3, (4), 5, 6, 7, 8, 9, 10, 11, 12, 14, (15), (21). Övningar inom parentes tas upp igen senare mer detaljerat.</i>
F3	Introduktion till standard-I/O och grundläggande stränghantering. Huvudmål: G15 <i>Lämpliga övningar (exercises) från Shaws bok: 13, 24.</i>
F4	Poster och unioner. Typnamnsdefinitioner, separatkompilering och headerfiler. Huvudmål: A3, D9, G15, G18, K30 <i>Lämpliga övningar (exercises) från Shaws bok: (15), 16, (22). Övningar inom parentes tas upp igen senare mer detaljerat.</i>
Lektion 1	Komma igång med inlämningsuppgift 1 och GDB. Huvudmål: R52
F5	Dynamisk minnesallokering. Pekare. Huvudmål: E10, H20, J26, J27, M38
F6	Automatisering (Make, scripting). Huvudmål: T55, U57, V58 <i>Lämpliga övningar (exercises) från Shaws bok: 32.</i>
F7	Pekare och arrayer. Dynamiska arrayer. Pekararrayer. Kommandoradsargument. Konvertering till och från strängar. Huvudmål: E10, H20, J26, M36 <i>Lämpliga övningar (exercises) från Shaws bok: 15, 34, 35.</i>

**Var kommer eventuell
ny information om
kursen?**



Piazza

The screenshot shows the Piazza Q&A interface for course 1DL221. The top navigation bar includes links for Q&A, Resources, Statistics, and Manage Class. The user profile of Tobias Wrigstad is visible in the top right. The left sidebar contains a 'New Post' button, a search bar, and a list of filters (Unread, Updated, Unresolved, Following). Below the filters, a 'Filtering by: Unanswered Questions' section is shown, followed by a 'TODAY' section with a post titled 'Private Malloc & free' and a 'LAST WEEK' section with two posts. The main content area features a 'Class at a Glance' summary with statistics: 60 total posts, 275 total contributions, 33 instructors' responses, 18 students' responses, and a 26 min avg. response time. Below this is a 'Student Enrollment' bar showing 135 enrolled out of 120 (estimated). The bottom section contains a message from the Piazza team to professors, detailing the launch of Piazza Careers and the team's vision for the service.

1DL221

1DL221 Q & A Resources Statistics Manage Class

fas1 fas2 fas3 c java administration piazza github auportal trello

Unread Updated Unresolved Following

New Post Search or add a post...

Filtering by: Unanswered Questions [Add shortcut](#)

TODAY

Private Malloc & free 1:09PM

Hej! Vi använder malloc ganska ofta för vår string input i lagerhanteraren. Hittills har vi inte använt free alls och in

LAST WEEK

6. Clementinen Ny i gruppen Wed

Tjena! Jag har fått den äran att vara med i er grupp och behöver en kodkompis. Någon som vill jobba ihop? Antar att vi s

WEEK 8/30 - 9/5

7. Halloet Selfie 9/3/15

Ses vi i foo bar?

8. Kwin Selfie 8/31/15

Hej alla Kwin! Jag tycker vi alla ska träffas onsdag cirka 14:30 i Skrubben för att ta selfie:n. Denna tid bör passa

Class at a Glance Updated 42 seconds ago. [Reload](#)

no unread posts

3 unanswered questions

2 unresolved followups

60 total posts

275 total contributions

33 instructors' responses

18 students' responses

26 min avg. response time

Student Enrollment ..out of 120 (estimated) [Edit](#)

135 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



Nytt i AU-Portalen

X mål i Y redovisningar före dig i kön
(*Refresha manuellt för uppdatering tills vidare...*)



Tips för rotation

- Se till att du har en partner **idag** så ni kan sätta igång i morgon/på måndag
- Välj gärna en partner som du ligger i fas med
- Ni har bådas kod att utgå från

T.ex. kan ni välja att gå vidare med vilken lagerhanterare ni vill

Lagerhanterare 2.0

- Inte så många nya funktioner — vi skall ändra på hur saker fungerar under huven

Ny funktion: packa en pall

- Kodens kvalitet

Inga globala variabler, inga magiska konstanter

Ingen onödig upprepning av kod

- Bättre datastrukturer — träd och listor

Vi har redan tittat lite på dessa i tidigare föreläsningar

- Dela upp programmet i lämpliga moduler

Vi har redan gått igenom modularisering

- Dokumentation och vettig namngivning

Helt OK program Inlupp 1

”Magiskt nummer”

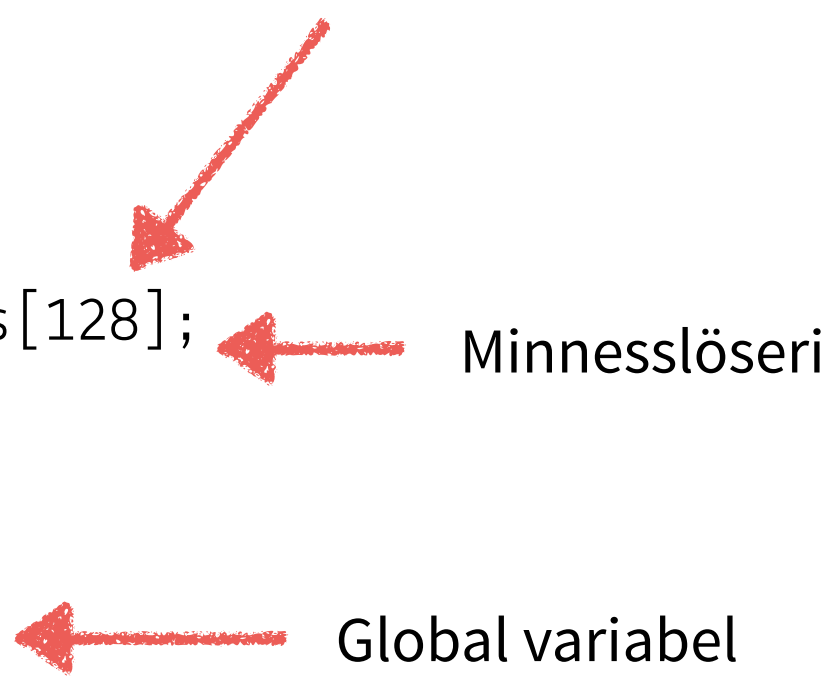
```
struct db
{
    good_t goods[128];
    int size;
};
```

Minnesslöseri

```
struct db DB;
```

Global variabel

```
int main(void)
{
    ...
    add_to_db();
}
```



Rimliga förändringar inför Inlupp 2

Definition inte siffra

```
#define INITIAL_DB_CAPACITY 128
```

```
struct db  
{
```

```
    good_t *goods;
```

```
    int capacity;
```

```
    int size;
```

```
};
```

Pekare till en array på heapen
(kommer att vara ett träd!)

