F06 - Pekare och fält

Programmeringsteknik med C och Matlab, 7,5 hp

Niclas Börlin
niclas.borlin@cs.umu.se

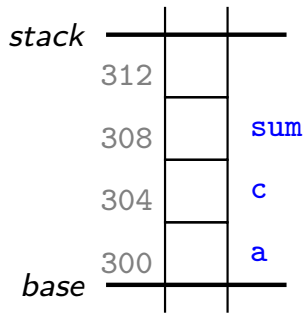
Datavetenskap, Umeå universitet

2023-10-06 Fre

Adresser och pekare

Adress-operatorn

- Alla variabler ligger nånstans i minnet
- Alla har en adress
- Adress-operatorn `&` returnerar *adressen* till variabeln
  - *ej* värdet



code/vars-on-stack.c

```
12  int main(void)
13  {
14      int a = 2;
15      int c = 3;
16      int sum;
```

- `printf("The address of c=%p\n", &c);`
  - `The address of c=304`

Pekare (1)

- En pekare eller pekarvariabel är en variabel som innehåller en adress till någonting
- Internt lagras den som ett heltal
  - 32 eller 64 bitar (4 eller 8 bytes) beroende på system
  - Detta dokument använder 4 bytes
- En pekare deklareras med en stjärna (`*`) efter typen
- Till exempel:
  - ```
    int *p;
    char *r;
    ```
  - Här är variabeln `p` är av typen "pekare till `int`" (*"`int` pointer"* eller *"pointer to `int`"*)
  - Variabeln `r` är av typen "pekare till `char`" (*"`char` pointer"* eller *"pointer to `char`"*)
    ```

Pekare (2)

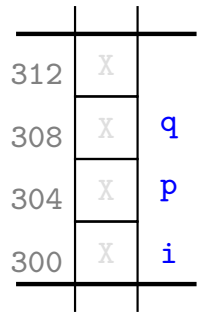
- Inget hindrar att vi har en pekare till en pekare
  - `int **q;`
  - Här är variabeln `q` av typen "pekare till pekare till `int`" eller "dubbelpekare till `int`" (*"`int` double pointer"*)
- Notera att om flera variabler deklareras i samma sats så är stjärnan kopplad till variabeln, inte till typen
  - Exempel:
    - `int i, *p, **q;`

    deklarerar
    - en variabel `i` av typen `int`
    - en variabel `p` av typen `int *` (enkelpekare)
    - en variabel `q` av typen `int **` (dubbelpekare)

Pekare och adress-operatorn

- En pekare till typen X kan tilldelas adressen för en variabel av typen X

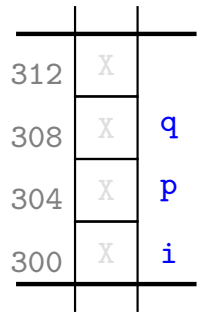
```
int i, *p, **q;
p = &i;
q = &p;
```



Pekare och adress-operatorn

- En pekare till typen X kan tilldelas adressen för en variabel av typen X

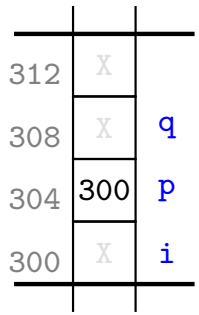
```
int i, *p, **q;
p = &i;
q = &p;
```



Pekare och adress-operatorn

- En pekare till typen X kan tilldelas adressen för en variabel av typen X

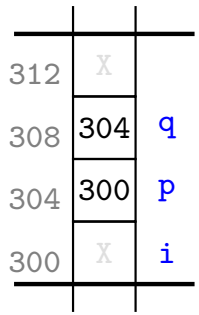
```
int i, *p, **q;
p = &i;
q = &p;
```



Pekare och adress-operatorn

- En pekare till typen X kan tilldelas adressen för en variabel av typen X

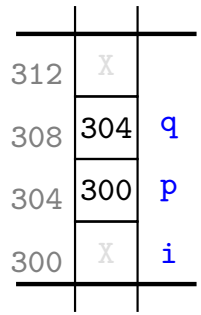
```
int i, *p, **q;
p = &i;
q = &p;
```



Pekare och adress-operatorn

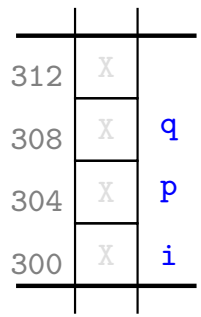
- En pekare till typen X kan tilldelas adressen för en variabel av typen X

```
int i, *p, **q;
p = &i;
q = &p;
```



- Ofta illustrerar man pekare med hjälp av en pil

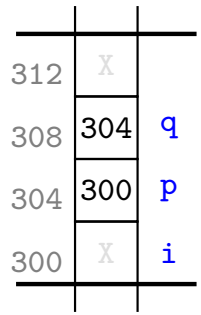
```
int i, *p, **q;
p = &i;
q = &p;
```



Pekare och adress-operatorn

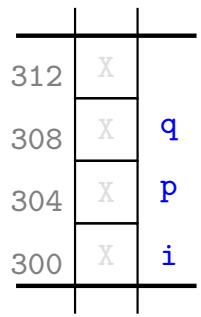
- En pekare till typen X kan tilldelas adressen för en variabel av typen X

```
int i, *p, **q;
p = &i;
q = &p;
```



- Ofta illustrerar man pekare med hjälp av en pil

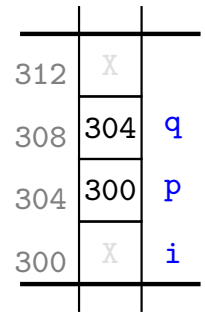
```
int i, *p, **q;
p = &i;
q = &p;
```



Pekare och adress-operatorn

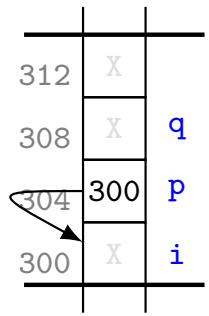
- En pekare till typen X kan tilldelas adressen för en variabel av typen X

```
int i, *p, **q;
p = &i;
q = &p;
```



- Ofta illustrerar man pekare med hjälp av en pil

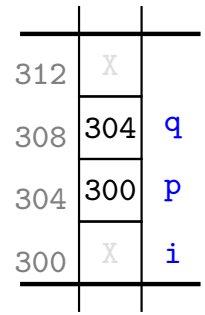
```
int i, *p, **q;
p = &i;
q = &p;
```



Pekare och adress-operatorn

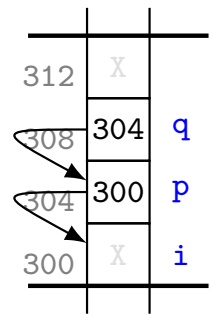
- En pekare till typen X kan tilldelas adressen för en variabel av typen X

```
int i, *p, **q;
p = &i;
q = &p;
```



- Ofta illustrerar man pekare med hjälp av en pil

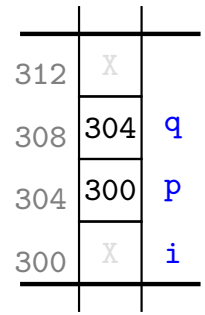
```
int i, *p, **q;
p = &i;
q = &p;
```



Pekare och adress-operatorn

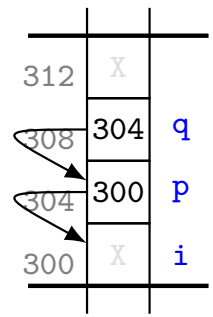
- En pekare till typen X kan tilldelas adressen för en variabel av typen X

```
int i, *p, **q;
p = &i;
q = &p;
```



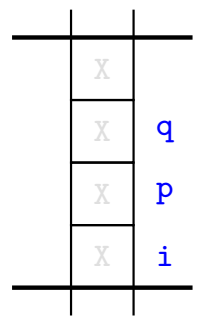
- Ofta illustrerar man pekare med hjälp av en pil

```
int i, *p, **q;
p = &i;
q = &p;
```



- Oftast utelämnar man adresserna helt

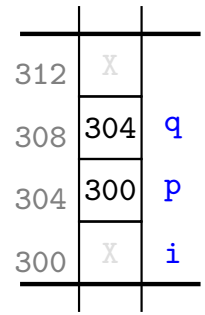
```
int i, *p, **q;
p = &i;
q = &p;
```



Pekare och adress-operatorn

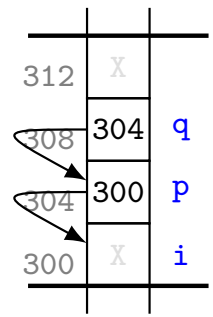
- En pekare till typen X kan tilldelas adressen för en variabel av typen X

```
int i, *p, **q;
p = &i;
q = &p;
```



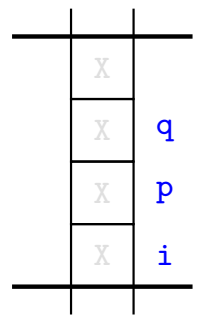
- Ofta illustrerar man pekare med hjälp av en pil

```
int i, *p, **q;
p = &i;
q = &p;
```



- Oftast utelämnar man adresserna helt

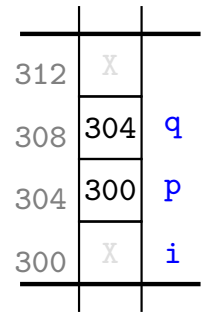
```
int i, *p, **q;
p = &i;
q = &p;
```



Pekare och adress-operatorn

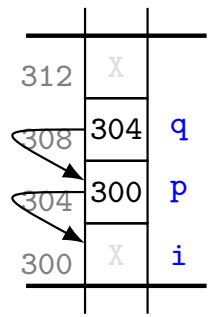
- En pekare till typen X kan tilldelas adressen för en variabel av typen X

```
int i, *p, **q;
p = &i;
q = &p;
```



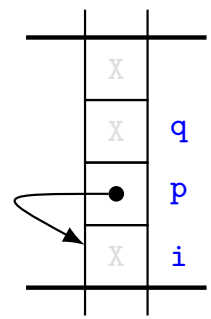
- Ofta illustrerar man pekare med hjälp av en pil

```
int i, *p, **q;
p = &i;
q = &p;
```



- Oftast utelämnar man adresserna helt

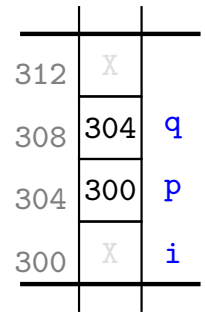
```
int i, *p, **q;
p = &i;
q = &p;
```



Pekare och adress-operatorn

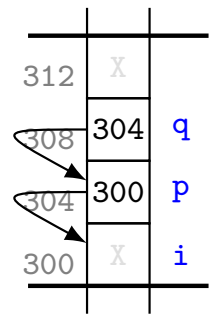
- En pekare till typen X kan tilldelas adressen för en variabel av typen X

```
int i, *p, **q;
p = &i;
q = &p;
```



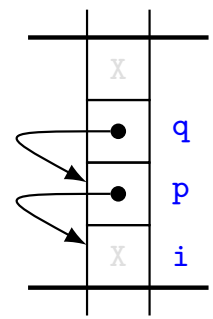
- Ofta illustrerar man pekare med hjälp av en pil

```
int i, *p, **q;
p = &i;
q = &p;
```



- Oftast utelämnar man adresserna helt

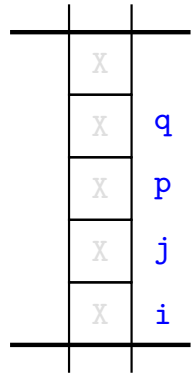
```
int i, *p, **q;
p = &i;
q = &p;
```



Dereferering

- Studera koden

```
int i, j, *p, *q;
p = &i;
q = &i;
*p = 4; // Same effect as i=4
j = *q; // Same effect as j=i
```

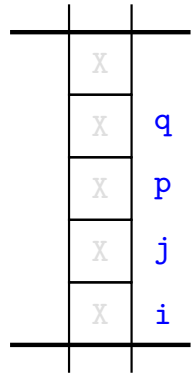


- Om `p` pekar på (refererar till) variabeln `i` så kan vi dereferera `p` för att komma åt värdet i `i`
  - Det kallas ibland att vi följer pekaren
- Det görs genom att skriva en stjärna framför variabelnamnet
  - Om `p` är av typen `int *` så är uttrycket `*p` av typen `int`
- Att `p` och `q` pekar på samma variabel kallas för aliasing
  - Aliasing riskerar att göra koden svårläslig

Dereferering

- Studera koden

```
int i, j, *p, *q;
p = &i;
q = &i;
*p = 4; // Same effect as i=4
j = *q; // Same effect as j=i
```

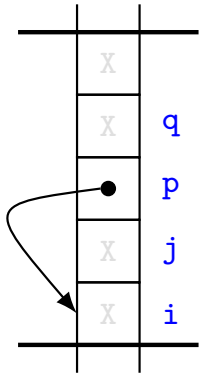


- Om `p` pekar på (refererar till) variabeln `i` så kan vi dereferera `p` för att komma åt värdet i `i`
  - Det kallas ibland att vi följer pekaren
- Det görs genom att skriva en stjärna framför variabelnamnet
  - Om `p` är av typen `int *` så är uttrycket `*p` av typen `int`
- Att `p` och `q` pekar på samma variabel kallas för aliasing
  - Aliasing riskerar att göra koden svårläslig

Dereferering

- Studera koden

```
int i, j, *p, *q;
p = &i;
q = &i;
*p = 4; // Same effect as i=4
j = *q; // Same effect as j=i
```

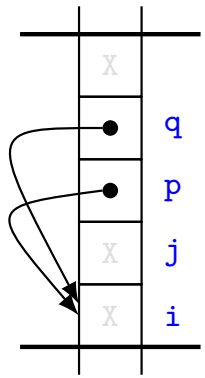


- Om `p` pekar på (refererar till) variabeln `i` så kan vi dereferera `p` för att komma åt värdet i `i`
  - Det kallas ibland att vi följer pekaren
- Det görs genom att skriva en stjärna framför variabelnamnet
  - Om `p` är av typen `int *` så är uttrycket `*p` av typen `int`
- Att `p` och `q` pekar på samma variabel kallas för aliasing
  - Aliasing riskerar att göra koden svårläslig

Dereferering

- Studera koden

```
int i, j, *p, *q;
p = &i;
q = &i;
*p = 4; // Same effect as i=4
j = *q; // Same effect as j=i
```

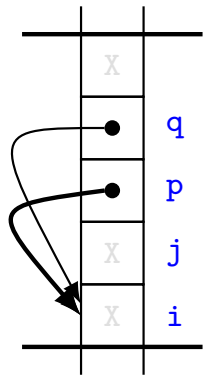


- Om `p` pekar på (refererar till) variabeln `i` så kan vi dereferera `p` för att komma åt värdet i `i`
  - Det kallas ibland att vi följer pekaren
- Det görs genom att skriva en stjärna framför variabelnamnet
  - Om `p` är av typen `int *` så är uttrycket `*p` av typen `int`
- Att `p` och `q` pekar på samma variabel kallas för aliasing
  - Aliasing riskerar att göra koden svårläslig

Dereferering

- Studera koden

```
int i, j, *p, *q;
p = &i;
q = &i;
*p = 4; // Same effect as i=4
j = *q; // Same effect as j=i
```

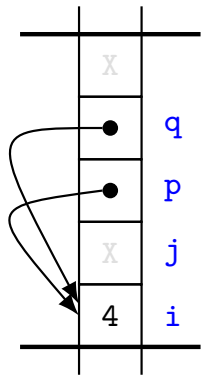


- Om `p` pekar på (refererar till) variabeln `i` så kan vi dereferera `p` för att komma åt värdet i `i`
  - Det kallas ibland att vi följer pekaren
- Det görs genom att skriva en stjärna framför variabelnamnet
  - Om `p` är av typen `int *` så är uttrycket `*p` av typen `int`
- Att `p` och `q` pekar på samma variabel kallas för aliasing
  - Aliasing riskerar att göra koden svårläslig

Dereferering

- Studera koden

```
int i, j, *p, *q;
p = &i;
q = &i;
*p = 4; // Same effect as i=4
j = *q; // Same effect as j=i
```

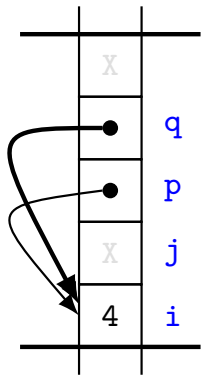


- Om `p` pekar på (refererar till) variabeln `i` så kan vi dereferera `p` för att komma åt värdet i `i`
  - Det kallas ibland att vi följer pekaren
- Det görs genom att skriva en stjärna framför variabelnamnet
  - Om `p` är av typen `int *` så är uttrycket `*p` av typen `int`
- Att `p` och `q` pekar på samma variabel kallas för aliasing
  - Aliasing riskerar att göra koden svårläslig

Dereferering

- Studera koden

```
int i, j, *p, *q;
p = &i;
q = &i;
*p = 4; // Same effect as i=4
j = *q; // Same effect as j=i
```

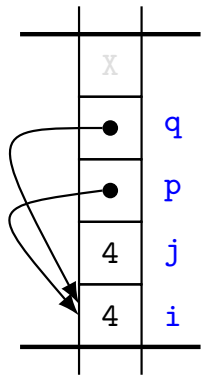


- Om `p` pekar på (refererar till) variabeln `i` så kan vi dereferera `p` för att komma åt värdet i `i`
  - Det kallas ibland att vi följer pekaren
- Det görs genom att skriva en stjärna framför variabelnamnet
  - Om `p` är av typen `int *` så är uttrycket `*p` av typen `int`
- Att `p` och `q` pekar på samma variabel kallas för aliasing
  - Aliasing riskerar att göra koden svårläslig

Dereferering

- Studera koden

```
int i, j, *p, *q;
p = &i;
q = &i;
*p = 4; // Same effect as i=4
j = *q; // Same effect as j=i
```



- Om `p` pekar på (refererar till) variabeln `i` så kan vi dereferera `p` för att komma åt värdet i `i`
  - Det kallas ibland att vi följer pekaren
- Det görs genom att skriva en stjärna framför variabelnamnet
  - Om `p` är av typen `int *` så är uttrycket `*p` av typen `int`
- Att `p` och `q` pekar på samma variabel kallas för aliasing
  - Aliasing riskerar att göra koden svårläslig

Parameteröverföring (1)

- Vad kommer att skrivas ut av den här koden?

code/add-one1.c

```c
#include <stdio.h>
void add_one(int n)
{
    n = n + 1;
}
int main(void)
{
    int a = 5;
    add_one(a);
    printf("a = %d\n", a);
    return 0;
}
```

- . . . eller den här?

code/add-one2.c

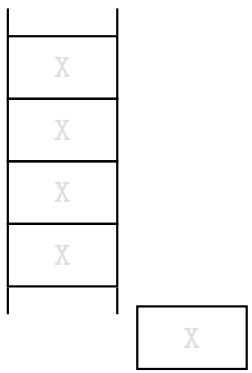
```c
#include <stdio.h>
void add_one(int *n)
{
    *n = *n + 1;
}
int main(void)
{
    int a = 5;
    add_one(&a);
    printf("a = %d\n", a);
    return 0;
}
```

Parameteröverföring (2)

- Vi testkör!

code/add-one1.c

```
1  #include <stdio.h>
2  void add_one(int n)
3  {
4      n = n + 1;
5  }
6  int main(void)
7  {
8      int a = 5;
9      add_one(a);
10     printf("a = %d\n", a);
11     return 0;
12 }
```

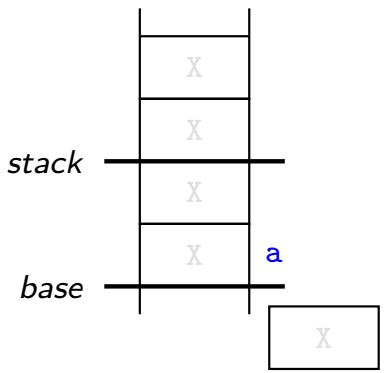


Parameteröverföring (2)

- Vi testkör!

code/add-one1.c

```
1  #include <stdio.h>
2  void add_one(int n)
3  {
4      n = n + 1;
5  }
6  int main(void)
7  {
8      int a = 5;
9      add_one(a);
10     printf("a = %d\n", a);
11     return 0;
12 }
```

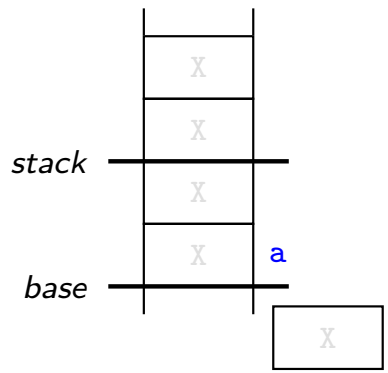


Parameteröverföring (2)

- Vi testkör!

code/add-one1.c

```
1  #include <stdio.h>
2  void add_one(int n)
3  {
4      n = n + 1;
5  }
6  int main(void)
7  {
8      int a = 5;
9      add_one(a);
10     printf("a = %d\n", a);
11     return 0;
12 }
```

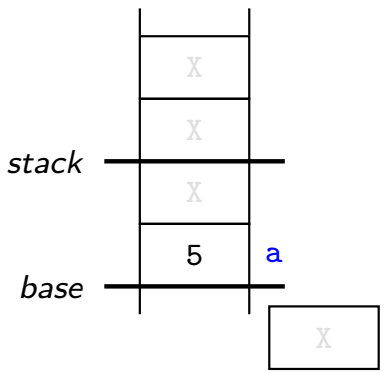


Parameteröverföring (2)

- Vi testkör!

code/add-one1.c

```
1  #include <stdio.h>
2  void add_one(int n)
3  {
4      n = n + 1;
5  }
6  int main(void)
7  {
8      int a = 5;
9      add_one(a);
10     printf("a = %d\n", a);
11     return 0;
12 }
```

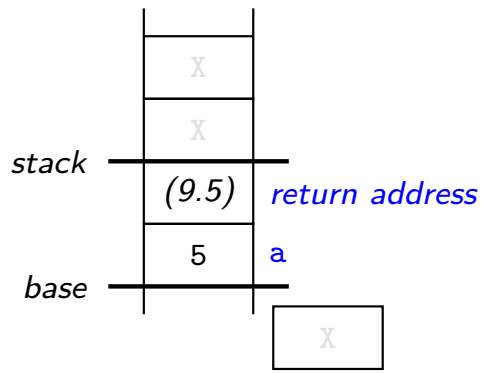


Parameteröverföring (2)

- Vi testkör!

code/add-one1.c

```
1  #include <stdio.h>
2  void add_one(int n)
3  {
4      n = n + 1;
5  }
6  int main(void)
7  {
8      int a = 5;
9      add_one(a);
10     printf("a = %d\n", a);
11     return 0;
12 }
```

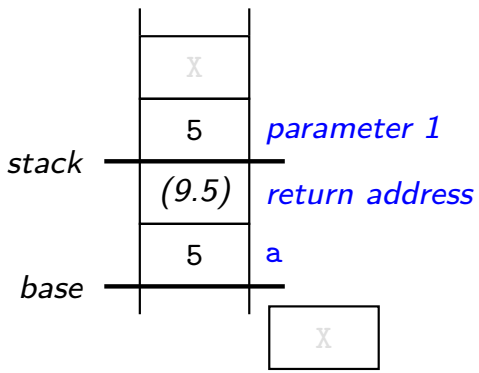


Parameteröverföring (2)

- Vi testkör!

code/add-one1.c

```
1  #include <stdio.h>
2  void add_one(int n)
3  {
4      n = n + 1;
5  }
6  int main(void)
7  {
8      int a = 5;
9      add_one(a);
10     printf("a = %d\n", a);
11     return 0;
12 }
```

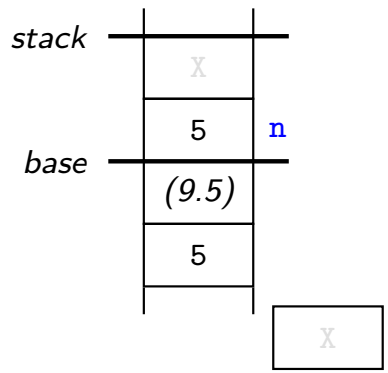


Parameteröverföring (2)

- Vi testkör!

code/add-one1.c

```
1  #include <stdio.h>
2  void add_one(int n)
3  {
4      n = n + 1;
5  }
6  int main(void)
7  {
8      int a = 5;
9      add_one(a);
10     printf("a = %d\n", a);
11     return 0;
12 }
```

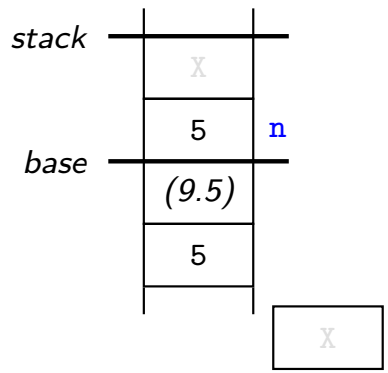


Parameteröverföring (2)

- Vi testkör!

code/add-one1.c

```
1  #include <stdio.h>
2  void add_one(int n)
3  {
4      n = n + 1;
5  }
6  int main(void)
7  {
8      int a = 5;
9      add_one(a);
10     printf("a = %d\n", a);
11     return 0;
12 }
```

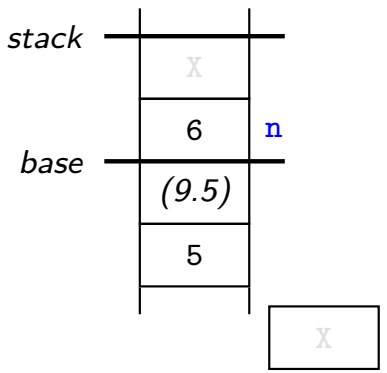


Parameteröverföring (2)

- Vi testkör!

code/add-one1.c

```
1  #include <stdio.h>
2  void add_one(int n)
3  {
4      n = n + 1;
5  }
6  int main(void)
7  {
8      int a = 5;
9      add_one(a);
10     printf("a = %d\n", a);
11     return 0;
12 }
```

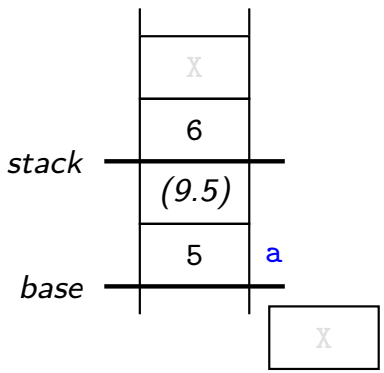


Parameteröverföring (2)

- Vi testkör!

code/add-one1.c

```
1  #include <stdio.h>
2  void add_one(int n)
3  {
4      n = n + 1;
5  }
6  int main(void)
7  {
8      int a = 5;
9      add_one(a);
10     printf("a = %d\n", a);
11     return 0;
12 }
```

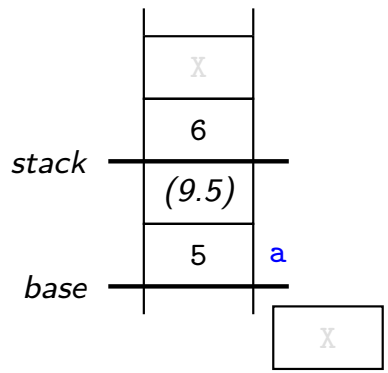


Parameteröverföring (2)

- Vi testkör!

code/add-one1.c

```
1  #include <stdio.h>
2  void add_one(int n)
3  {
4      n = n + 1;
5  }
6  int main(void)
7  {
8      int a = 5;
9      add_one(a);
10     printf("a = %d\n", a);
11     return 0;
12 }
```



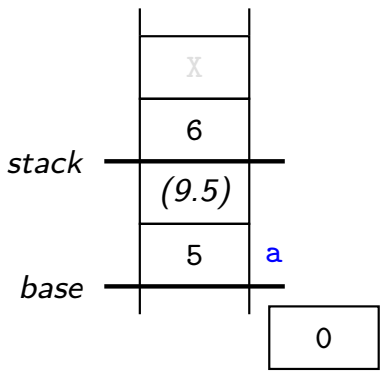

```
a = 5
```

Parameteröverföring (2)

- Vi testkör!

code/add-one1.c

```
1  #include <stdio.h>
2  void add_one(int n)
3  {
4      n = n + 1;
5  }
6  int main(void)
7  {
8      int a = 5;
9      add_one(a);
10     printf("a = %d\n", a);
11     return 0;
12 }
```



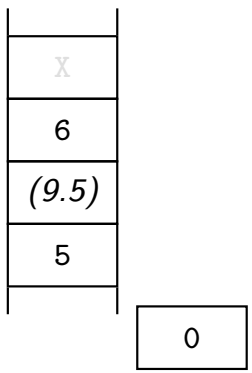

```
a = 5
```

Parameteröverföring (2)

- Vi testkör!

code/add-one1.c

```
1  #include <stdio.h>
2  void add_one(int n)
3  {
4      n = n + 1;
5  }
6  int main(void)
7  {
8      int a = 5;
9      add_one(a);
10     printf("a = %d\n", a);
11     return 0;
12 }
```



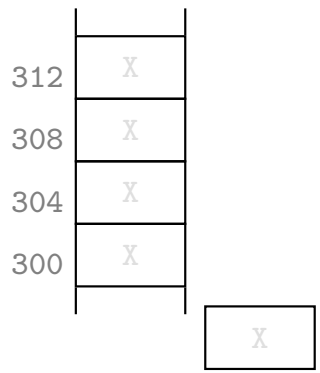

```
a = 5
```

Parameteröverföring (3)

- Vi testkör det andra examplet!

code/add-one2.c

```
1  #include <stdio.h>
2  void add_one(int *n)
3  {
4      *n = *n + 1;
5  }
6  int main(void)
7  {
8      int a = 5;
9      add_one(&a);
10     printf("a = %d\n", a);
11     return 0;
12 }
```

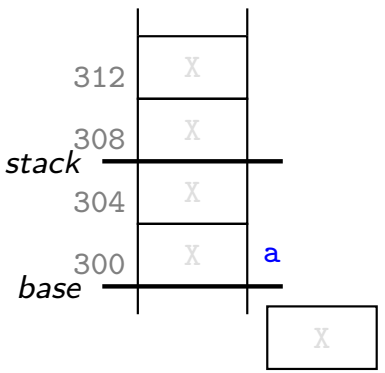


Parameteröverföring (3)

- Vi testkör det andra examplet!

code/add-one2.c

```
1  #include <stdio.h>
2  void add_one(int *n)
3  {
4      *n = *n + 1;
5  }
6  int main(void)
7  {
8      int a = 5;
9      add_one(&a);
10     printf("a = %d\n", a);
11     return 0;
12 }
```

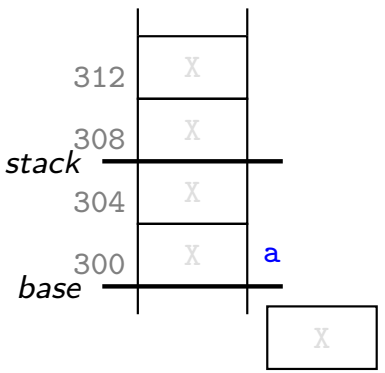


Parameteröverföring (3)

- Vi testkör det andra examplet!

code/add-one2.c

```
1  #include <stdio.h>
2  void add_one(int *n)
3  {
4      *n = *n + 1;
5  }
6  int main(void)
7  {
8      int a = 5;
9      add_one(&a);
10     printf("a = %d\n", a);
11     return 0;
12 }
```

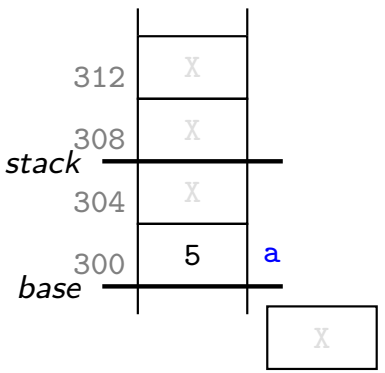


Parameteröverföring (3)

- Vi testkör det andra examplet!

code/add-one2.c

```
1  #include <stdio.h>
2  void add_one(int *n)
3  {
4      *n = *n + 1;
5  }
6  int main(void)
7  {
8      int a = 5;
9      add_one(&a);
10     printf("a = %d\n", a);
11     return 0;
12 }
```

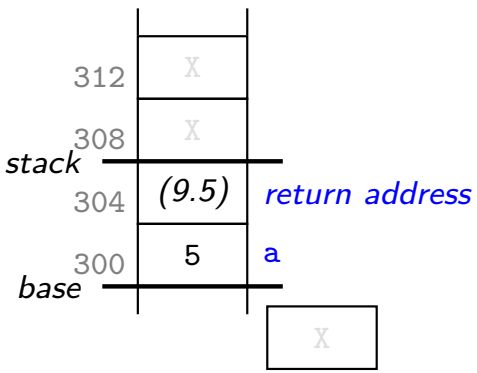


Parameteröverföring (3)

- Vi testkör det andra examplet!

code/add-one2.c

```
1  #include <stdio.h>
2  void add_one(int *n)
3  {
4      *n = *n + 1;
5  }
6  int main(void)
7  {
8      int a = 5;
9      add_one(&a);
10     printf("a = %d\n", a);
11     return 0;
12 }
```

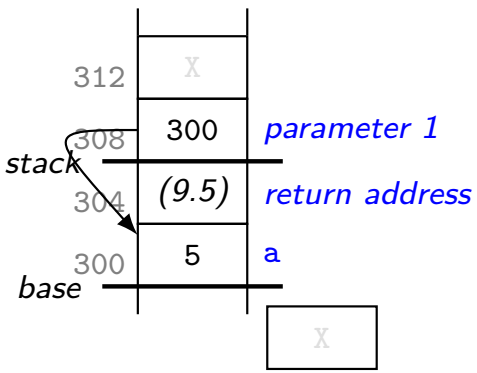


Parameteröverföring (3)

- Vi testkör det andra examplet!

code/add-one2.c

```
1  #include <stdio.h>
2  void add_one(int *n)
3  {
4      *n = *n + 1;
5  }
6  int main(void)
7  {
8      int a = 5;
9      add_one(&a);
10     printf("a = %d\n", a);
11     return 0;
12 }
```

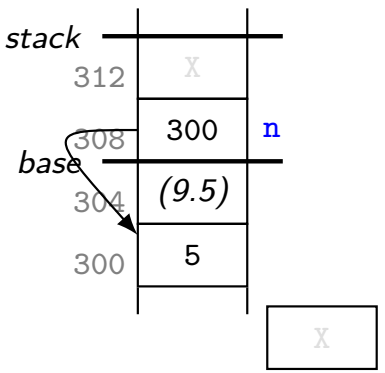


Parameteröverföring (3)

- Vi testkör det andra examplet!

code/add-one2.c

```
1  #include <stdio.h>
2  void add_one(int *n)
3  {
4      *n = *n + 1;
5  }
6  int main(void)
7  {
8      int a = 5;
9      add_one(&a);
10     printf("a = %d\n", a);
11     return 0;
12 }
```

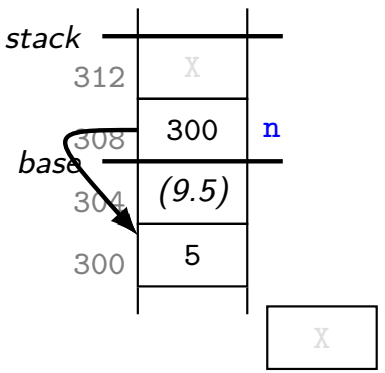


Parameteröverföring (3)

- Vi testkör det andra examplet!

code/add-one2.c

```
1  #include <stdio.h>
2  void add_one(int *n)
3  {
4      *n = *n + 1;
5  }
6  int main(void)
7  {
8      int a = 5;
9      add_one(&a);
10     printf("a = %d\n", a);
11     return 0;
12 }
```

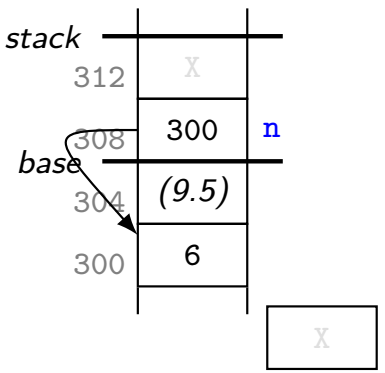


Parameteröverföring (3)

- Vi testkör det andra examplet!

code/add-one2.c

```
1  #include <stdio.h>
2  void add_one(int *n)
3  {
4      *n = *n + 1;
5  }
6  int main(void)
7  {
8      int a = 5;
9      add_one(&a);
10     printf("a = %d\n", a);
11     return 0;
12 }
```

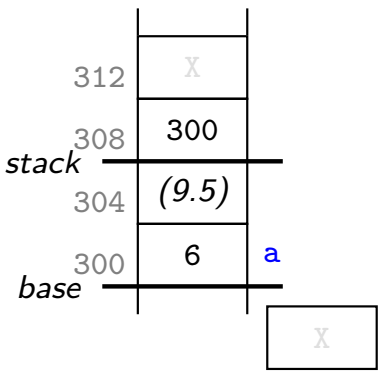


Parameteröverföring (3)

- Vi testkör det andra examplet!

code/add-one2.c

```
1  #include <stdio.h>
2  void add_one(int *n)
3  {
4      *n = *n + 1;
5  }
6  int main(void)
7  {
8      int a = 5;
9      add_one(&a);
10     printf("a = %d\n", a);
11     return 0;
12 }
```

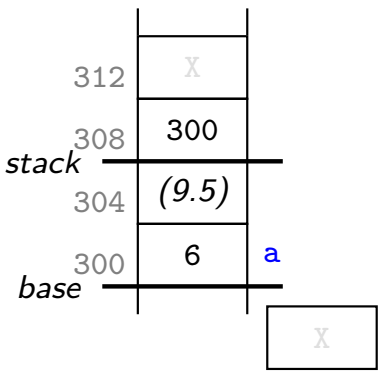


Parameteröverföring (3)

- Vi testkör det andra examplet!

code/add-one2.c