Ingen hårdkodad kapacitet

Lokal variabel

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

```
    struct db DB = { .capacity = INITIAL_DB_CAPACITY; }
```

```
    add_to_db(&db);
```

```
}
```

Skickas som parameter (jmf. PKD)



Helt OK program Inlupp 1

lager.c

```
struct list {
    struct link *first, *last;
};

struct link {
    int value;
    struct link *next;
};

typedef struct list list_t;

int length(list_t*);
int empty(list_t*);

struct link* mk_link(...);

void append(...) {
    ...
}

int length(...) {
    ...
}

int empty(...) {
    ...
}
```



Rimliga förändringar inför Inlupp 2

lager.h

```
#ifdef
#define

typedefs...

functionsprototyper...

#endif
```

lager.c

```
#include "lager.h"

strukturedefinitioner...

funktionsprototyper...

implementation av koden
```

...plus flera andra



Kraven växer med din kunskap!

- Första inluppen: vi fokuserar på att programmet gör **vad** det skall

Komma igång med C (se Z100)

- Andra inluppen: vi fokuserar på **hur** programmet gör det den skall

Minneshantering, globala variabler, datastrukturer, moduluppdelning, namngivning, etc. (se Z101)

- Tredje inluppen: vi fokuserar på programmet i dess helhet

Testning, makefiler, kodgranskning, etc. (se Z102)

Föreläsning 7

Tobias Wrigstad

*Pekare och arrayer. Dynamiska arrayer.
Pekararrayer och kommandorads-
argument.*



Pekare och arrayer (är nästan samma sak)

- Vad är skillnaden mellan dessa?

```
char *s = "Hello";
```

```
char s[] = { 'H', 'e', 'l', 'l', 'o', '\0' };
```

```
char s[] = "Hello";
```

- De sista två är exakt samma, strängarna hamnar på stacken. Den första lägger strängen i "programmet" (ROM), eftersom den pekar på "Hello" som ligger i programmet.

```
s[0] == 'H'
```

```
s[5] == '\n'
```

```
s[6] == vad?
```

Pekare och arrayer (är nästan samma sak)

- Vad är skillnaden mellan dessa?

```
char *s = "Hello";
```

```
char s[] = { 'H', 'e', 'l', 'l', 'o', '\0' };
```

```
char s[] = "Hello";
```

```
char *s = strdup("Hello");
```

```
char s[] = strdup("Hello"); // kompilerar ej
```

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

```
int main(int argc, char *argv[])  
{  
    char *s1 = strdup("Hello"); // 5  
    char s2[] = strdup("Hello"); // 6  
    return 0;  
}
```

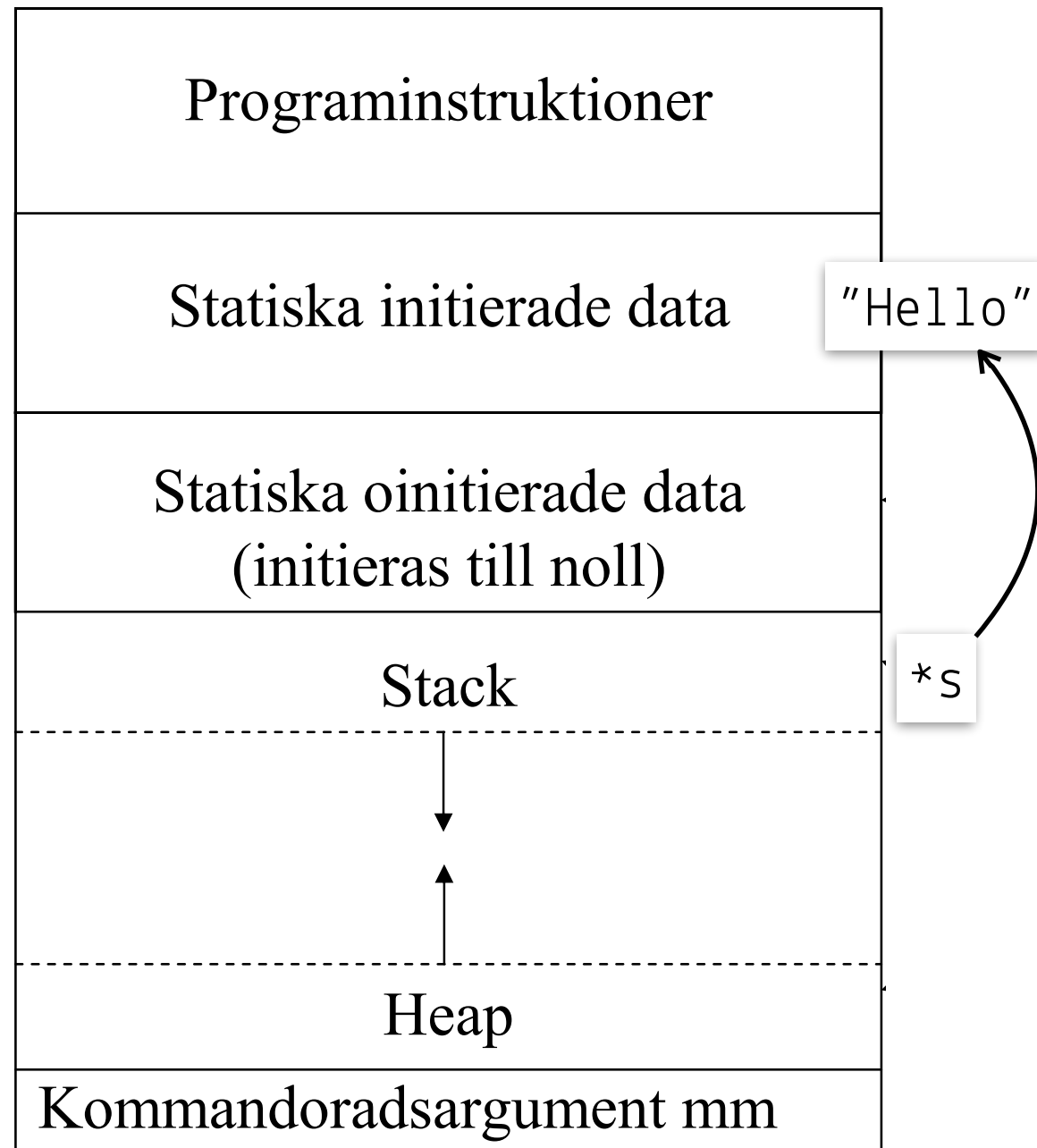
```
$ gcc temp.c
```

```
foo.c:6:8: error: array initializer must be an  
initializer list or string literal
```

```
    char s2[] = strdup("Hello");  
                ^
```



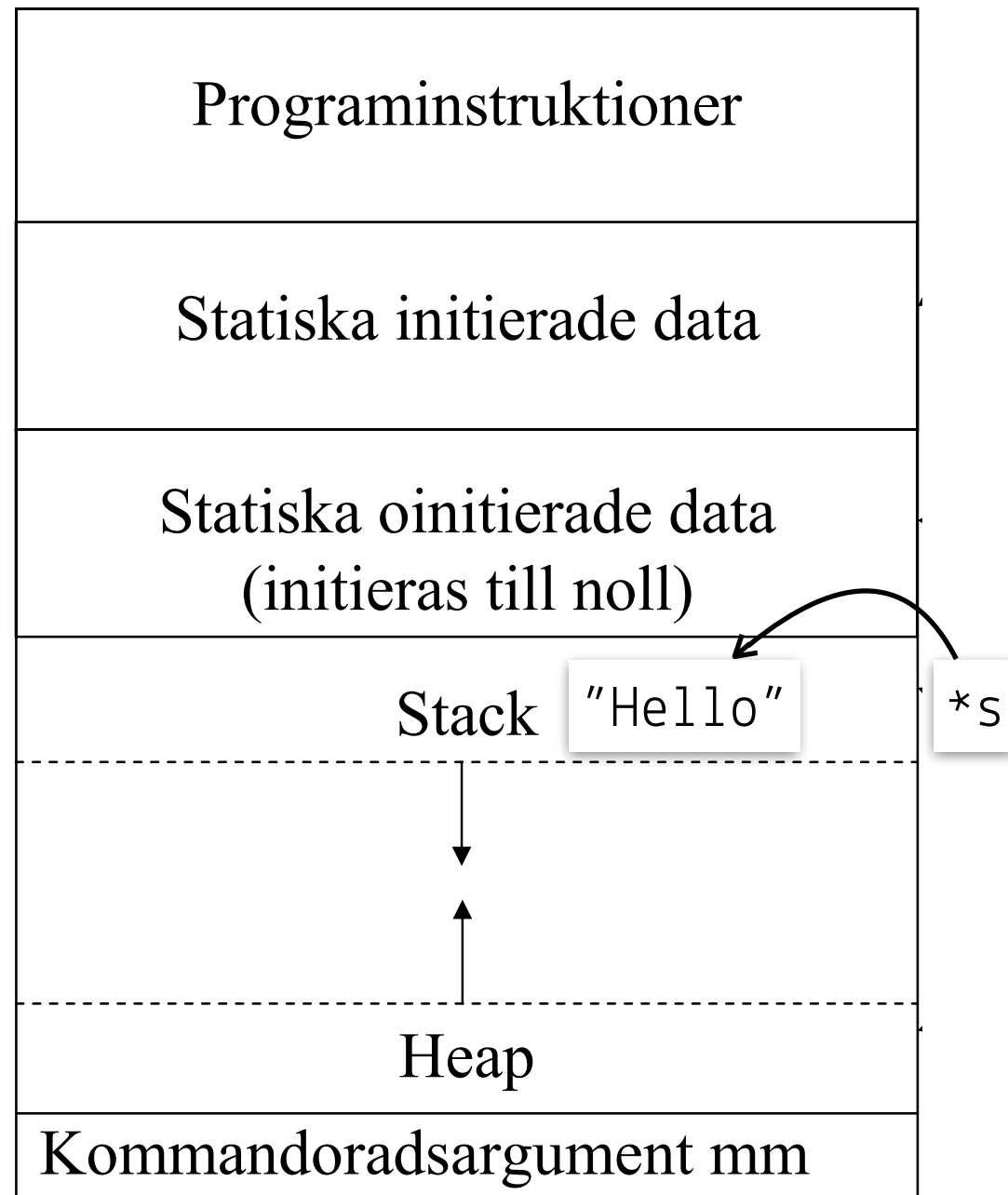
```
char *s = "Hello";
```



```
s[4] = 'x'; // BOOM!
```

```
s[6] = ...
```