```
1  #include <stdio.h>
2  void add_one(int *n)
3  {
4      *n = *n + 1;
5  }
6  int main(void)
7  {
8      int a = 5;
9      add_one(&a);
10     printf("a = %d\n", a);
11     return 0;
12 }
```



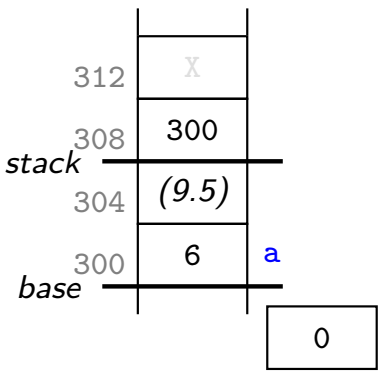

```
a = 6
```

Parameteröverföring (3)

- Vi testkör det andra examplet!

code/add-one2.c

```
1  #include <stdio.h>
2  void add_one(int *n)
3  {
4      *n = *n + 1;
5  }
6  int main(void)
7  {
8      int a = 5;
9      add_one(&a);
10     printf("a = %d\n", a);
11     return 0;
12 }
```



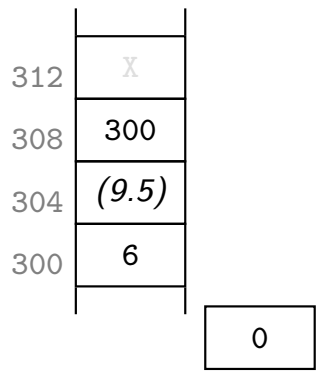

```
a = 6
```

Parameteröverföring (3)

- Vi testkör det andra examplet!

code/add-one2.c

```
1  #include <stdio.h>
2  void add_one(int *n)
3  {
4      *n = *n + 1;
5  }
6  int main(void)
7  {
8      int a = 5;
9      add_one(&a);
10     printf("a = %d\n", a);
11     return 0;
12 }
```




```
a = 6
```

Vad är skillnaden?

- När funktionen tar emot en `int`-variabel så skickas värdet lagrat i variabeln a till funktionen
  - Eftersom funktionen inte har a:s adress kan den inte ändra värdet som lagrats i a
- När funktionen tar emot en pekare och vi skickar adressen till variabeln a kan funktionen ändra vad som finns lagrat i a via pekaren
- Via pekare kan en funktion ändra variabler utanför sin *stack frame*
  - i princip var som helst i minnet...

Returvärden från funktioner (1)

- Det normala sättet att returnera värden från en funktion är med `return`
  - Fungerar för enkla datatyper, t.ex. `int`, `double`
  - Fungerar för endast ett returvärde
  - Exempel: `sin(x)`
- Pekarparametrar tar sig förbi dessa begränsningar
  - Det är möjligt att returnera egendefinierade typer (senare)
  - Det är också möjligt att ”returnera” flera värden
    - Exempel:

```
1  void swap(double *v1, double *v2)
2  {
3      double d = *v1;
4      *v1 = *v2;
5      *v2 = d;
6  }
```

Returvärden från funktioner (2)

- En del funktioner kombinerar pekarparametrar med returvärden
- Då används ofta returvärdet som en signal om allt gick som det skulle med de övriga parametrarna
- Exempel:
  - Funktionen `scanf()` tar pekare för att lagra värden
  - Returvärdet från `scanf()` är i normalfallet antalet lyckade matchningar
  - Testet

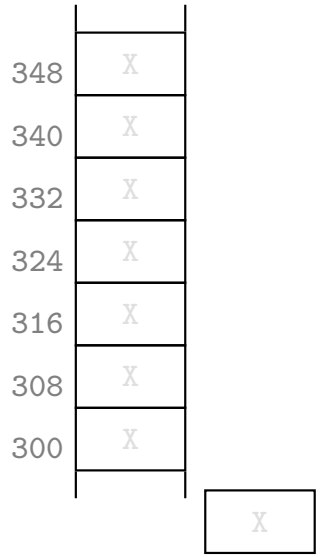
```
if (scanf("%d,%d", &a, &b) == 2) {
    // We have good values in a and b
} else {
    // Do error handling
}
```

kan användas för att säkerställa att vi bara jobbar på giltiga värden för `a` och `b`

Ett till exempel

code/swap.c

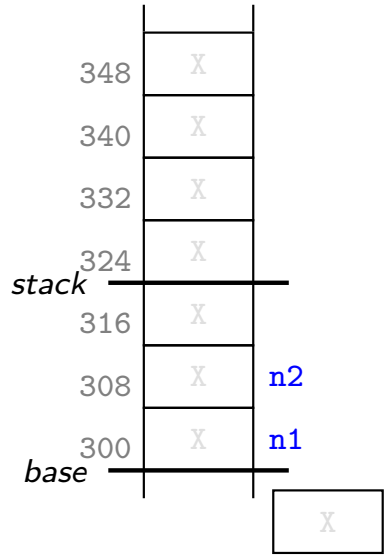
```c
#include <stdio.h>
void swap(double *v1, double *v2)
{
    double tmp;
    tmp = *v1;
    *v1 = *v2;
    *v2 = tmp;
}
int main(void)
{
    double n1 = 0.53, n2 = 7.34;
    swap(&n1, &n2);
    printf("n1 = %4.2f, n2 = %4.2f\n", n1, n2);
    return 0;
}
```



Ett till exempel

code/swap.c

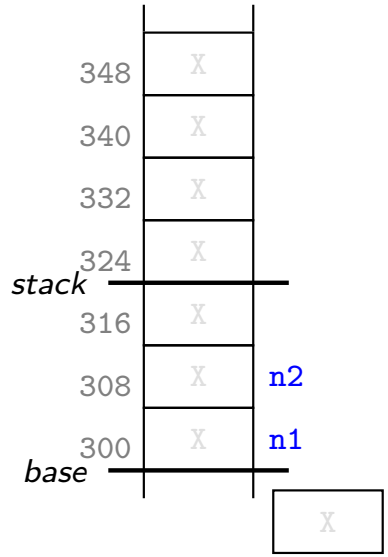
```
1  #include <stdio.h>
2  void swap(double *v1, double *v2)
3  {
4      double tmp;
5      tmp = *v1;
6      *v1 = *v2;
7      *v2 = tmp;
8  }
9  int main(void)
10 {
11     double n1 = 0.53, n2 = 7.34;
12     swap(&n1, &n2);
13     printf("n1 = %4.2f, n2 = %4.2f\n", n1, n2);
14     return 0;
15 }
```



Ett till exempel

code/swap.c

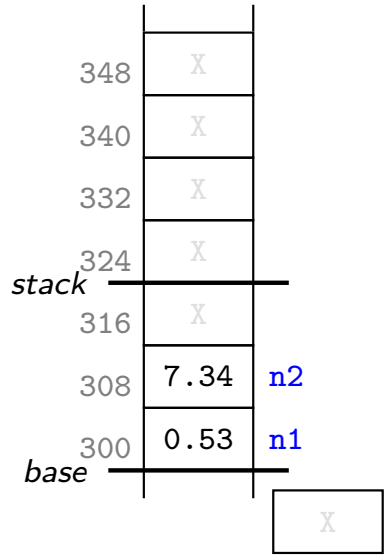
```
1  #include <stdio.h>
2  void swap(double *v1, double *v2)
3  {
4      double tmp;
5      tmp = *v1;
6      *v1 = *v2;
7      *v2 = tmp;
8  }
9  int main(void)
10 {
11     double n1 = 0.53, n2 = 7.34;
12     swap(&n1, &n2);
13     printf("n1 = %4.2f, n2 = %4.2f\n", n1, n2);
14     return 0;
15 }
```



Ett till exempel

code/swap.c

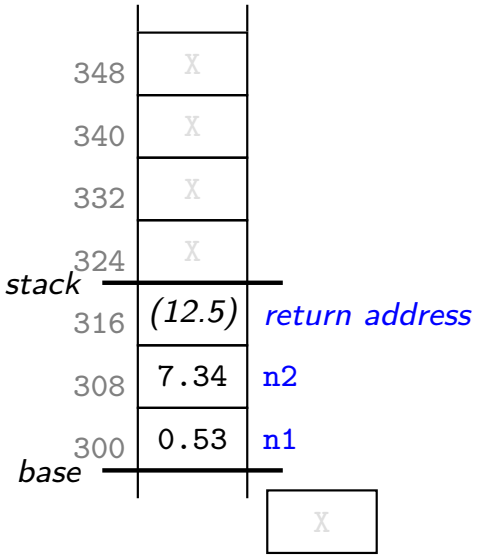
```
1  #include <stdio.h>
2  void swap(double *v1, double *v2)
3  {
4      double tmp;
5      tmp = *v1;
6      *v1 = *v2;
7      *v2 = tmp;
8  }
9  int main(void)
10 {
11     double n1 = 0.53, n2 = 7.34;
12     swap(&n1, &n2);
13     printf("n1 = %4.2f, n2 = %4.2f\n", n1, n2);
14     return 0;
15 }
```



Ett till exempel

code/swap.c

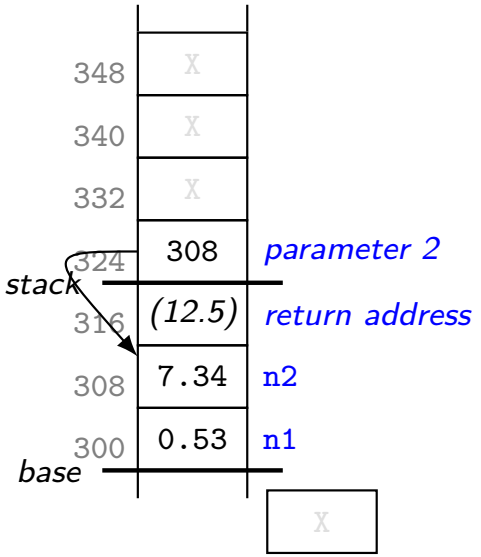
```c
#include <stdio.h>
void swap(double *v1, double *v2)
{
    double tmp;
    tmp = *v1;
    *v1 = *v2;
    *v2 = tmp;
}
int main(void)
{
    double n1 = 0.53, n2 = 7.34;
    swap(&n1, &n2);
    printf("n1 = %4.2f, n2 = %4.2f\n", n1, n2);
    return 0;
}
```



Ett till exempel

code/swap.c

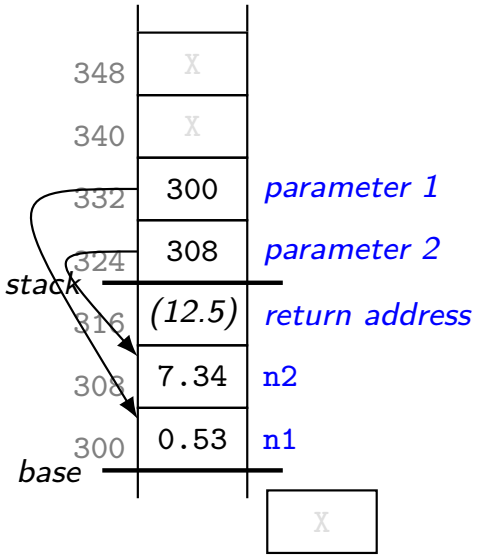
```c
#include <stdio.h>
void swap(double *v1, double *v2)
{
    double tmp;
    tmp = *v1;
    *v1 = *v2;
    *v2 = tmp;
}
int main(void)
{
    double n1 = 0.53, n2 = 7.34;
    swap(&n1, &n2);
    printf("n1 = %4.2f, n2 = %4.2f\n", n1, n2);
    return 0;
}
```



Ett till exempel

code/swap.c

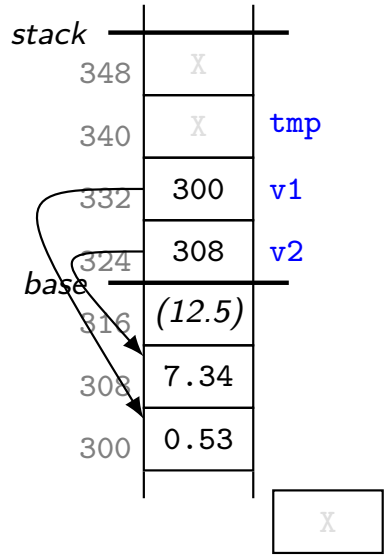
```c
#include <stdio.h>
void swap(double *v1, double *v2)
{
    double tmp;
    tmp = *v1;
    *v1 = *v2;
    *v2 = tmp;
}
int main(void)
{
    double n1 = 0.53, n2 = 7.34;
    swap(&n1, &n2);
    printf("n1 = %4.2f, n2 = %4.2f\n", n1, n2);
    return 0;
}
```



Ett till exempel

code/swap.c

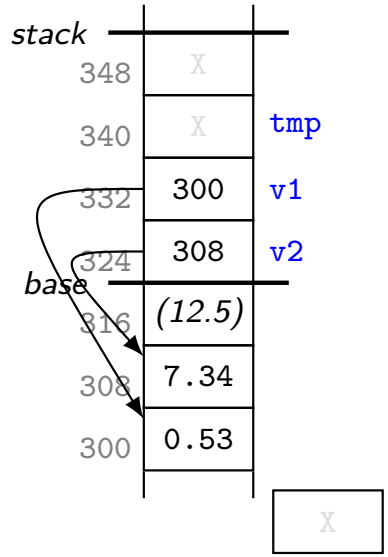
```
1   #include <stdio.h>
2   void swap(double *v1, double *v2)
3   {
4       double tmp;
5       tmp = *v1;
6       *v1 = *v2;
7       *v2 = tmp;
8   }
9   int main(void)
10  {
11      double n1 = 0.53, n2 = 7.34;
12      swap(&n1, &n2);
13      printf("n1 = %4.2f, n2 = %4.2f\n", n1, n2);
14      return 0;
15  }
```



Ett till exempel

code/swap.c

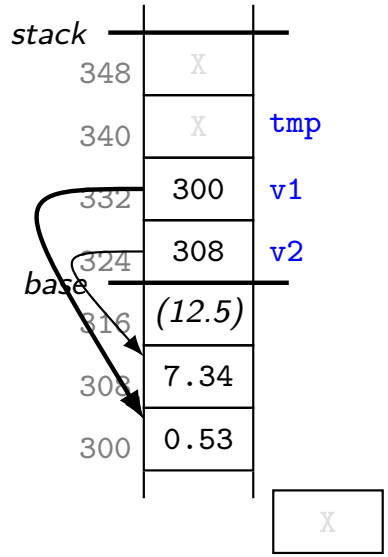
```c
#include <stdio.h>
void swap(double *v1, double *v2)
{
    double tmp;
    tmp = *v1;
    *v1 = *v2;
    *v2 = tmp;
}
int main(void)
{
    double n1 = 0.53, n2 = 7.34;
    swap(&n1, &n2);
    printf("n1 = %4.2f, n2 = %4.2f\n", n1, n2);
    return 0;
}
```



Ett till exempel

code/swap.c

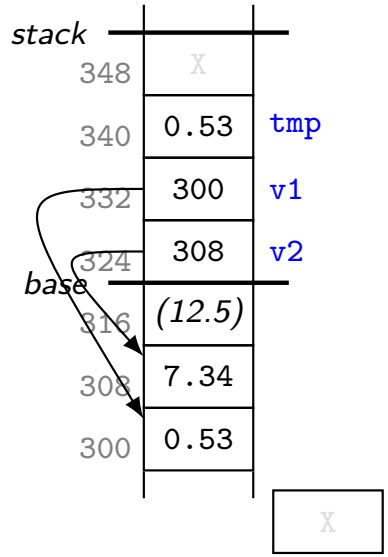
```
1  #include <stdio.h>
2  void swap(double *v1, double *v2)
3  {
4      double tmp;
5      tmp = *v1;
6      *v1 = *v2;
7      *v2 = tmp;
8  }
9  int main(void)
10 {
11     double n1 = 0.53, n2 = 7.34;
12     swap(&n1, &n2);
13     printf("n1 = %4.2f, n2 = %4.2f\n", n1, n2);
14     return 0;
15 }
```



Ett till exempel

code/swap.c

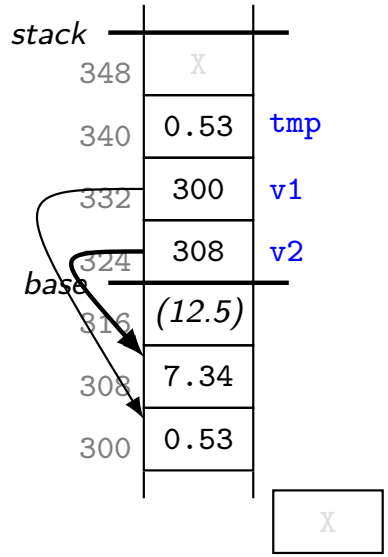
```c
#include <stdio.h>
void swap(double *v1, double *v2)
{
    double tmp;
    tmp = *v1;
    *v1 = *v2;
    *v2 = tmp;
}
int main(void)
{
    double n1 = 0.53, n2 = 7.34;
    swap(&n1, &n2);
    printf("n1 = %4.2f, n2 = %4.2f\n", n1, n2);
    return 0;
}
```



Ett till exempel

code/swap.c

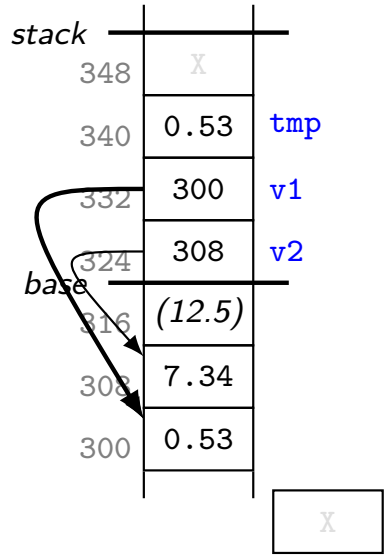
```
1  #include <stdio.h>
2  void swap(double *v1, double *v2)
3  {
4      double tmp;
5      tmp = *v1;
6      *v1 = *v2;
7      *v2 = tmp;
8  }
9  int main(void)
10 {
11     double n1 = 0.53, n2 = 7.34;
12     swap(&n1, &n2);
13     printf("n1 = %4.2f, n2 = %4.2f\n", n1, n2);
14     return 0;
15 }
```



Ett till exempel

code/swap.c

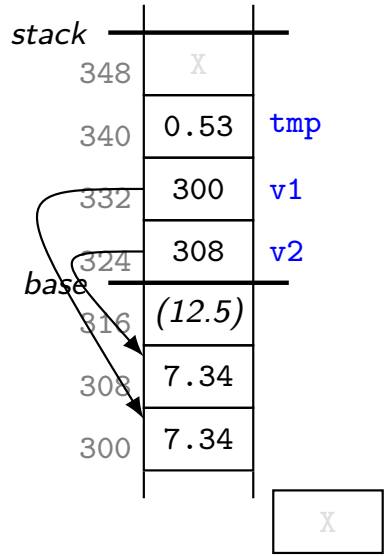
```
1  #include <stdio.h>
2  void swap(double *v1, double *v2)
3  {
4      double tmp;
5      tmp = *v1;
6      *v1 = *v2;
7      *v2 = tmp;
8  }
9  int main(void)
10 {
11     double n1 = 0.53, n2 = 7.34;
12     swap(&n1, &n2);
13     printf("n1 = %4.2f, n2 = %4.2f\n", n1, n2);
14     return 0;
15 }
```



Ett till exempel

code/swap.c

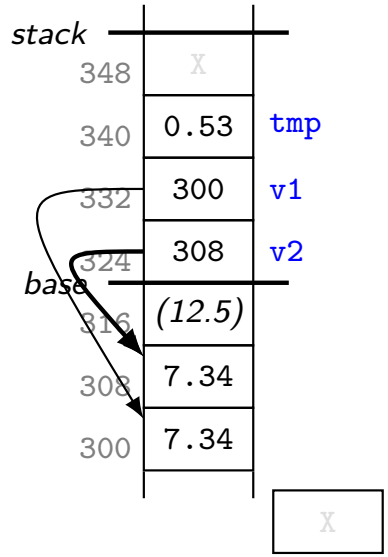
```
1   #include <stdio.h>
2   void swap(double *v1, double *v2)
3   {
4       double tmp;
5       tmp = *v1;
6       *v1 = *v2;
7       *v2 = tmp;
8   }
9   int main(void)
10  {
11      double n1 = 0.53, n2 = 7.34;
12      swap(&n1, &n2);
13      printf("n1 = %4.2f, n2 = %4.2f\n", n1, n2);
14      return 0;
15  }
```



Ett till exempel

code/swap.c

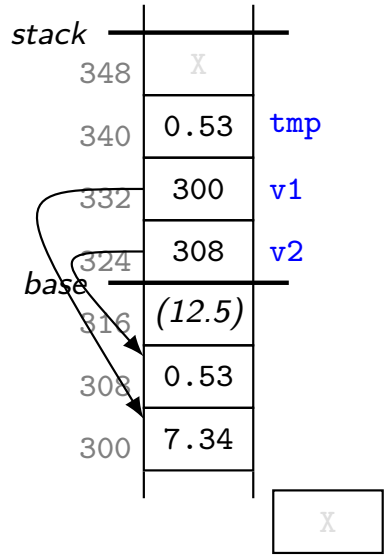
```
1  #include <stdio.h>
2  void swap(double *v1, double *v2)
3  {
4      double tmp;
5      tmp = *v1;
6      *v1 = *v2;
7      *v2 = tmp;
8  }
9  int main(void)
10 {
11     double n1 = 0.53, n2 = 7.34;
12     swap(&n1, &n2);
13     printf("n1 = %4.2f, n2 = %4.2f\n", n1, n2);
14     return 0;
15 }
```



Ett till exempel

code/swap.c

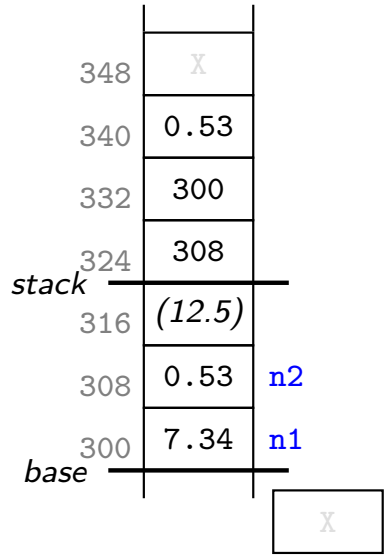
```
1  #include <stdio.h>
2  void swap(double *v1, double *v2)
3  {
4      double tmp;
5      tmp = *v1;
6      *v1 = *v2;
7      *v2 = tmp;
8  }
9  int main(void)
10 {
11     double n1 = 0.53, n2 = 7.34;
12     swap(&n1, &n2);
13     printf("n1 = %4.2f, n2 = %4.2f\n", n1, n2);
14     return 0;
15 }
```



Ett till exempel

code/swap.c

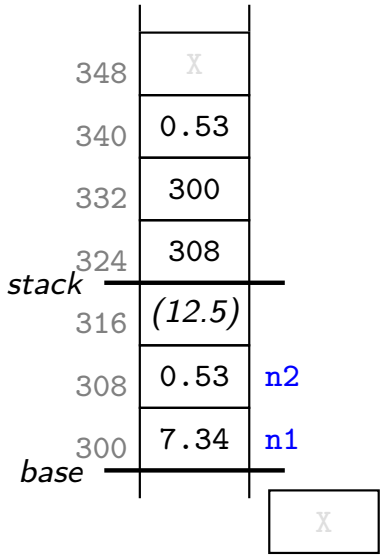
```c
#include <stdio.h>
void swap(double *v1, double *v2)
{
    double tmp;
    tmp = *v1;
    *v1 = *v2;
    *v2 = tmp;
}
int main(void)
{
    double n1 = 0.53, n2 = 7.34;
    swap(&n1, &n2);
    printf("n1 = %4.2f, n2 = %4.2f\n", n1, n2);
    return 0;
}
```



Ett till exempel

code/swap.c