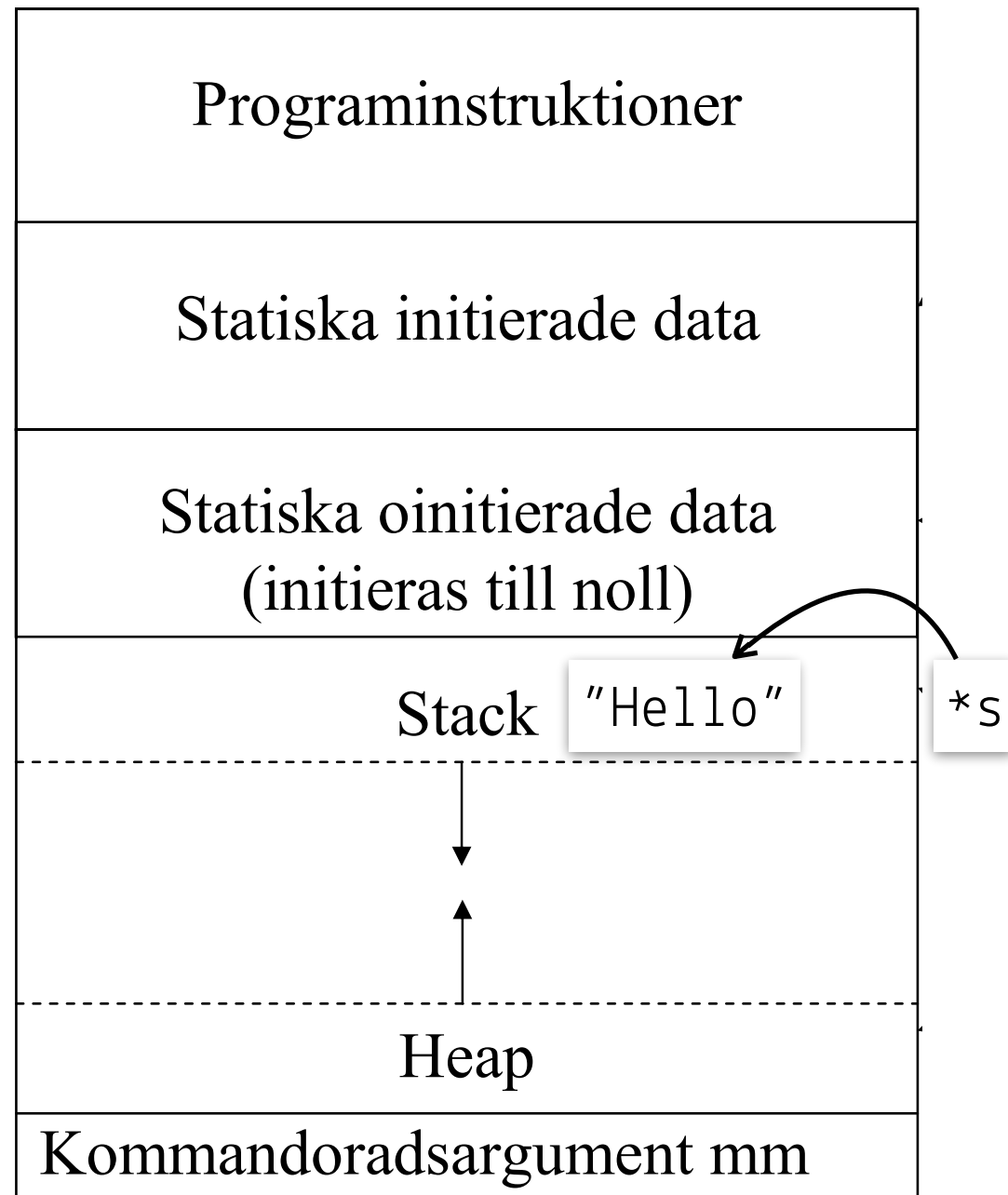
```
char s[] = "Hello";
```



```
s[4] = 'x'; // OK!
```

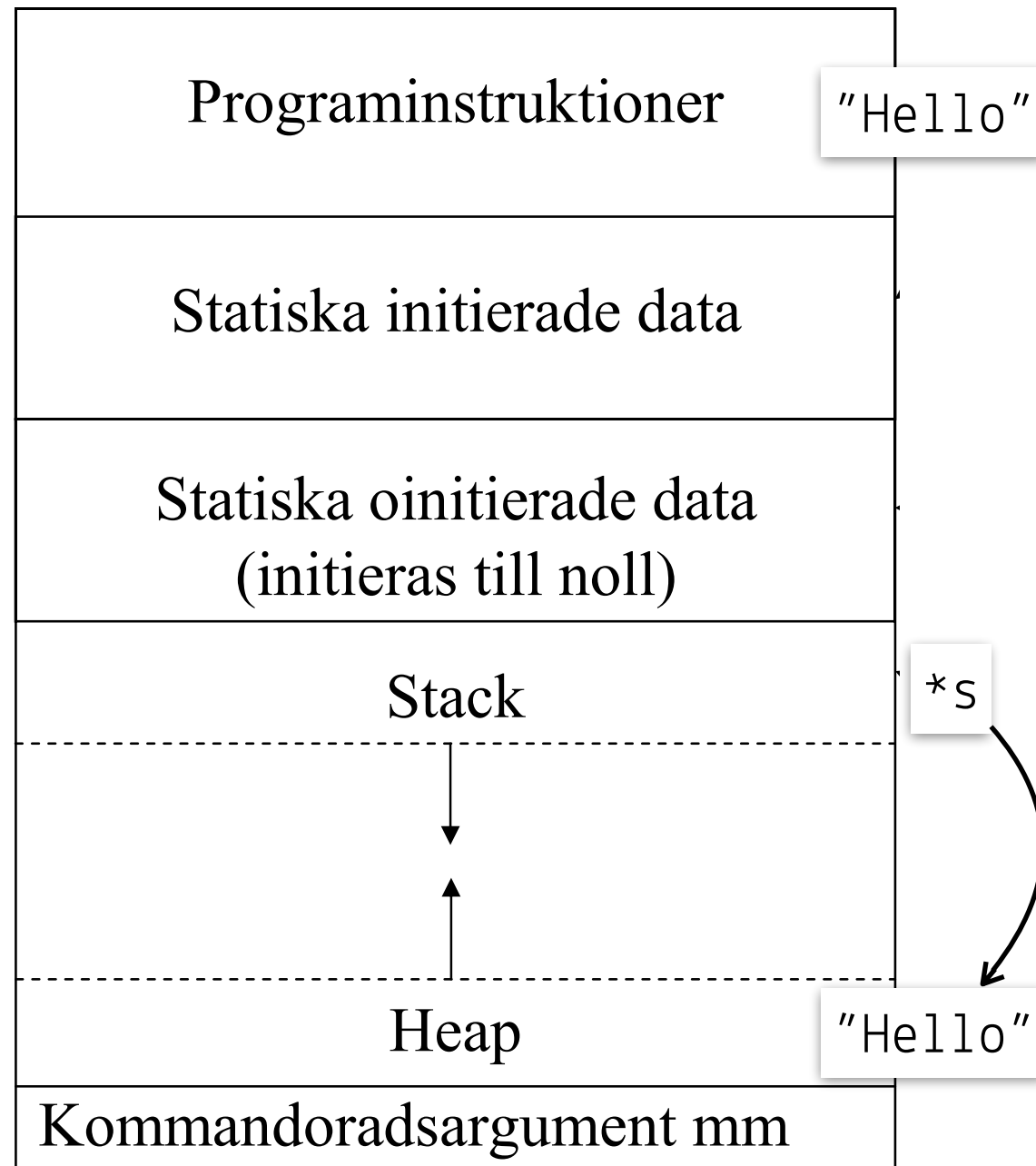


```
char s[] = { 'H', 'e', 'l', 'l', 'o', '\0' };
```



```
s[4] = 'x'; // OK!
```

```
char *s = strdup("Hello");
```