```c
#include <stdio.h>
void swap(double *v1, double *v2)
{
    double tmp;
    tmp = *v1;
    *v1 = *v2;
    *v2 = tmp;
}
int main(void)
{
    double n1 = 0.53, n2 = 7.34;
    swap(&n1, &n2);
    printf("n1 = %4.2f, n2 = %4.2f\n", n1, n2);
    return 0;
}
```

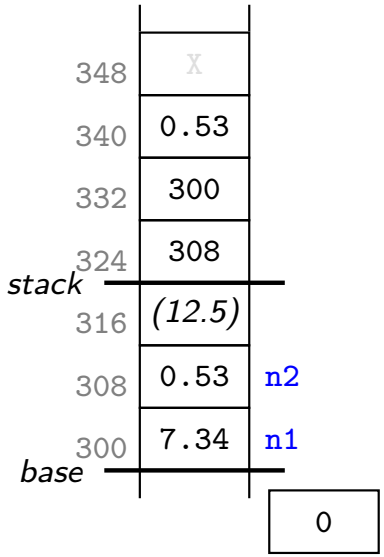



```
n1 = 7.34, n2 = 0.53
```

Ett till exempel

code/swap.c

```c
#include <stdio.h>
void swap(double *v1, double *v2)
{
    double tmp;
    tmp = *v1;
    *v1 = *v2;
    *v2 = tmp;
}
int main(void)
{
    double n1 = 0.53, n2 = 7.34;
    swap(&n1, &n2);
    printf("n1 = %4.2f, n2 = %4.2f\n", n1, n2);
    return 0;
}
```

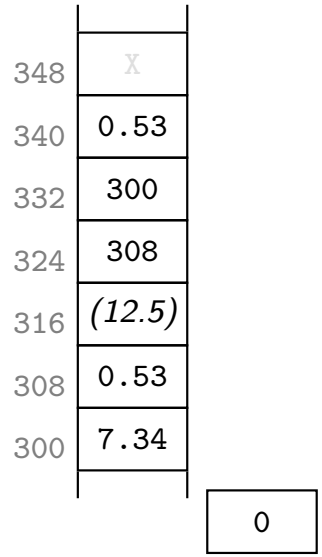



```
n1 = 7.34, n2 = 0.53
```

Ett till exempel

code/swap.c

```c
#include <stdio.h>
void swap(double *v1, double *v2)
{
    double tmp;
    tmp = *v1;
    *v1 = *v2;
    *v2 = tmp;
}
int main(void)
{
    double n1 = 0.53, n2 = 7.34;
    swap(&n1, &n2);
    printf("n1 = %4.2f, n2 = %4.2f\n", n1, n2);
    return 0;
}
```




```
n1 = 7.34, n2 = 0.53
```

Variabler och fält

Variabler

- En variabel har ett namn
- En variabel är en namngiven plats i minnet
- En variabel är bundet till ett värde
- Värdet kan ändras dynamiskt genom tilldelning

Variablers begränsningar

- Lagra ett provresultat:

```
int res = 7;
```

- Lagra tre provresultat:

```
int res0 = 7;
int res1 = 9;
int res2 = 4;
```

- Lagra N provresultat:
  - ???
- Beräkna medelvärdet av tjugo provresultat:

```
int res0, res1, res2, res3, res4, res5, res6, res7, res8, res9,
    res10, res12, res13, res14, res15, res16, res17, res18, res19;
double avg = (res0 + res1 + res2 + res3 + res4 + res5 + res6
    + res7 + res8 + res9 + res10 + res12 + res13 + res14 + res15
    + res16 + res17 + res18 + res19) / 20;
```

Lösningen: Fält (*array*)

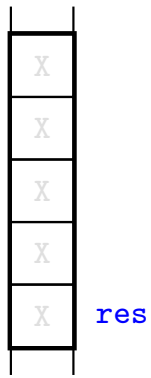
- Program arbetar ofta med stora mängder värden som hör ihop
  - Att gruppera sådana värden konceptuellt underlättar för programmeraren
  - Att gruppera värdena fysiskt (i datorns minne) gör program mer effektiva
- Ett fält är ett objekt som kan lagra flera element av samma typ
  - Varje element har ett värde
  - Elementen lagras konsekutivt i minnet

Deklaration av fält

- Ett fält måste deklareras innan användning
  - Måste ange typ, namn och storlek
- Denna kod

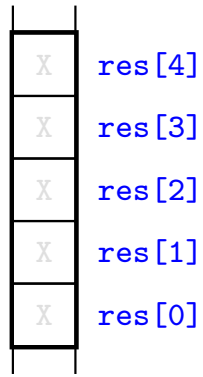
```
int res[5];
```

säger åt kompilatorn att reservera 5 konsekutiva minnesplatser som kan lagra värden av typ `int` och att associera denna sekvens med namnet `res`



Indexering

- ”Kärt barn har många namn”
  - fält
  - arrayer
  - vektorer
  - matriser
  - indexerade variabler
- Man refererar till elementen i ett fält med hjälp av heltalsindexering
  - Varje lagringsplats i fältet har eget index
    - `res[0]` är det första elementet
    - `res[4]` är det sista elementet
  - Variabelnamnet `res` refererar till hela fältet



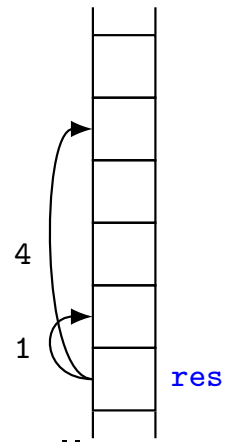
Indexet är ett *uttryck*

- Det är en avgörande skillnad mellan dessa två:

  `res1` en variabel med namnet `res1`

  `res[1]` ett element med index 1 i variabeln (fältet) `res`

- Skillnaden är att indexet är ett uttryck som evalueras under exekveringen av koden
- Vi kan därför uttrycka saker som
  - `res[a + b]`
  - Om a = 1 och b = 3 kommer `res[a + b]` att referera till samma element som `res[4]`
- Kompilatorn översätter texten `res[k]` som en referens till minnet `k` stycken steg efter i minnet från `res`
  - Storleken på varje steg bestäms av typen för `res`
  - Om `res` börjar på adress 300 och varje element tar 4 bytes så kommer `res[4]` att referera till minne på adress 316

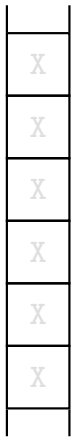


Initiering

- Precis som med alla andra variabler så måste vi initiera elementen i ett fält

code/init-res.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int res[5];
6      res[0] = 2;
7      res[1] = 6;
8      res[2] = 19;
9      res[3] = -1;
10     res[4] = 1;
11     return 0;
12 }
```

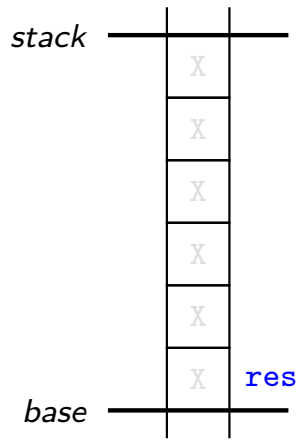


Initiering

- Precis som med alla andra variabler så måste vi initiera elementen i ett fält

code/init-res.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int res[5];
6      res[0] = 2;
7      res[1] = 6;
8      res[2] = 19;
9      res[3] = -1;
10     res[4] = 1;
11     return 0;
12 }
```

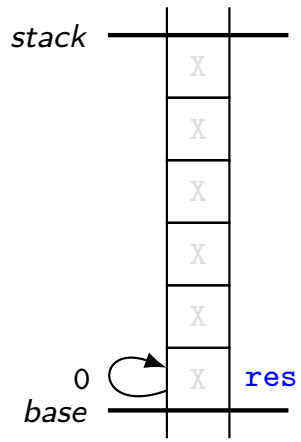


Initiering

- Precis som med alla andra variabler så måste vi initiera elementen i ett fält

code/init-res.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int res[5];
6      res[0] = 2;
7      res[1] = 6;
8      res[2] = 19;
9      res[3] = -1;
10     res[4] = 1;
11     return 0;
12 }
```

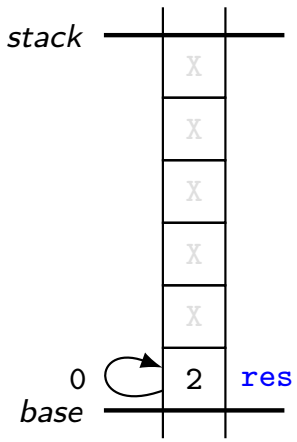


Initiering

- Precis som med alla andra variabler så måste vi initiera elementen i ett fält

code/init-res.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int res[5];
6      res[0] = 2;
7      res[1] = 6;
8      res[2] = 19;
9      res[3] = -1;
10     res[4] = 1;
11     return 0;
12 }
```

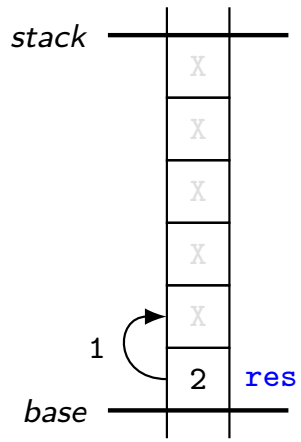


Initiering

- Precis som med alla andra variabler så måste vi initiera elementen i ett fält

code/init-res.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int res[5];
6      res[0] = 2;
7      res[1] = 6;
8      res[2] = 19;
9      res[3] = -1;
10     res[4] = 1;
11     return 0;
12 }
```

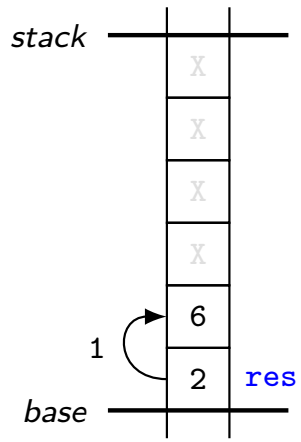


Initiering

- Precis som med alla andra variabler så måste vi initiera elementen i ett fält

code/init-res.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int res[5];
6      res[0] = 2;
7      res[1] = 6;
8      res[2] = 19;
9      res[3] = -1;
10     res[4] = 1;
11     return 0;
12 }
```

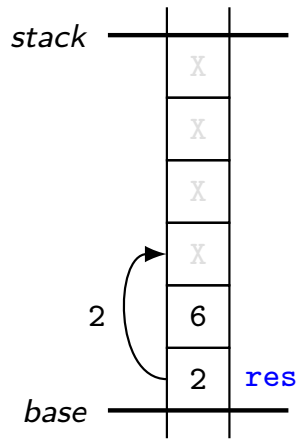


Initiering

- Precis som med alla andra variabler så måste vi initiera elementen i ett fält

code/init-res.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int res[5];
6      res[0] = 2;
7      res[1] = 6;
8      res[2] = 19;
9      res[3] = -1;
10     res[4] = 1;
11     return 0;
12 }
```

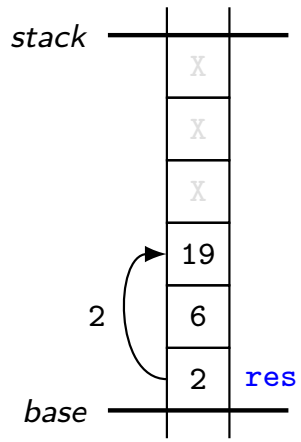


Initiering

- Precis som med alla andra variabler så måste vi initiera elementen i ett fält

code/init-res.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int res[5];
6      res[0] = 2;
7      res[1] = 6;
8      res[2] = 19;
9      res[3] = -1;
10     res[4] = 1;
11     return 0;
12 }
```

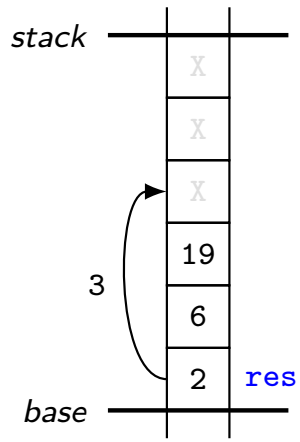


Initiering

- Precis som med alla andra variabler så måste vi initiera elementen i ett fält

code/init-res.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int res[5];
6      res[0] = 2;
7      res[1] = 6;
8      res[2] = 19;
9      res[3] = -1;
10     res[4] = 1;
11     return 0;
12 }
```

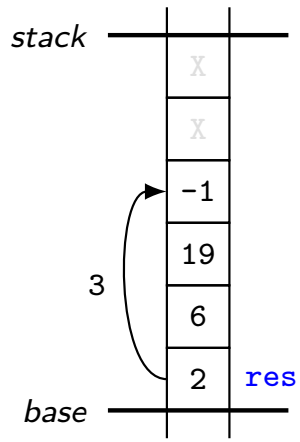


Initiering

- Precis som med alla andra variabler så måste vi initiera elementen i ett fält

code/init-res.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int res[5];
6      res[0] = 2;
7      res[1] = 6;
8      res[2] = 19;
9      res[3] = -1;
10     res[4] = 1;
11     return 0;
12 }
```



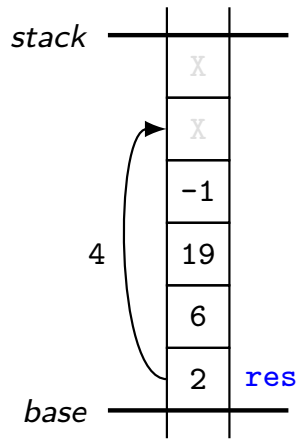
Initiering

- Precis som med alla andra variabler så måste vi initiera elementen i ett fält

code/init-res.c

```c
#include <stdio.h>

int main(void)
{
    int res[5];
    res[0] = 2;
    res[1] = 6;
    res[2] = 19;
    res[3] = -1;
    res[4] = 1;
    return 0;
}
```



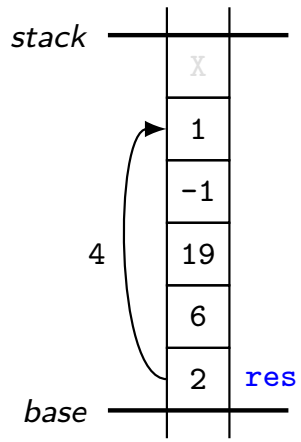
Initiering

- Precis som med alla andra variabler så måste vi initiera elementen i ett fält

code/init-res.c