(argumentet till `strdup`)

```
s[4] = 'x'; // OK!
```

Vanlig felkälla i lagerhanteraren

- Läsa in en sträng:

```
char buf[256];  
scanf("%s", buf);  
return buf;
```

- Vad är felet?
- Hur skiljer sig detta (också felaktiga) program?

```
char *buf;  
scanf("%s", buf);  
return buf;
```

Två lösningar

- Läs in i buffert, kopiera strängen på heapen, returnera pekare till kopian

```
char buf[256];  
scanf("%s", buf);  
return strdup(buf);
```

- OBS! Kräver att strängen frigörs med `free` på annan plats i programmet!
- Ännu bättre lösning (varför):

```
char *buf = NULL;  
size_t buf_len = 0;  
getline(&buf, &buf_len, stdin);  
return buf;
```

- *(Man kan också skicka med en buffert från anropsplatsen.)*

STACK

HEAP

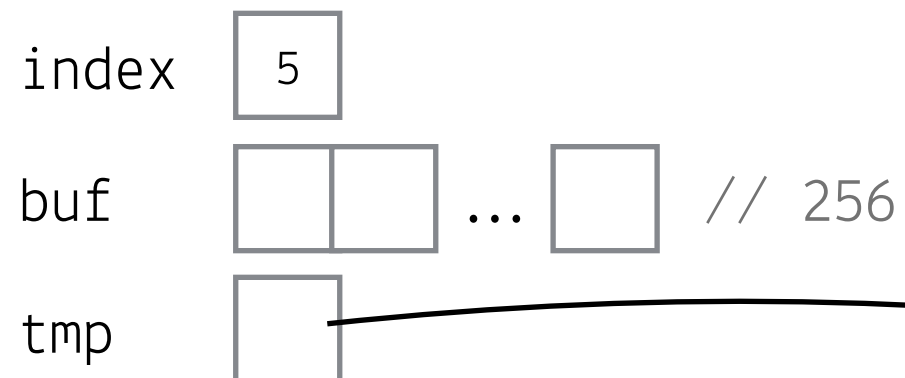
STATISKA INITIERADE VARIABLER

PROGRAMINSTRUKTIONER

```
void set_name(int index)
{
    char buf[256];
    scanf("%s", buf);
    char *tmp = strdup(buf);
    DB.goods[index].name = tmp;
}
```



STACK



HEAP



STATISKA INITIERADE VARIABLER

DB



PROGRAMINSTRUKTIONER

```
void set_name(int index)
{
    char buf[256];
    scanf("%s", buf);
    char *tmp = strdup(buf);
    DB.goods[index].name = tmp;
}
```



Använd alltid funktioner ”som terminerar”!

- Många standardfunktioner har en version som också tar ett gränsvärde:

`strncpy` — jämför de första n tecknen i två strängar (terminerar efter n steg)

`strncpy` — kopiera n tecken från a till b (terminerar efter n steg)

`getline` — allokerar själv en buffert som rymmer indata

...

- Försök från och med nu att undvika kod som ser ut så här:

```
char buf[256];
```

```
scanf("%s", buf); // kraschar om input är större än 256
```

```
return strdup(buf);
```

- Observera att lösningen **inte** är ”en större buffert”.

Dynamiska arrayer

- Exempel

Hur kan man implementera en array i C som kan växa och krympa?

- Exemplifierar

Manuell minneshantering

Värdesemantik vs. pekarsemantik

```
#include <string.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdbool.h>

typedef int T;

struct dyn_array
{
    uint16_t capacity; // 64K element max
    uint16_t used;
    T *elements;
};

typedef struct dyn_array dyn_array_t;
```


Interface

`darray_create` — skapa en ny dynamisk array med en given kapacitet

Allokera minne!

`darray_free` — frigör en dynamisk array

Avallokera minne!

`darray_set` — uppdatera ett givet element med indexkontroll

`darray_get` — skaffa en pekare till ett givet element med indexkontroll

`darray_append` — öka storleken på arrayen och lägg till ett nytt element sist

Ändra på minnesstorlek!