```c
#include <stdio.h>

int main(void)
{
    int res[5];
    res[0] = 2;
    res[1] = 6;
    res[2] = 19;
    res[3] = -1;
    res[4] = 1;
    return 0;
}
```

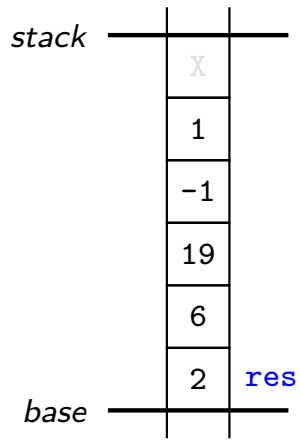


Initiering

- Precis som med alla andra variabler så måste vi initiera elementen i ett fält

code/init-res.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int res[5];
6      res[0] = 2;
7      res[1] = 6;
8      res[2] = 19;
9      res[3] = -1;
10     res[4] = 1;
11     return 0;
12 }
```

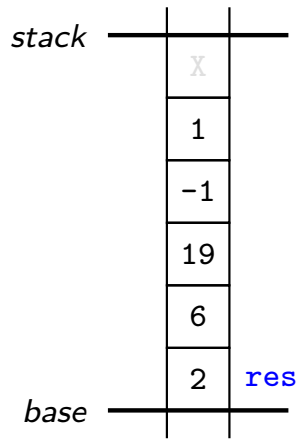


Initiering

- Precis som med alla andra variabler så måste vi initiera elementen i ett fält

code/init-res.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int res[5];
6      res[0] = 2;
7      res[1] = 6;
8      res[2] = 19;
9      res[3] = -1;
10     res[4] = 1;
11     return 0;
12 }
```

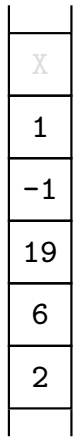


Initiering

- Precis som med alla andra variabler så måste vi initiera elementen i ett fält

code/init-res.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int res[5];
6      res[0] = 2;
7      res[1] = 6;
8      res[2] = 19;
9      res[3] = -1;
10     res[4] = 1;
11     return 0;
12 }
```



Pekare och fält, indexering

- Det går att indexera sig relativt en pekare också
- Indexeringen fungerar nästan likadant för en pekare som för ett fält
- För ett fält används adressen för fältet som utgångspunkt
  - För en pekare så används adressen i pekarvariabeln
- Syntaxen och semantiken är i övrigt identisk

code/init-res2.c

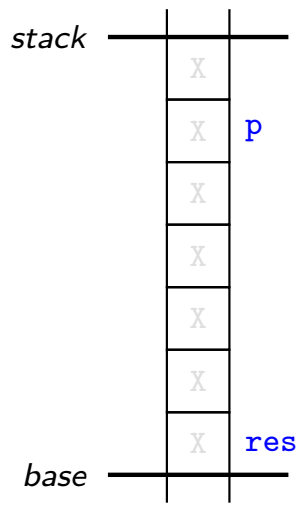
```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int res[5];
6      int *p = res;
7      p[0] = 2;
8      p[1] = 6;
9      p[2] = 19;
10     p[3] = -1;
11     p[4] = 1;
12     return 0;
13 }
```

Pekare och fält, indexering

- Det går att indexera sig relativt en pekare också
- Indexeringen fungerar nästan likadant för en pekare som för ett fält
- För ett fält används adressen för fältet som utgångspunkt
  - För en pekare så används adressen i pekarvariabeln
- Syntaxen och semantiken är i övrigt identisk

code/init-res2.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int res[5];
6      int *p = res;
7      p[0] = 2;
8      p[1] = 6;
9      p[2] = 19;
10     p[3] = -1;
11     p[4] = 1;
12     return 0;
13 }
```

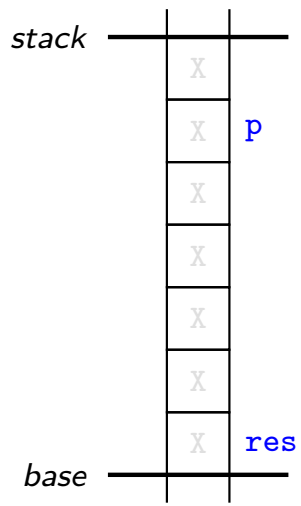


Pekare och fält, indexering

- Det går att indexera sig relativt en pekare också
- Indexeringen fungerar nästan likadant för en pekare som för ett fält
- För ett fält används adressen för fältet som utgångspunkt
  - För en pekare så används adressen i pekarvariabeln
- Syntaxen och semantiken är i övrigt identisk

code/init-res2.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int res[5];
6      int *p = res;
7      p[0] = 2;
8      p[1] = 6;
9      p[2] = 19;
10     p[3] = -1;
11     p[4] = 1;
12     return 0;
13 }
```

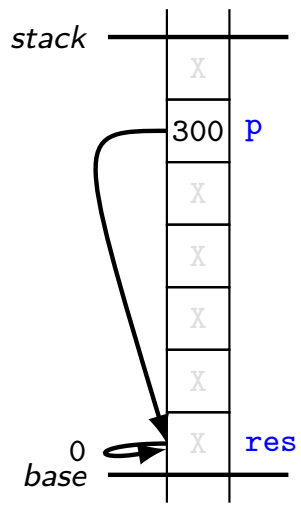


Pekare och fält, indexering

- Det går att indexera sig relativt en pekare också
- Indexeringen fungerar nästan likadant för en pekare som för ett fält
- För ett fält används adressen för fältet som utgångspunkt
  - För en pekare så används adressen i pekarvariabeln
- Syntaxen och semantiken är i övrigt identisk

code/init-res2.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int res[5];
6      int *p = res;
7      p[0] = 2;
8      p[1] = 6;
9      p[2] = 19;
10     p[3] = -1;
11     p[4] = 1;
12     return 0;
13 }
```

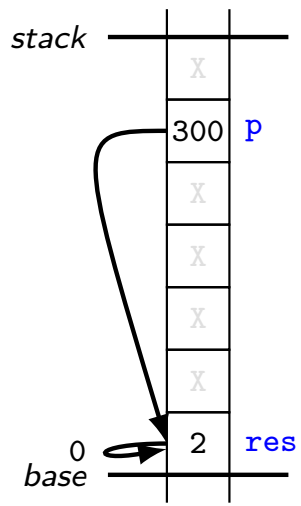


Pekare och fält, indexering

- Det går att indexera sig relativt en pekare också
- Indexeringen fungerar nästan likadant för en pekare som för ett fält
- För ett fält används adressen för fältet som utgångspunkt
  - För en pekare så används adressen i pekarvariabeln
- Syntaxen och semantiken är i övrigt identisk

code/init-res2.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int res[5];
6      int *p = res;
7      p[0] = 2;
8      p[1] = 6;
9      p[2] = 19;
10     p[3] = -1;
11     p[4] = 1;
12     return 0;
13 }
```

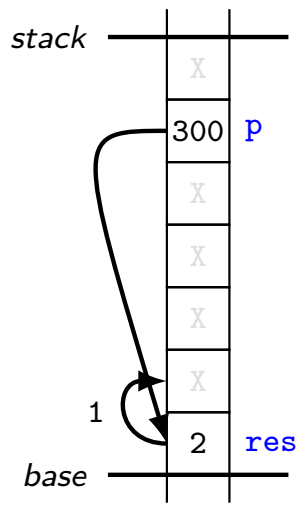


Pekare och fält, indexering

- Det går att indexera sig relativt en pekare också
- Indexeringen fungerar nästan likadant för en pekare som för ett fält
- För ett fält används adressen för fältet som utgångspunkt
  - För en pekare så används adressen i pekarvariabeln
- Syntaxen och semantiken är i övrigt identisk

code/init-res2.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int res[5];
6      int *p = res;
7      p[0] = 2;
8      p[1] = 6;
9      p[2] = 19;
10     p[3] = -1;
11     p[4] = 1;
12     return 0;
13 }
```

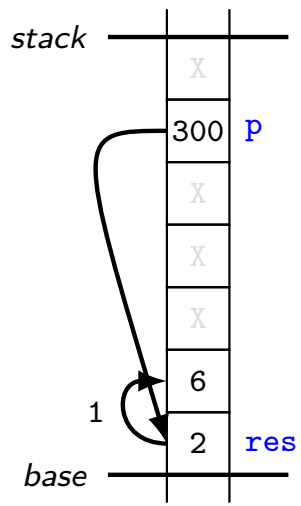


Pekare och fält, indexering

- Det går att indexera sig relativt en pekare också
- Indexeringen fungerar nästan likadant för en pekare som för ett fält
- För ett fält används adressen för fältet som utgångspunkt
  - För en pekare så används adressen i pekarvariabeln
- Syntaxen och semantiken är i övrigt identisk

code/init-res2.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int res[5];
6      int *p = res;
7      p[0] = 2;
8      p[1] = 6;
9      p[2] = 19;
10     p[3] = -1;
11     p[4] = 1;
12     return 0;
13 }
```

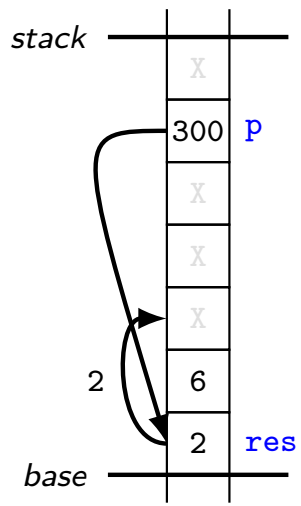


Pekare och fält, indexering

- Det går att indexera sig relativt en pekare också
- Indexeringen fungerar nästan likadant för en pekare som för ett fält
- För ett fält används adressen för fältet som utgångspunkt
  - För en pekare så används adressen i pekarvariabeln
- Syntaxen och semantiken är i övrigt identisk

code/init-res2.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int res[5];
6      int *p = res;
7      p[0] = 2;
8      p[1] = 6;
9      p[2] = 19;
10     p[3] = -1;
11     p[4] = 1;
12     return 0;
13 }
```

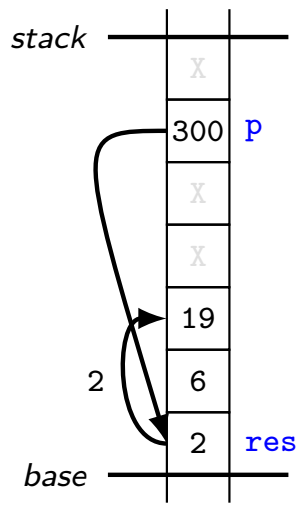


Pekare och fält, indexering

- Det går att indexera sig relativt en pekare också
- Indexeringen fungerar nästan likadant för en pekare som för ett fält
- För ett fält används adressen för fältet som utgångspunkt
  - För en pekare så används adressen i pekarvariabeln
- Syntaxen och semantiken är i övrigt identisk

code/init-res2.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int res[5];
6      int *p = res;
7      p[0] = 2;
8      p[1] = 6;
9      p[2] = 19;
10     p[3] = -1;
11     p[4] = 1;
12     return 0;
13 }
```

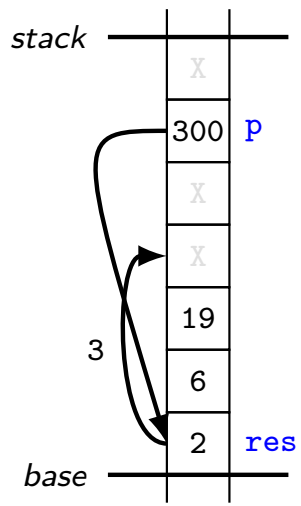


Pekare och fält, indexering

- Det går att indexera sig relativt en pekare också
- Indexeringen fungerar nästan likadant för en pekare som för ett fält
- För ett fält används adressen för fältet som utgångspunkt
  - För en pekare så används adressen i pekarvariabeln
- Syntaxen och semantiken är i övrigt identisk

code/init-res2.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int res[5];
6      int *p = res;
7      p[0] = 2;
8      p[1] = 6;
9      p[2] = 19;
10     p[3] = -1;
11     p[4] = 1;
12     return 0;
13 }
```

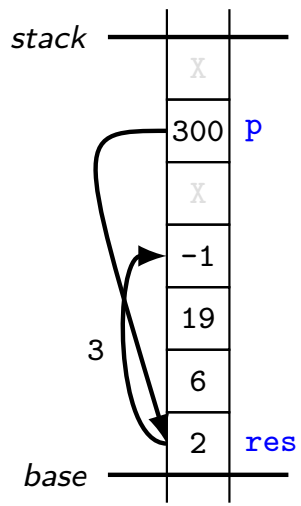


Pekare och fält, indexering

- Det går att indexera sig relativt en pekare också
- Indexeringen fungerar nästan likadant för en pekare som för ett fält
- För ett fält används adressen för fältet som utgångspunkt
  - För en pekare så används adressen i pekarvariabeln
- Syntaxen och semantiken är i övrigt identisk

code/init-res2.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int res[5];
6      int *p = res;
7      p[0] = 2;
8      p[1] = 6;
9      p[2] = 19;
10     p[3] = -1;
11     p[4] = 1;
12     return 0;
13 }
```

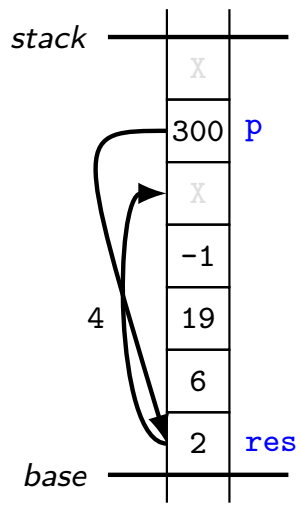


Pekare och fält, indexering

- Det går att indexera sig relativt en pekare också
- Indexeringen fungerar nästan likadant för en pekare som för ett fält
- För ett fält används adressen för fältet som utgångspunkt
  - För en pekare så används adressen i pekarvariabeln
- Syntaxen och semantiken är i övrigt identisk

code/init-res2.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int res[5];
6      int *p = res;
7      p[0] = 2;
8      p[1] = 6;
9      p[2] = 19;
10     p[3] = -1;
11     p[4] = 1;
12     return 0;
13 }
```

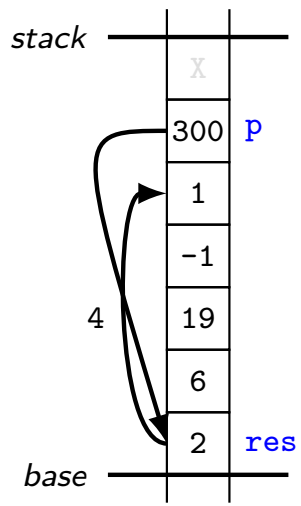


Pekare och fält, indexering

- Det går att indexera sig relativt en pekare också
- Indexeringen fungerar nästan likadant för en pekare som för ett fält
- För ett fält används adressen för fältet som utgångspunkt
  - För en pekare så används adressen i pekarvariabeln
- Syntaxen och semantiken är i övrigt identisk

code/init-res2.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int res[5];
6      int *p = res;
7      p[0] = 2;
8      p[1] = 6;
9      p[2] = 19;
10     p[3] = -1;
11     p[4] = 1;
12     return 0;
13 }
```

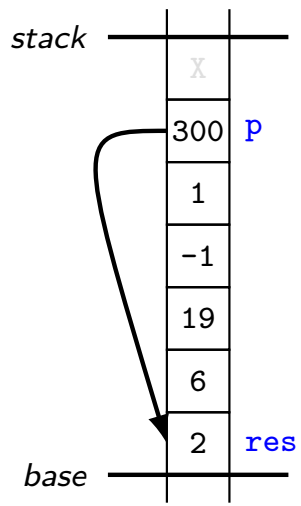


Pekare och fält, indexering

- Det går att indexera sig relativt en pekare också
- Indexeringen fungerar nästan likadant för en pekare som för ett fält
- För ett fält används adressen för fältet som utgångspunkt
  - För en pekare så används adressen i pekarvariabeln
- Syntaxen och semantiken är i övrigt identisk

code/init-res2.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int res[5];
6      int *p = res;
7      p[0] = 2;
8      p[1] = 6;
9      p[2] = 19;
10     p[3] = -1;
11     p[4] = 1;
12     return 0;
13 }
```



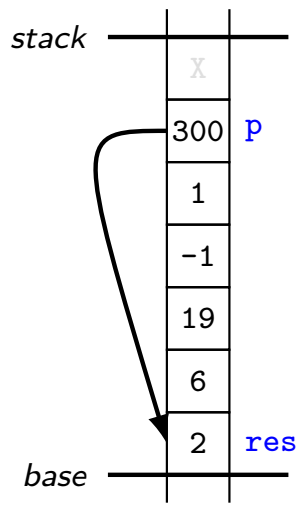
Pekare och fält, indexering

- Det går att indexera sig relativt en pekare också
- Indexeringen fungerar nästan likadant för en pekare som för ett fält
- För ett fält används adressen för fältet som utgångspunkt
  - För en pekare så används adressen i pekarvariabeln
- Syntaxen och semantiken är i övrigt identisk

code/init-res2.c

```c
#include <stdio.h>

int main(void)
{
    int res[5];
    int *p = res;
    p[0] = 2;
    p[1] = 6;
    p[2] = 19;
    p[3] = -1;
    p[4] = 1;
    return 0;
}
```



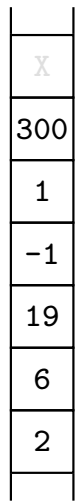
Pekare och fält, indexering

- Det går att indexera sig relativt en pekare också
- Indexeringen fungerar nästan likadant för en pekare som för ett fält
- För ett fält används adressen för fältet som utgångspunkt
  - För en pekare så används adressen i pekarvariabeln
- Syntaxen och semantiken är i övrigt identisk

code/init-res2.c

```c
#include <stdio.h>

int main(void)
{
    int res[5];
    int *p = res;
    p[0] = 2;
    p[1] = 6;
    p[2] = 19;
    p[3] = -1;
    p[4] = 1;
    return 0;
}
```



Deklarera och initiera

- Vi kan initiera fält vid deklaration enligt följande

```
int a[2] = {1, 2};
int b[] = {1, 2, 3, 4};
```

- Det är dock klokt att explicit sätta storleken på fältet, dvs.