`darray_prepend` — öka storleken på arrayen och lägg till ett nytt element först

Ändra på minnesstorlek!

```
// i main  
a = darray_create(4);
```

darray_create

capacity 4

capacity	4
elements	

main

a

capacity	4
elements	

Stack

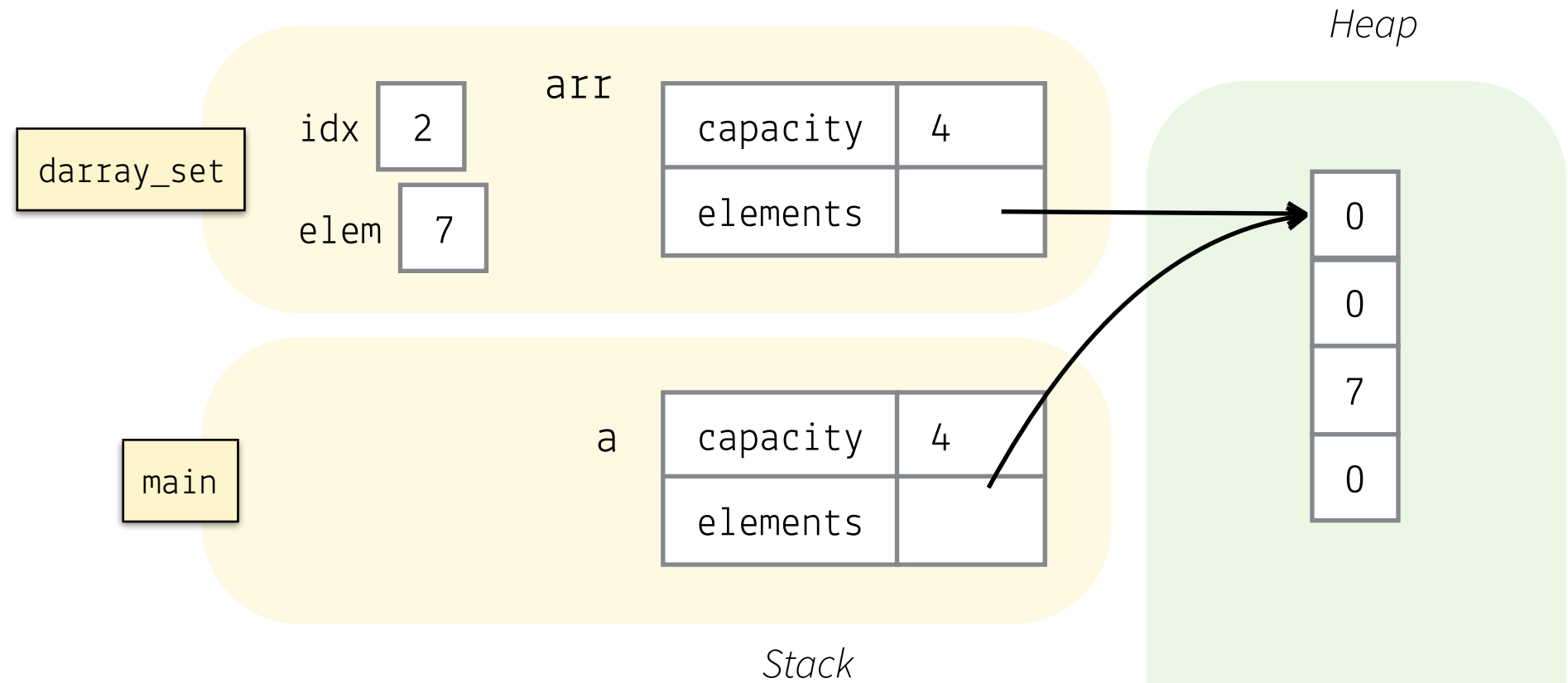
Heap

0
0
0
0

```
dyn_array_t darray_create(uint16_t capacity)  
{  
    // OBS! Borde göra felkontroll!  
    return (dyn_array_t) {  
        .capacity = capacity,  
        .elements = calloc(capacity, sizeof(T)) };  
}  
  
void darray_free(dyn_array_t *arr)  
{  
    free(arr->elements);  
    free(arr);  
}
```



darray_set(a, 2, 7);

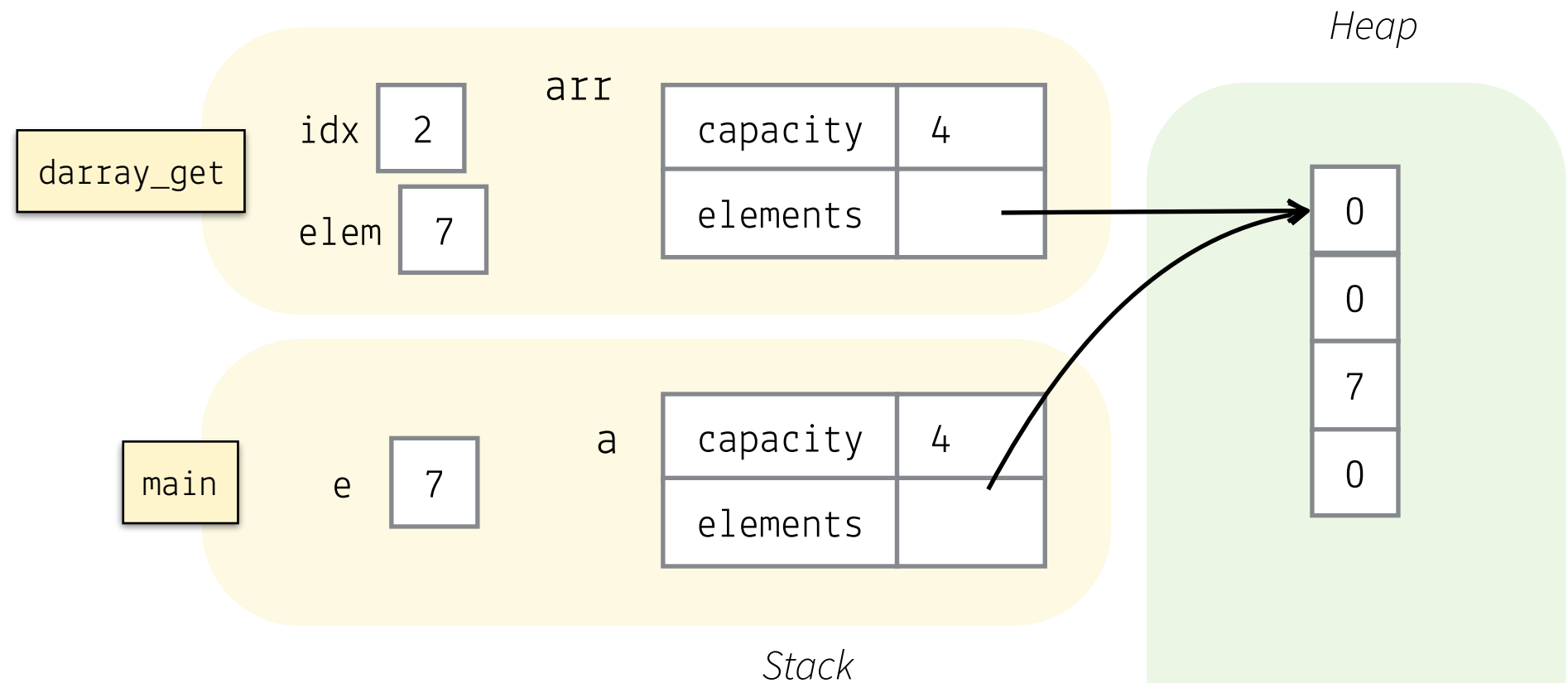


```
bool darray_set(dyn_array_t arr, uint16_t idx, T elem)
{
    if (idx < arr.capacity)
    {
        arr.elements[idx] = elem;
        return true;
    }

    return false;
}
```