```
int days[12] = {31,28,31,30,31,30,31,31,30,31,30,31};
```

Indexgränser

- Vad som ligger utanför de deklarerade indexgränserna är odefinierat
- Detta innebär att när följande kod körts

```
int a[2] = {1, 2};
int k = a[2]; // Refers to outside valid range!
```

  så kan `k` innehålla vad som helst!
- Vi måste själva hålla reda på hur många element ett fält har och se till att vi håller oss inom gränserna
- Vi får ingen varning vid kompilering!
- Följden vid fel kan bli
  - konstiga resultat (ett logiskt fel)
  - eller så kraschar programmet (ett s.k. runtime-fel)

Fält och iterationer (1)

- Det är ofta naturligt att iterera över indexen i ett fält:

```c
int main(void)
{
    int a[6] = {1, 2, 3, 7, 5, 24};
    for (int i = 0; i < 6; i++) {
        printf("a[%d] = %d\n", i, a[i]);
    }
    return 0;
}
```

- Utskrift:

```
a[0] = 1
a[1] = 2
a[2] = 3
a[3] = 7
a[4] = 5
a[5] = 24
```

Fält och iterationer (2)

- Ett exempel till:

```
int main(void)
{
    int a[6] = {1, 2, 3, 7, 5, 24};
    double sum = 0.0;
    for (int i = 0; i < 6; i++) {
        sum += a[i]
    }
    printf("The average value of the array is: %.2f\n",
           sum/6);
    return 0;
}
```

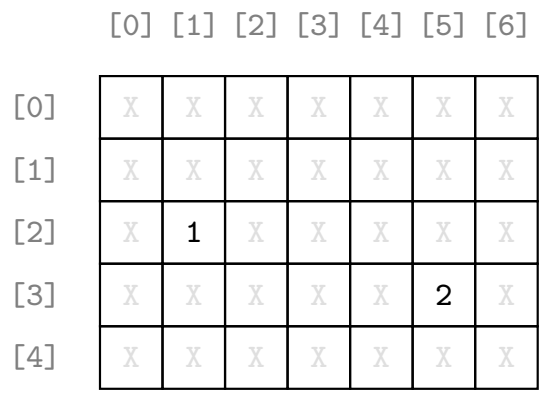
- Utskrift:

```
The average value of the array is: 7.00
```

Tvådimensionella fält

- Ett tvådimensionellt fält kallas ibland för en matris
- Ett tvådimensionellt fält av heltal med 5 rader och 7 kolumner kan t.ex. skapas med

```
int mat[5][7];
mat[2][1] = 1;
mat[3][5] = 2;
```



- Precis som tidigare räknas indexen med start i 0

Två-dimensionella fält (1)

- Det är lättast att tänka på två-dimensionella fält som en matris, med rader och kolumner
- Deklarationen `int a[3][4]` kan ses på följande vis:

`a[0][0]`	`a[0][1]`	`a[0][2]`	`a[0][3]`
`a[1][0]`	`a[1][1]`	`a[1][2]`	`a[1][3]`
`a[2][0]`	`a[2][1]`	`a[2][2]`	`a[2][3]`

- I minnet lagras dock flerdimensionella fält kontinuerligt...

`a[0][0]`	`a[0][1]`	`a[0][2]`	`a[0][3]`	`a[1][0]`	`a[1][1]`	···

- ...och det finns fortfarande inget skydd om man indexerar sig utanför gränserna...

Två-dimensionella fält (2)

- Följande tre variabler

```
int mat1[5][7] = {{0}};
int mat2[5*7];
int mat3[7][5];
```

  kommer alla att lagras som ett 35 element långt fält med `int` i minnet
- Fältet `mat1` kommer att vara initierad till bara 0:or
- Man kan skapa fält med tre eller fler dimensioner

Kodexempel med tvådimensionellt fält

- Vad skriver följande kod ut?

```
1  #include <stdio.h>
2  int main(void)
3  {
4      const int rows = 6;
5      const int columns = 10;
6      int a[rows][columns];
7      for (int i = 0 ; i < rows ; i++) {
8          for (int j = 0 ; j < columns ; j++) {
9              a[i][j] = (i + 1) * (j + 1);
10         }
11     }
12     for (int i = 0 ; i < rows ; i++) {
13         for (int j = 0 ; j < columns ; j++) {
14             printf("%2d ", a[i][j]);
15         }
16         printf("\n");
17     }
18     return 0;
19 }
```

Kodexempel med tvådimensionellt fält

- Vad skriver följande kod ut?

```
1  #include <stdio.h>
2  int main(void)
3  {
4      const int rows = 6;
5      const int columns = 10;
6      int a[rows][columns];
7      for (int i = 0 ; i < rows ; i++) {
8          for (int j = 0 ; j < columns ; j++) {
9              a[i][j] = (i + 1) * (j + 1);
10         }
11     }
12     for (int i = 0 ; i < rows ; i++) {
13         for (int j = 0 ; j < columns ; j++) {
14             printf("%2d ", a[i][j]);
15         }
16         printf("\n");
17     }
18     return 0;
19 }
```

- Utskrift

```
 1  2  3  4  5  6  7  8  9 10
 2  4  6  8 10 12 14 16 18 20
 3  6  9 12 15 18 21 24 27 30
 4  8 12 16 20 24 28 32 36 40
 5 10 15 20 25 30 35 40 45 50
 6 12 18 24 30 36 42 48 54 60
```

Fält och parametrar

Element som parametrar

- Element i ett fält fungerar som vilken variabel som helst
- Är fältet deklarerat av typen `int`

```
int arr[5];
```

så är `arr[i]` helt enkelt en `int`

```
printf("arr[%d] = %2d\n", i, arr[i]);
```

Fält som parametrar (1)

- Skickar vi fältet som parameter så skickas adressen

code/add-one-arr.c

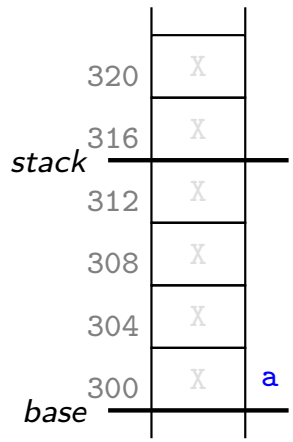
```c
#include <stdio.h>
void add_four(int *b)
{
    b[0] = b[1] + 4;
}
int main(void)
{
    int a[2] = {5, 3};
    add_four(a);
    for (int i=0; i<2; i++) {
        printf("a[%d] = %d\n", i, a[i]);
    }
    return 0;
}
```

Fält som parametrar (1)

- Skickar vi fältet som parameter så skickas adressen

code/add-one-arr.c

```
1  #include <stdio.h>
2  void add_four(int *b)
3  {
4      b[0] = b[1] + 4;
5  }
6  int main(void)
7  {
8      int a[2] = {5, 3};
9      add_four(a);
10     for (int i=0; i<2; i++) {
11         printf("a[%d] = %d\n", i, a[i]);
12     }
13     return 0;
14 }
```

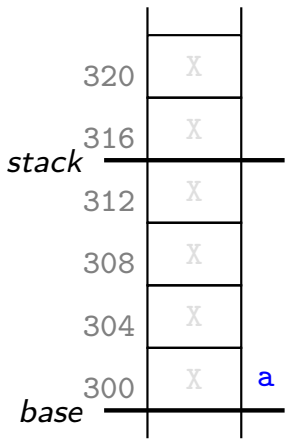


Fält som parametrar (1)

- Skickar vi fältet som parameter så skickas adressen

code/add-one-arr.c

```
1  #include <stdio.h>
2  void add_four(int *b)
3  {
4      b[0] = b[1] + 4;
5  }
6  int main(void)
7  {
8      int a[2] = {5, 3};
9      add_four(a);
10     for (int i=0; i<2; i++) {
11         printf("a[%d] = %d\n", i, a[i]);
12     }
13     return 0;
14 }
```

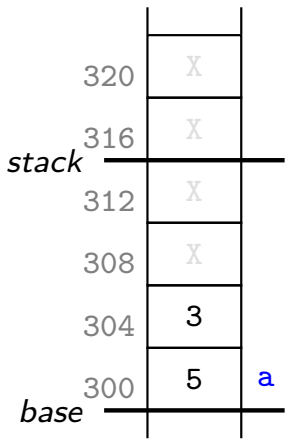


Fält som parametrar (1)

- Skickar vi fältet som parameter så skickas adressen

code/add-one-arr.c

```
1  #include <stdio.h>
2  void add_four(int *b)
3  {
4      b[0] = b[1] + 4;
5  }
6  int main(void)
7  {
8      int a[2] = {5, 3};
9      add_four(a);
10     for (int i=0; i<2; i++) {
11         printf("a[%d] = %d\n", i, a[i]);
12     }
13     return 0;
14 }
```

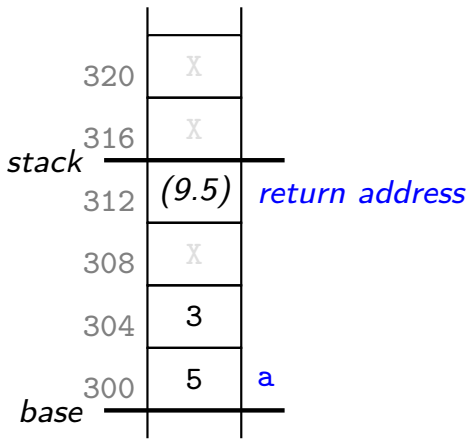


Fält som parametrar (1)

- Skickar vi fältet som parameter så skickas adressen

code/add-one-arr.c

```
1  #include <stdio.h>
2  void add_four(int *b)
3  {
4      b[0] = b[1] + 4;
5  }
6  int main(void)
7  {
8      int a[2] = {5, 3};
9      add_four(a);
10     for (int i=0; i<2; i++) {
11         printf("a[%d] = %d\n", i, a[i]);
12     }
13     return 0;
14 }
```

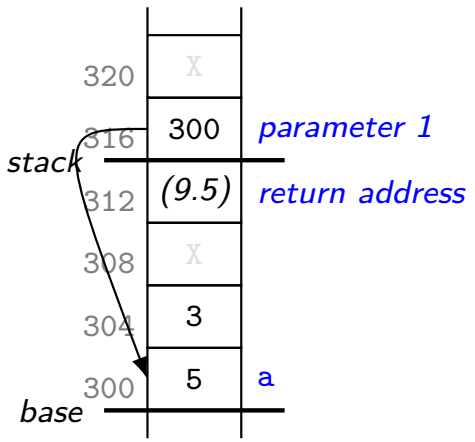


Fält som parametrar (1)

- Skickar vi fältet som parameter så skickas adressen

code/add-one-arr.c

```
1  #include <stdio.h>
2  void add_four(int *b)
3  {
4      b[0] = b[1] + 4;
5  }
6  int main(void)
7  {
8      int a[2] = {5, 3};
9      add_four(a);
10     for (int i=0; i<2; i++) {
11         printf("a[%d] = %d\n", i, a[i]);
12     }
13     return 0;
14 }
```

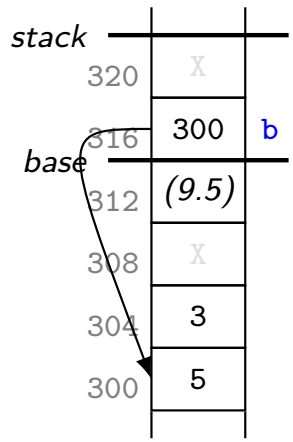


Fält som parametrar (1)

- Skickar vi fältet som parameter så skickas adressen

code/add-one-arr.c

```
1  #include <stdio.h>
2  void add_four(int *b)
3  {
4      b[0] = b[1] + 4;
5  }
6  int main(void)
7  {
8      int a[2] = {5, 3};
9      add_four(a);
10     for (int i=0; i<2; i++) {
11         printf("a[%d] = %d\n", i, a[i]);
12     }
13     return 0;
14 }
```

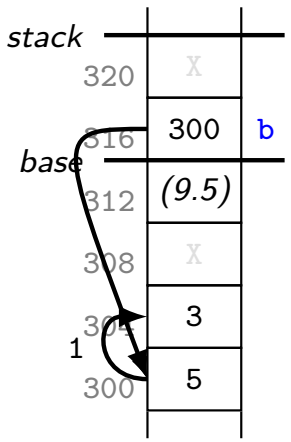


Fält som parametrar (1)

- Skickar vi fältet som parameter så skickas adressen

code/add-one-arr.c

```
1  #include <stdio.h>
2  void add_four(int *b)
3  {
4      b[0] = b[1] + 4;
5  }
6  int main(void)
7  {
8      int a[2] = {5, 3};
9      add_four(a);
10     for (int i=0; i<2; i++) {
11         printf("a[%d] = %d\n", i, a[i]);
12     }
13     return 0;
14 }
```

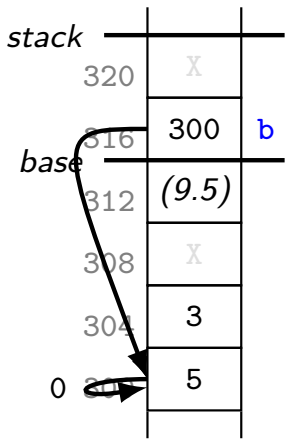


Fält som parametrar (1)

- Skickar vi fältet som parameter så skickas adressen

code/add-one-arr.c

```
1  #include <stdio.h>
2  void add_four(int *b)
3  {
4      b[0] = b[1] + 4;
5  }
6  int main(void)
7  {
8      int a[2] = {5, 3};
9      add_four(a);
10     for (int i=0; i<2; i++) {
11         printf("a[%d] = %d\n", i, a[i]);
12     }
13     return 0;
14 }
```

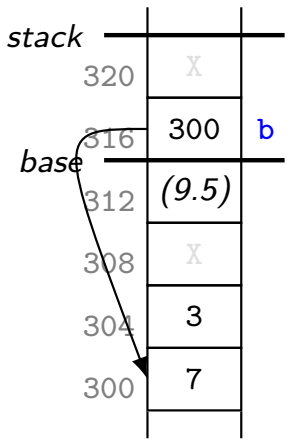


Fält som parametrar (1)

- Skickar vi fältet som parameter så skickas adressen

code/add-one-arr.c

```
1  #include <stdio.h>
2  void add_four(int *b)
3  {
4      b[0] = b[1] + 4;
5  }
6  int main(void)
7  {
8      int a[2] = {5, 3};
9      add_four(a);
10     for (int i=0; i<2; i++) {
11         printf("a[%d] = %d\n", i, a[i]);
12     }
13     return 0;
14 }
```

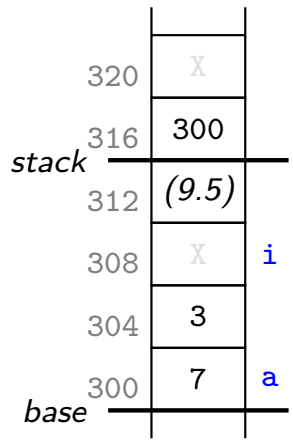


Fält som parametrar (1)

- Skickar vi fältet som parameter så skickas adressen

code/add-one-arr.c

```
1  #include <stdio.h>
2  void add_four(int *b)
3  {
4      b[0] = b[1] + 4;
5  }
6  int main(void)
7  {
8      int a[2] = {5, 3};
9      add_four(a);
10     for (int i=0; i<2; i++) {
11         printf("a[%d] = %d\n", i, a[i]);
12     }
13     return 0;
14 }
```

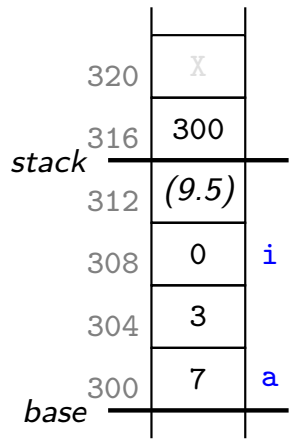


Fält som parametrar (1)

- Skickar vi fältet som parameter så skickas adressen

code/add-one-arr.c

```
1  #include <stdio.h>
2  void add_four(int *b)
3  {
4      b[0] = b[1] + 4;
5  }
6  int main(void)
7  {
8      int a[2] = {5, 3};
9      add_four(a);
10     for (int i=0; i<2; i++) {
11         printf("a[%d] = %d\n", i, a[i]);
12     }
13     return 0;
14 }
```

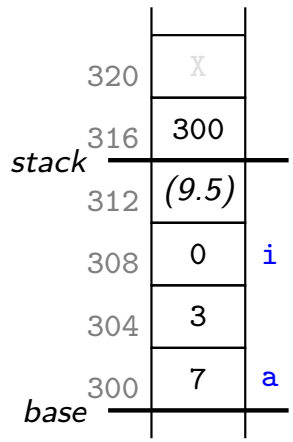


Fält som parametrar (1)

- Skickar vi fältet som parameter så skickas adressen

code/add-one-arr.c

```c
#include <stdio.h>
void add_four(int *b)
{
    b[0] = b[1] + 4;
}
int main(void)
{
    int a[2] = {5, 3};
    add_four(a);
    for (int i=0; i<2; i++) {
        printf("a[%d] = %d\n", i, a[i]);
    }
    return 0;
}
```



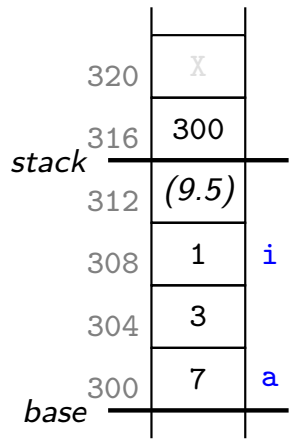

```
a[0] = 7
```

Fält som parametrar (1)

- Skickar vi fältet som parameter så skickas adressen

code/add-one-arr.c

```
1  #include <stdio.h>
2  void add_four(int *b)
3  {
4      b[0] = b[1] + 4;
5  }
6  int main(void)
7  {
8      int a[2] = {5, 3};
9      add_four(a);
10     for (int i=0; i<2; i++) {
11         printf("a[%d] = %d\n", i, a[i]);
12     }
13     return 0;
14 }
```



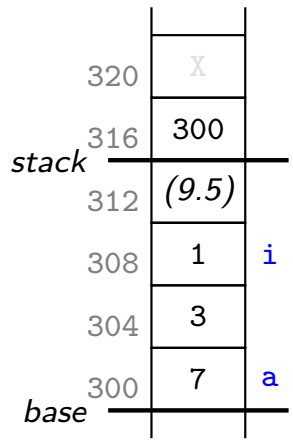

```
a[0] = 7
```

Fält som parametrar (1)

- Skickar vi fältet som parameter så skickas adressen

code/add-one-arr.c

```c
#include <stdio.h>
void add_four(int *b)
{
    b[0] = b[1] + 4;
}
int main(void)
{
    int a[2] = {5, 3};
    add_four(a);
    for (int i=0; i<2; i++) {
        printf("a[%d] = %d\n", i, a[i]);
    }
    return 0;
}
```



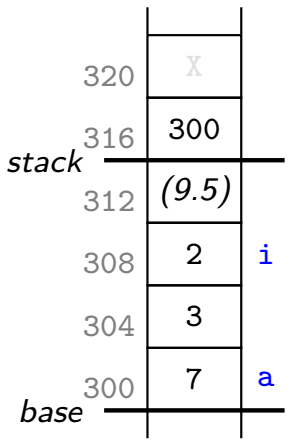

```
a[0] = 7
a[1] = 3
```

Fält som parametrar (1)

- Skickar vi fältet som parameter så skickas adressen

code/add-one-arr.c

```c
#include <stdio.h>
void add_four(int *b)
{
    b[0] = b[1] + 4;
}
int main(void)
{
    int a[2] = {5, 3};
    add_four(a);
    for (int i=0; i<2; i++) {
        printf("a[%d] = %d\n", i, a[i]);
    }
    return 0;
}
```



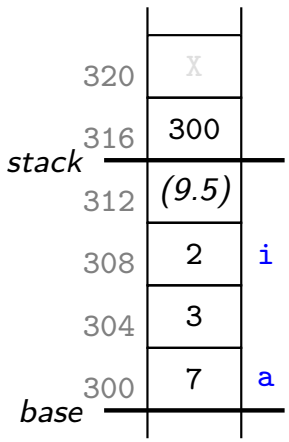

```
a[0] = 7
a[1] = 3
```

Fält som parametrar (1)

- Skickar vi fältet som parameter så skickas adressen

code/add-one-arr.c

```c
#include <stdio.h>
void add_four(int *b)
{
    b[0] = b[1] + 4;
}
int main(void)
{
    int a[2] = {5, 3};
    add_four(a);
    for (int i=0; i<2; i++) {
        printf("a[%d] = %d\n", i, a[i]);
    }
    return 0;
}
```



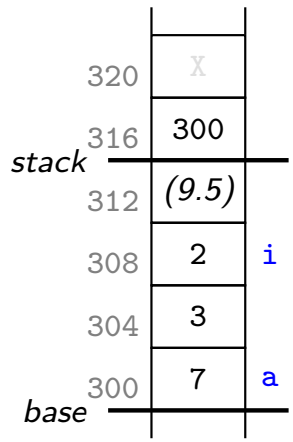

```
a[0] = 7
a[1] = 3
```

Fält som parametrar (1)

- Skickar vi fältet som parameter så skickas adressen

code/add-one-arr.c

```c
#include <stdio.h>
void add_four(int *b)
{
    b[0] = b[1] + 4;
}
int main(void)
{
    int a[2] = {5, 3};
    add_four(a);
    for (int i=0; i<2; i++) {
        printf("a[%d] = %d\n", i, a[i]);
    }
    return 0;
}
```




```
a[0] = 7
a[1] = 3
```

Fält som parametrar (2)

- Notera att fältet tas emot som en pekare

```
1  void add_four(int *b)
2  {
3      b[0] = b[1] + 4;
4  }
```

- Det går också att skriva så här:

```
1  void add_four(int b[])
2  {
3      b[0] = b[1] + 4;
4  }
```

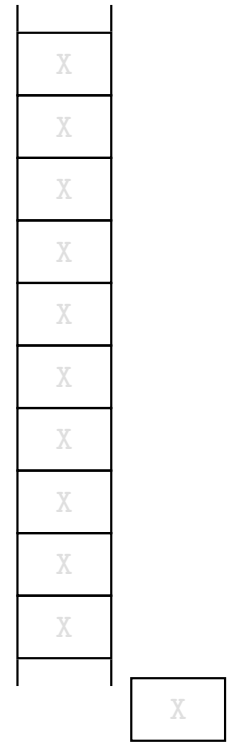
- Formuleringarna är syntaktiskt ekvivalenta

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```

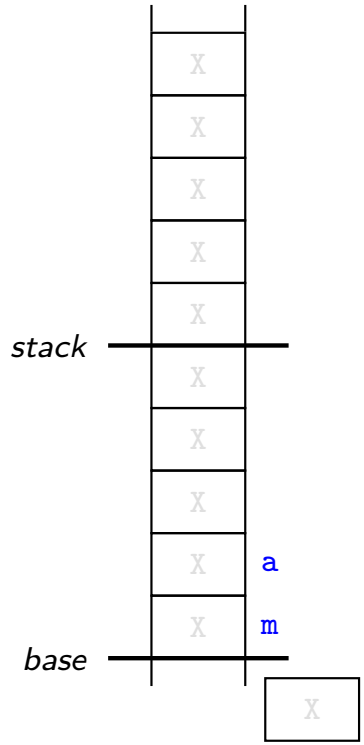


Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```

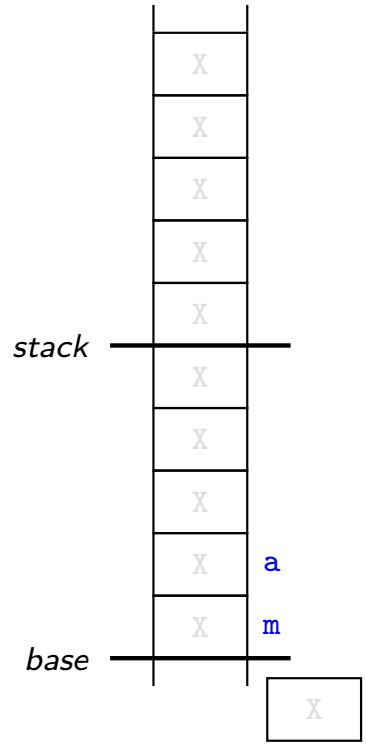


Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```

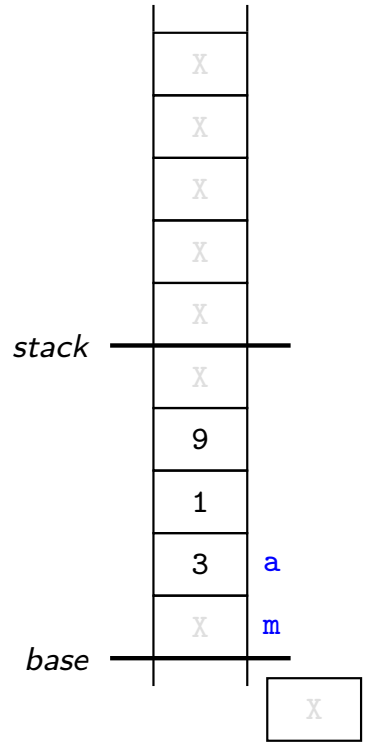


Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```

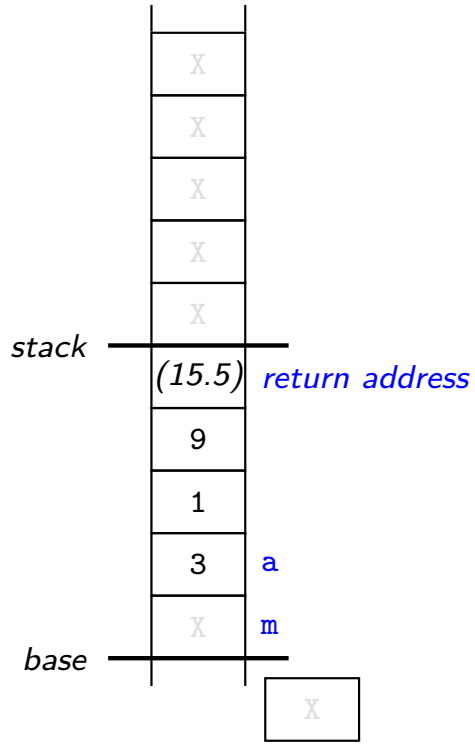


Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```

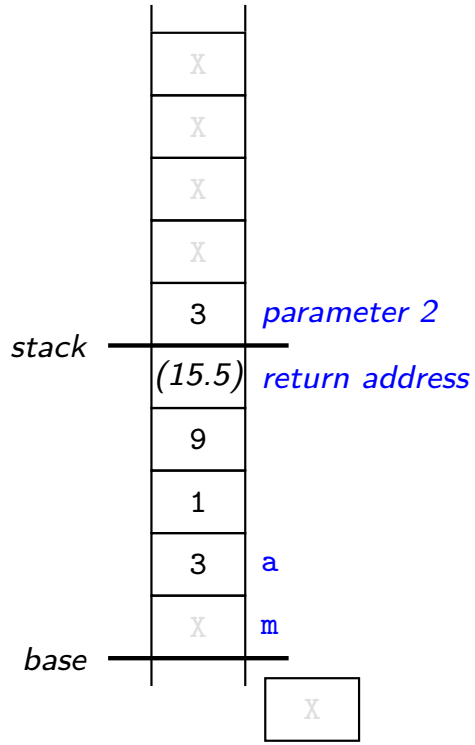


Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```

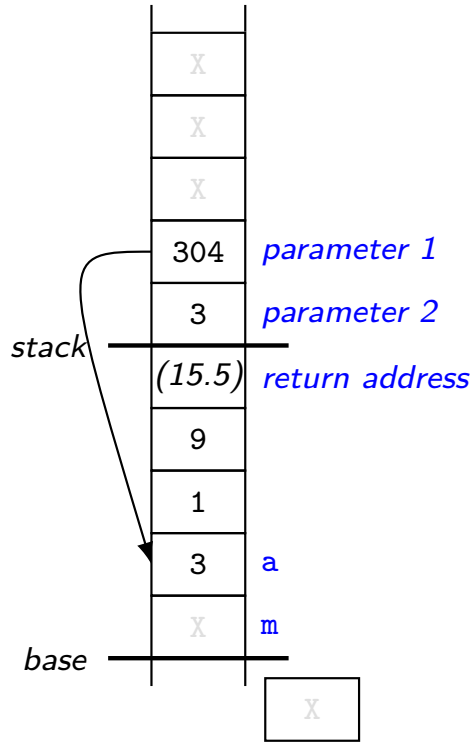


Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```

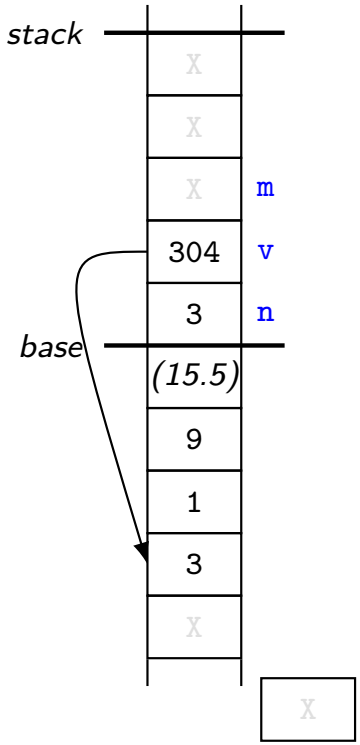


Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```

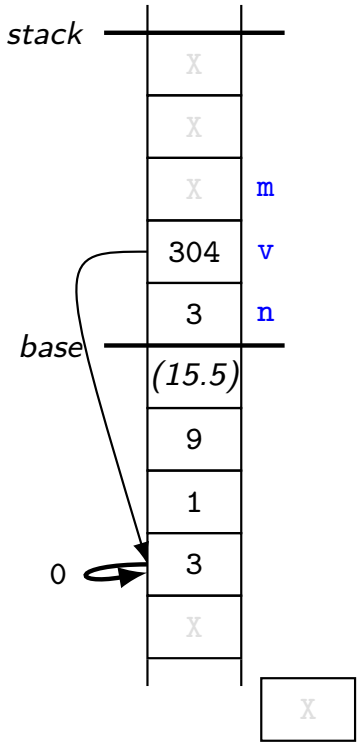


Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```

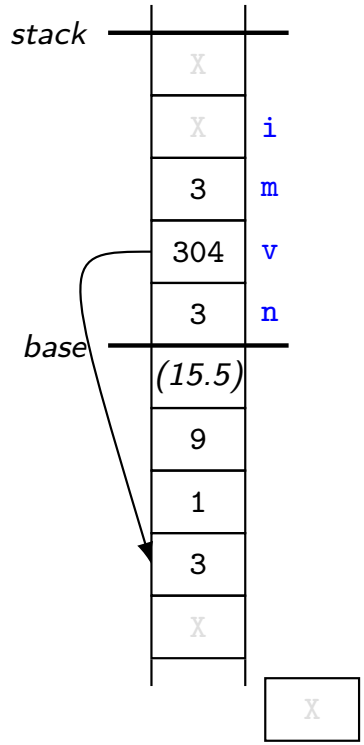


Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```

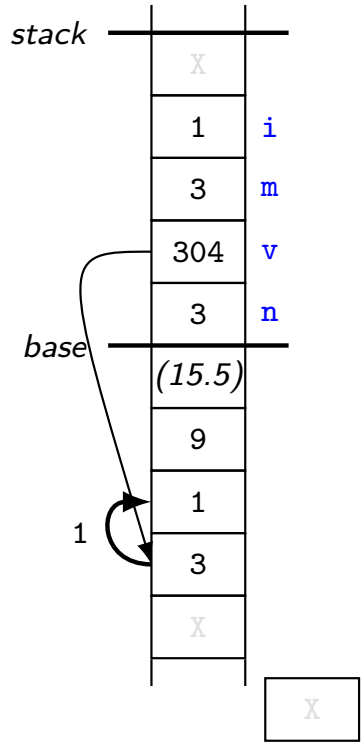


Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```

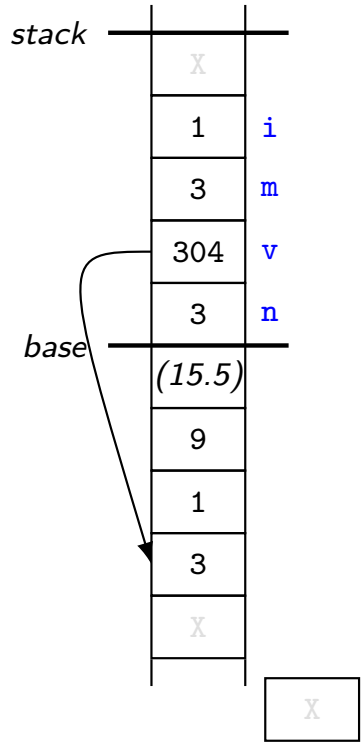


Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```

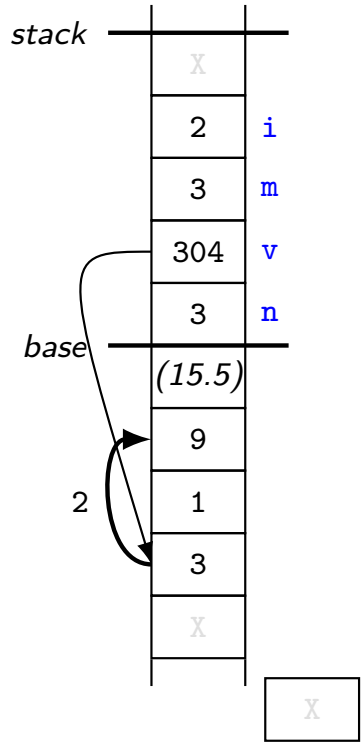


Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```

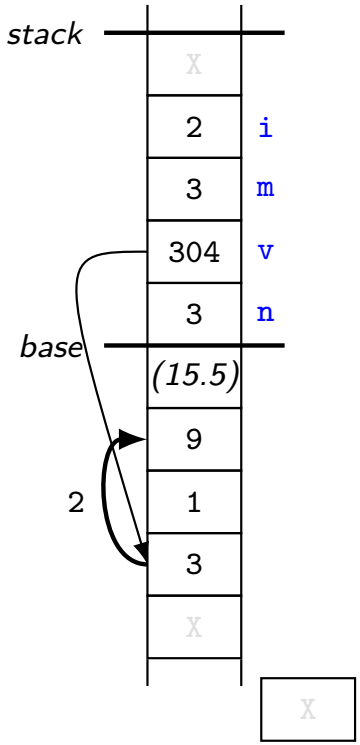


Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```

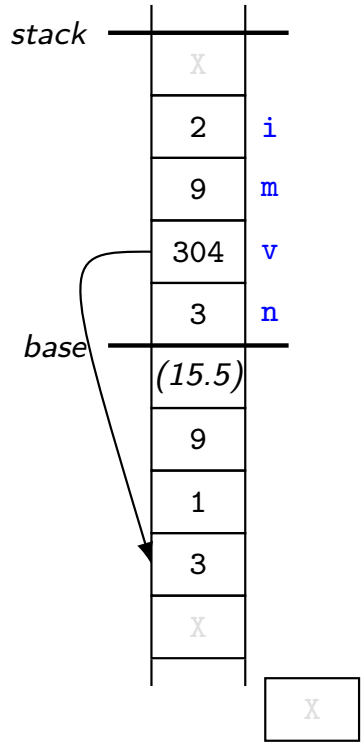


Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```

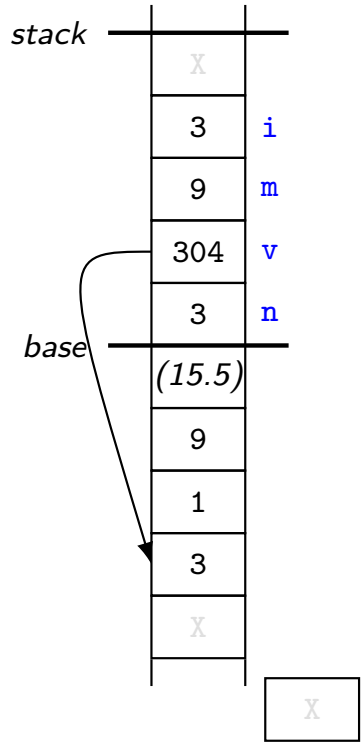


Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```

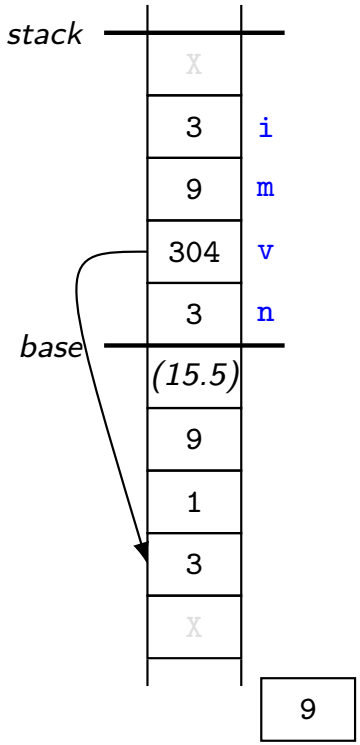


Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```

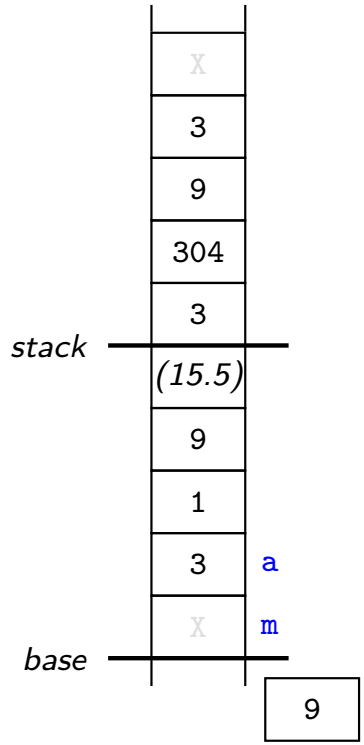


Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```

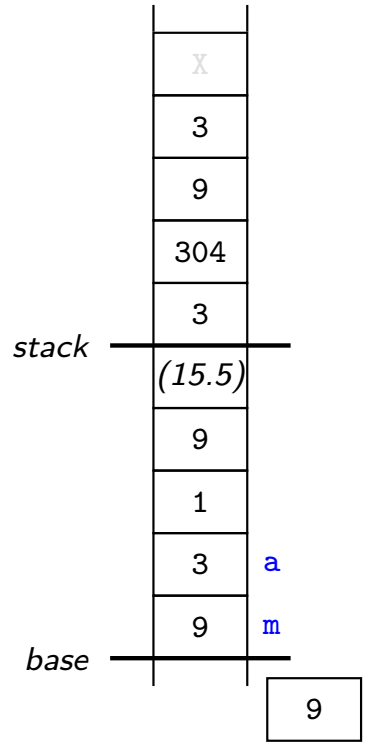


Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```

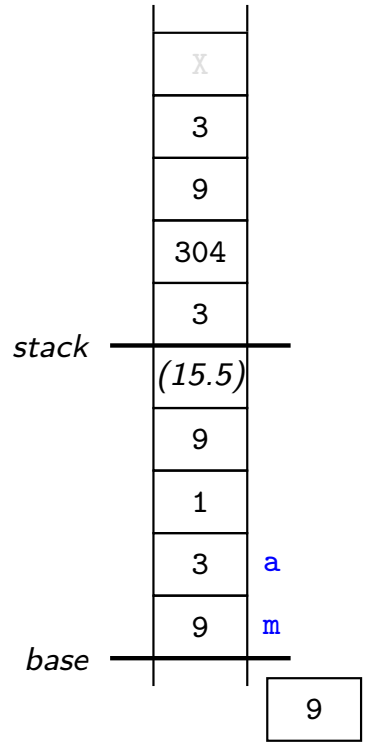


Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```



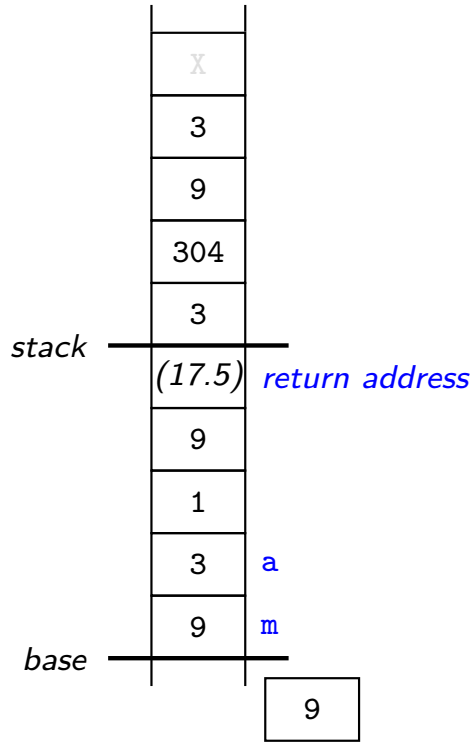