`e = *darray_get(a, 2);`



```
T *darray_get(dyn_array_t arr, uint16_t idx)
{
    return (idx < arr.capacity) ? &arr.elements[idx] : NULL;
}
```

```
bool darray_get(dyn_array_t arr, uint16_t idx, T *result)
{
    ... // övning!
}
```



Heap

darray_append(&a, 9);

darray_append

elem

9

arr

main

a

capacity

8

elements

Stack

```
void darray_append(dyn_array_t *arr, T elem)
{
    int idx = arr->capacity;

    arr->capacity *= 2;
    arr->elements =
        realloc(arr->elements, arr->capacity * sizeof(T));

    arr->elements[arr->idx] = elem;
}
```

0

0

7

0

9

0

0

0



darray_prepend(&a, 1);

darray_prepend

elem

9

arr

main

a

capacity

8

elements

Stack

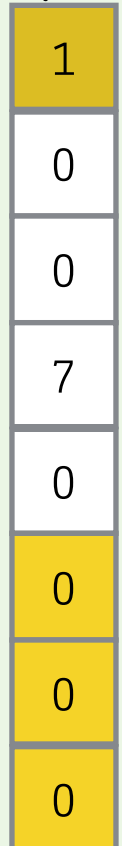
Heap

```
void darray_prepend(dyn_array_t *arr, T elem)
{
    int max = arr->capacity;

    arr->capacity *= 2;
    arr->elements =
        realloc(arr->elements, arr->capacity * sizeof(T));

    for (int i = max; i > 0; --i)
        arr->elements[i] = arr->elements[i-1];

    arr->elements[0] = elem;
}
```



realloc och calloc

- `ptr = realloc(ptr, new_size)`

Ändrar storleken på ett minnesutrymme, möjligen genom att flytta det

Farligt om det finns alias till `ptr`

- `ptr = calloc(number, size)`

Allokerar `number * size` antal bytes

Nollställer minnet

Frivillig övningsuppgift hemma

- Varför används pekarsemantik ibland och värdesemantik ibland?

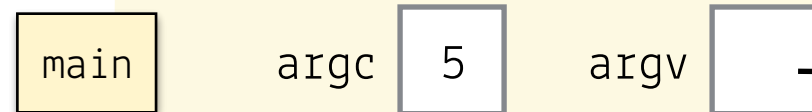
Vad skulle hända om man bytte från pekarsemantik till värdesemantik eller tvärtom i t.ex. `darray_prepend`?

- Hur fungerar `malloc`, `free`, `calloc` och `realloc`?

Läs gärna man-sidorna (`$ man calloc`) så du har koll på man till kodprovet!

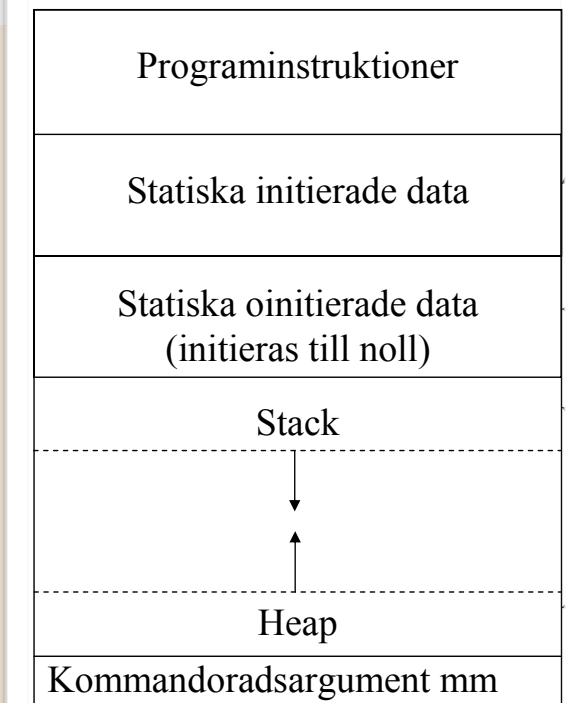
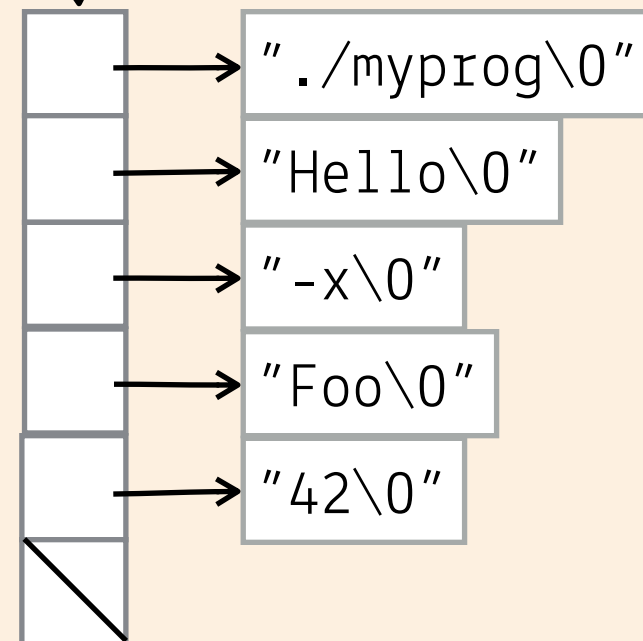
- Om man ändrade på typen `T` till att vara en pekare — vad skulle hända då med biblioteket?

Pekararrayer och kommandoradsargument



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

```
$ ./myprog Hello -x Foo 42
```



Läsbarhet?

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

```
int main(int argc, char *argv[])
{
    for (int i = 0; i < argc; ++i)
    {
        puts(argv[i]);
    }

    return 0;
}
```

Förstör inte heller argv!

Genericitet

- Vår dynamiska array tog emot en pekare av typen `T` som var definierad som en `int`

Återanvändning — man kan ändra `T` till något annat och kompilera om

Återanvändning flera gånger i samma program?

Två möjligheter: skapa ett makro som skapar flera datastrukturer — eller `void *`

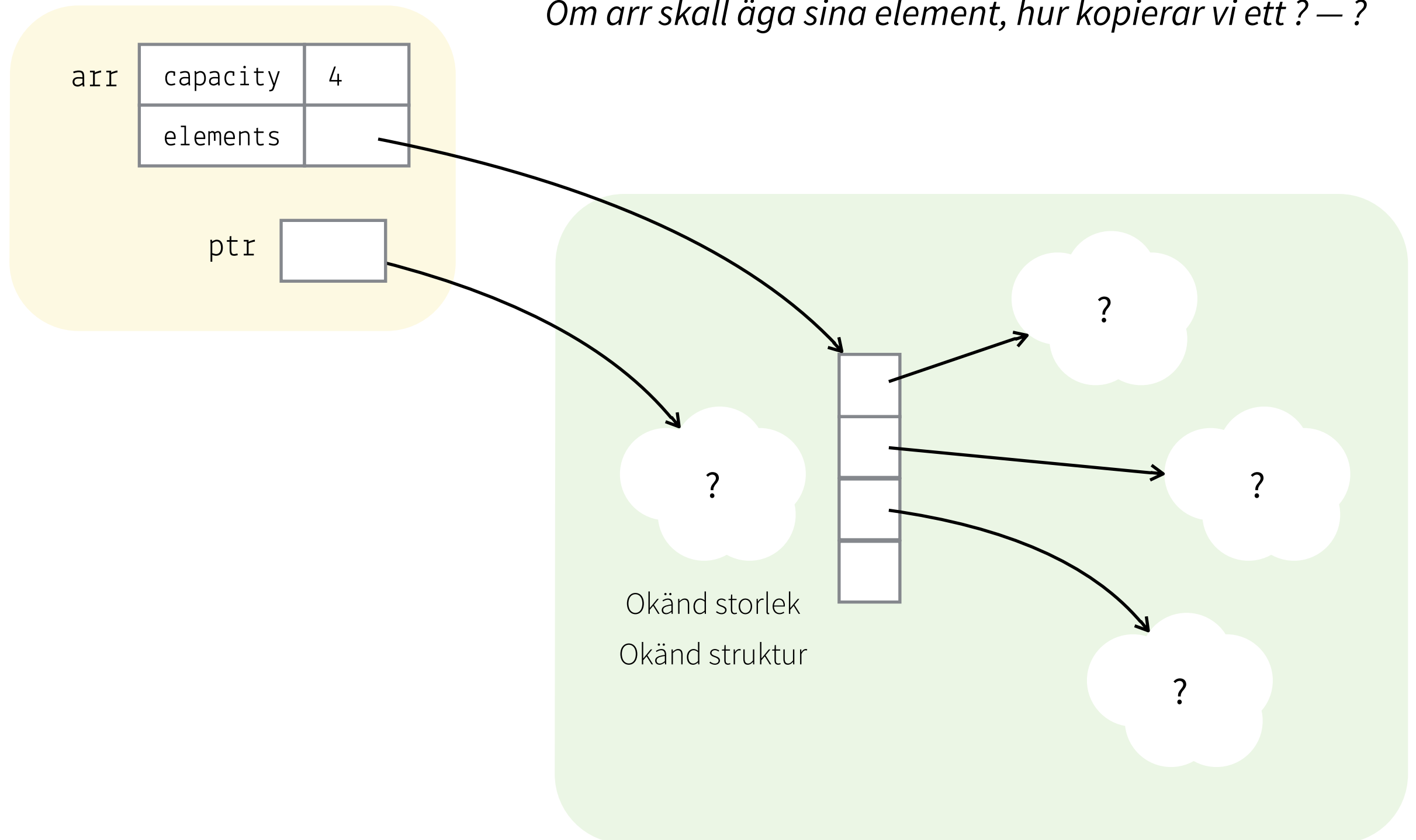
- Använda **`typedef void *T;`**

Eventuellt problem: kan inte längre beräkna `sizeof(*T)` (— varför inte?)

- Scenario: vi vill att den dynamiska arrayen skall äga sitt minne

darray_set(arr, 3, ptr)

Om arr skall äga sina element, hur kopierar vi ett? — ?



...men vad händer om T innehåller pekare?

```
void darray_free(dyn_array_t *arr)
{
    for (int i = 0; i < arr->capacity; ++i)
    {
        free(arr->elements[i]); // kan läcka minne!
    }
    free(arr->elements);
    free(arr);
}
```

Vi skall se en lösning på detta på föreläsning 10!