```
max(a,3) = 9
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```c
#include <stdio.h>
int max(int *v, int n)
{
    int m = v[0];
    for(int i = 1; i < n; i++) {
        if (v[i] > m) {
            m = v[i];
        }
    }
    return m;
}
int main(void)
{
    int m, a[3] = {3, 1, 9};
    m = max(a,3);
    printf("max(a,3) = %d\n", m);
    m = max(a,2);
    printf("max(a,2) = %d\n", m);
    m = max(a,4);
    printf("max(a,4) = %d\n", m);
    return 0;
}
```



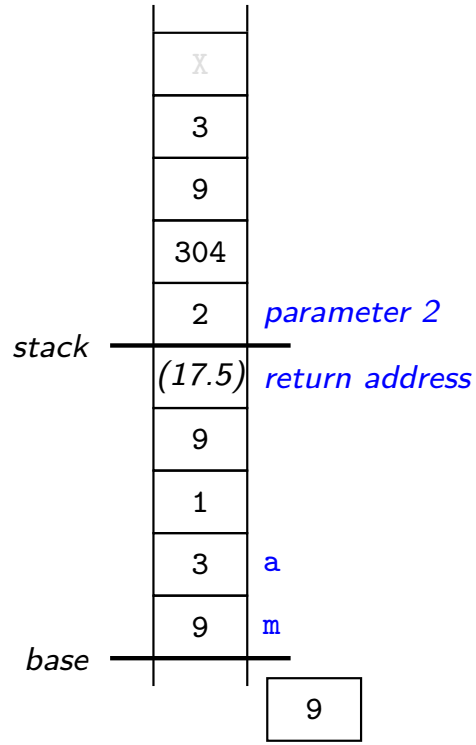

```
max(a,3) = 9
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```



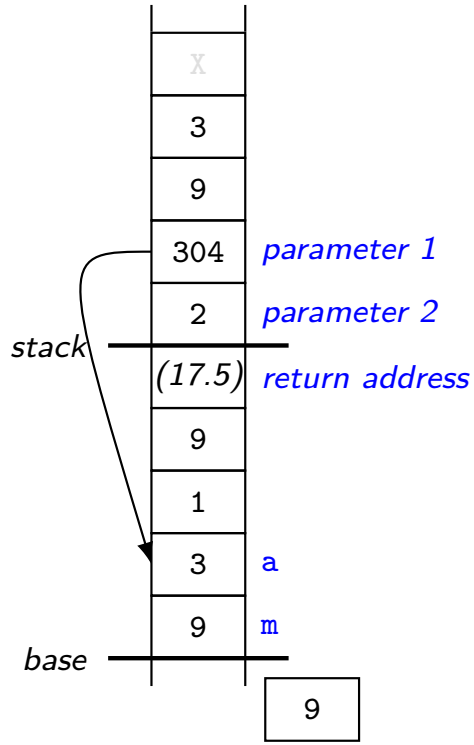

```
max(a,3) = 9
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```



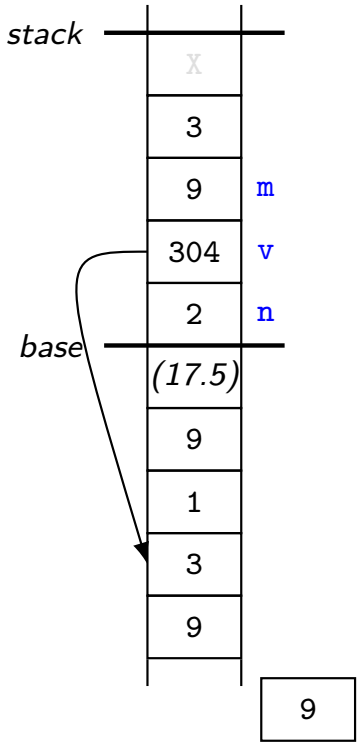

```
max(a,3) = 9
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```



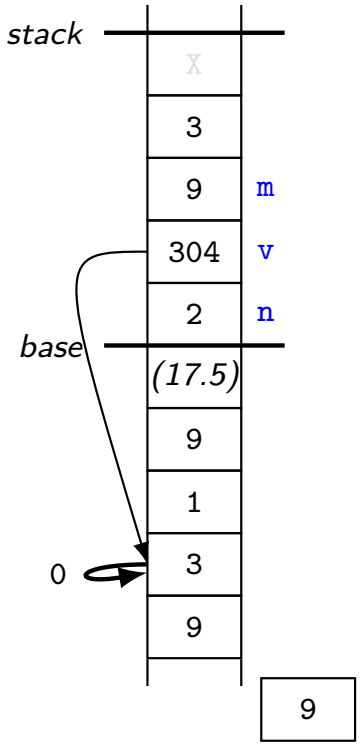

```
max(a,3) = 9
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```



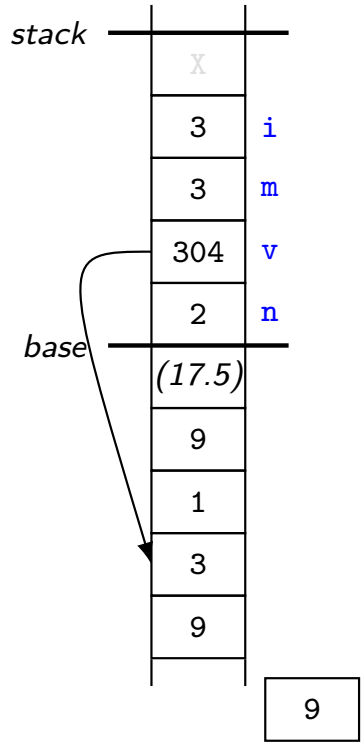

```
max(a,3) = 9
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```c
#include <stdio.h>
int max(int *v, int n)
{
    int m = v[0];
    for(int i = 1; i < n; i++) {
        if (v[i] > m) {
            m = v[i];
        }
    }
    return m;
}
int main(void)
{
    int m, a[3] = {3, 1, 9};
    m = max(a,3);
    printf("max(a,3) = %d\n", m);
    m = max(a,2);
    printf("max(a,2) = %d\n", m);
    m = max(a,4);
    printf("max(a,4) = %d\n", m);
    return 0;
}
```



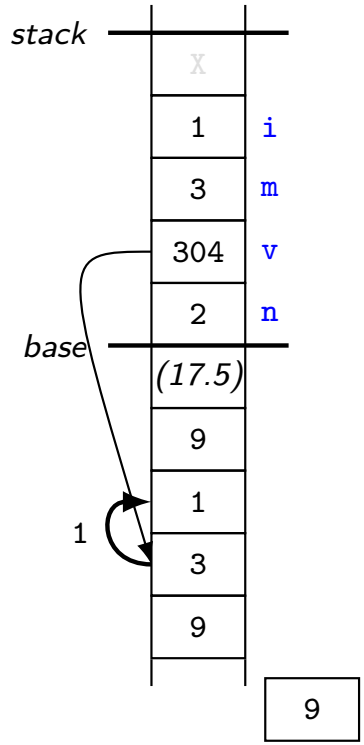

```
max(a,3) = 9
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```



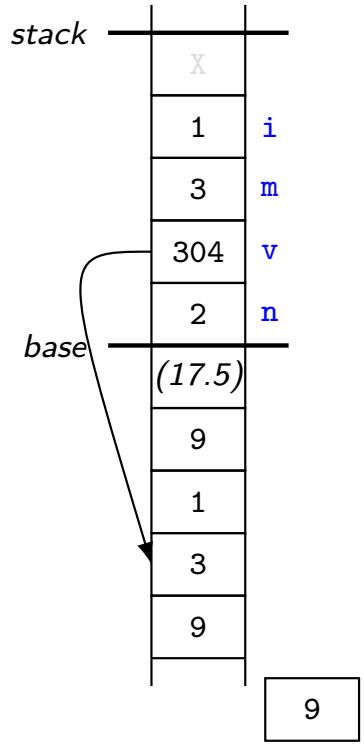

```
max(a,3) = 9
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```



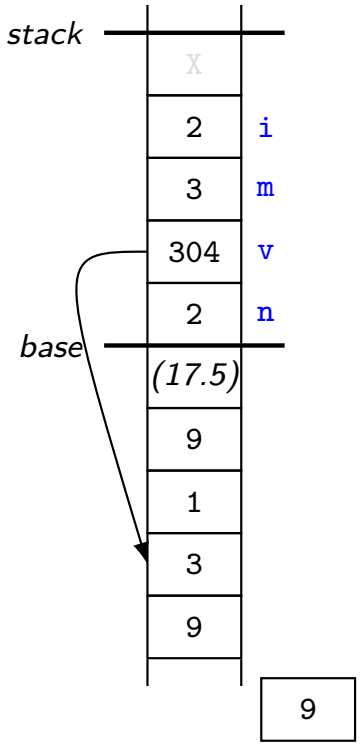

```
max(a,3) = 9
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```



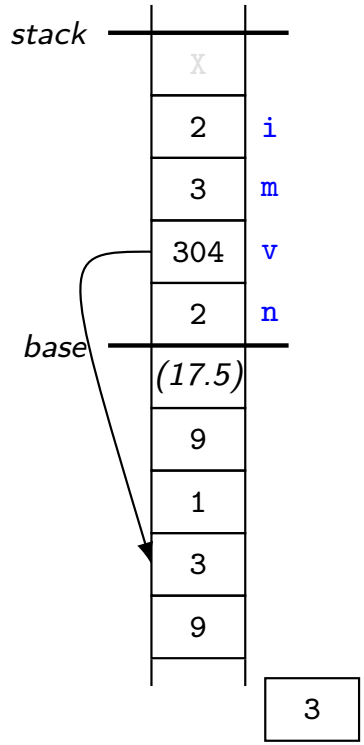

```
max(a,3) = 9
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```



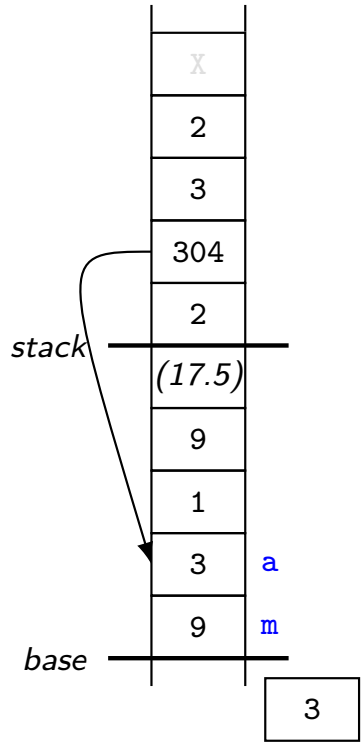

```
max(a,3) = 9
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```



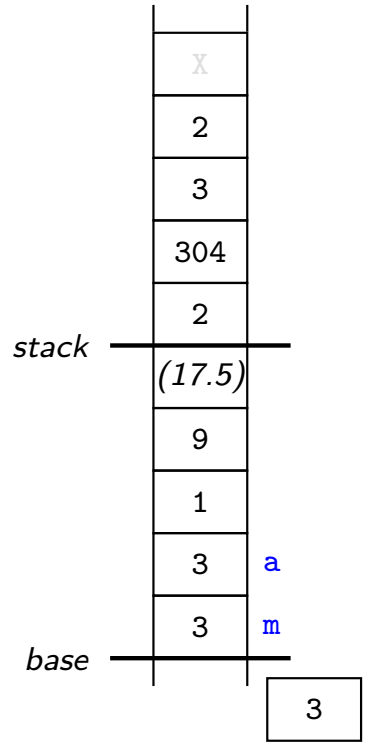

```
max(a,3) = 9
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```



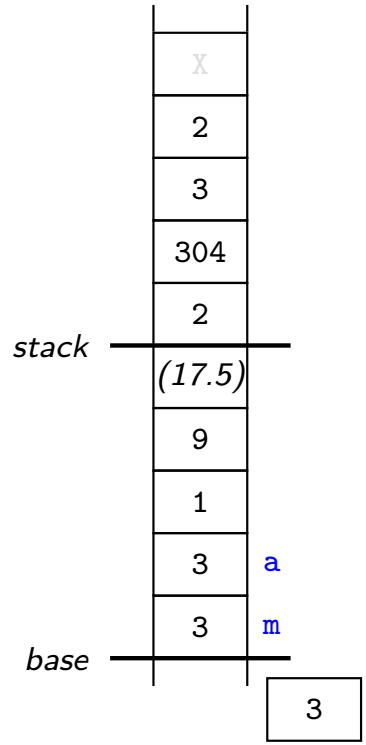

```
max(a,3) = 9
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```



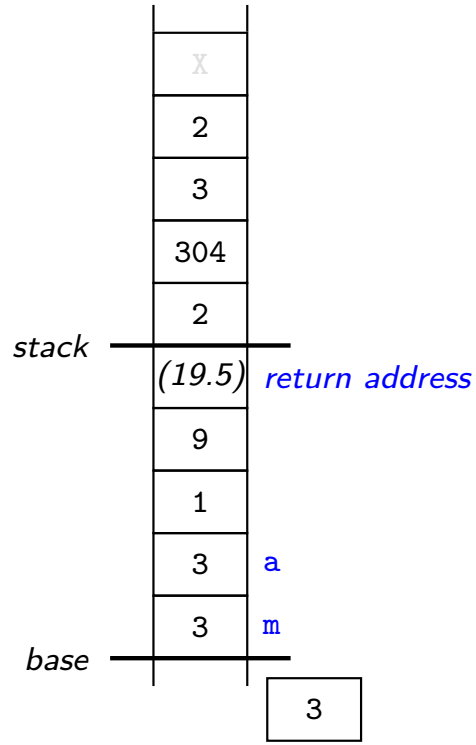

```
max(a,3) = 9
max(a,2) = 3
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```



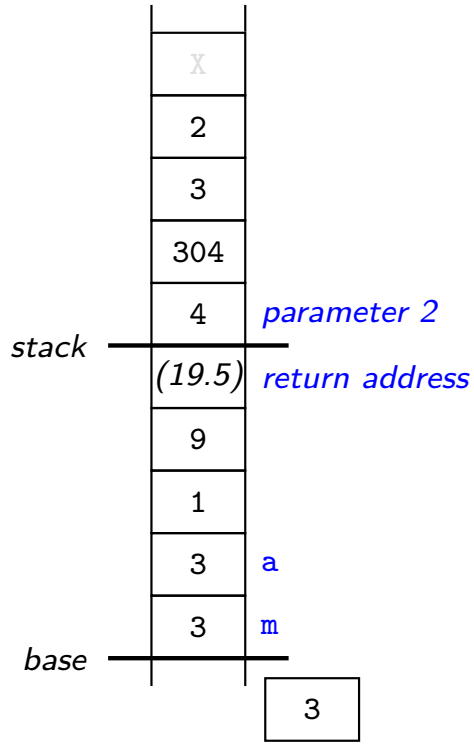

```
max(a,3) = 9
max(a,2) = 3
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```



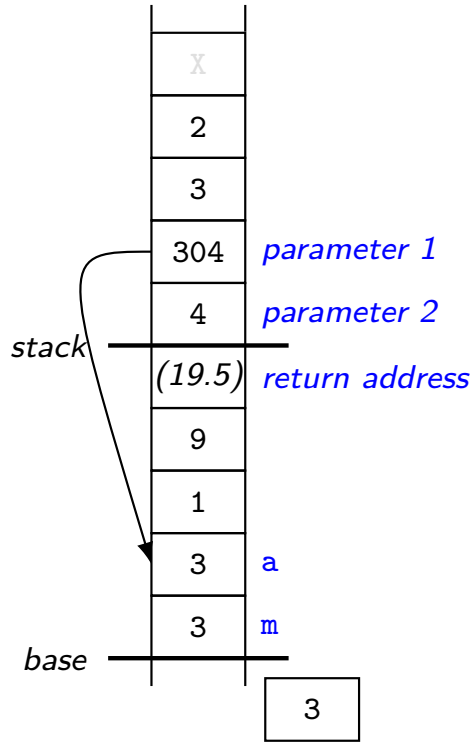

```
max(a,3) = 9
max(a,2) = 3
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```



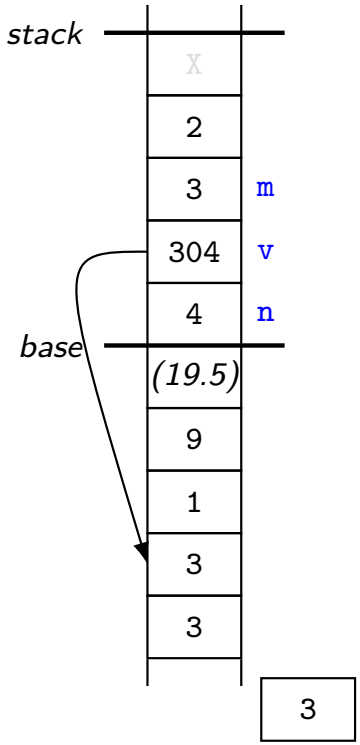

```
max(a,3) = 9
max(a,2) = 3
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```



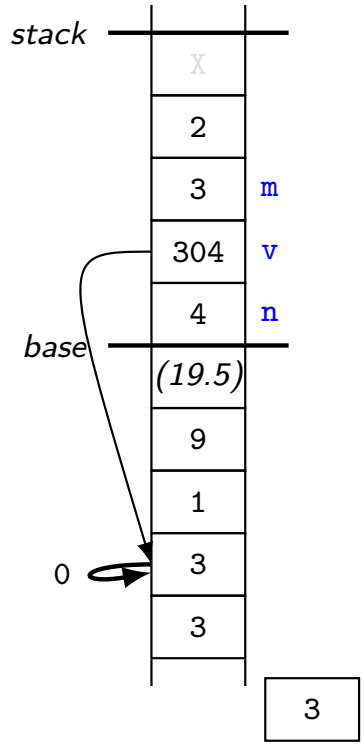

```
max(a,3) = 9
max(a,2) = 3
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```



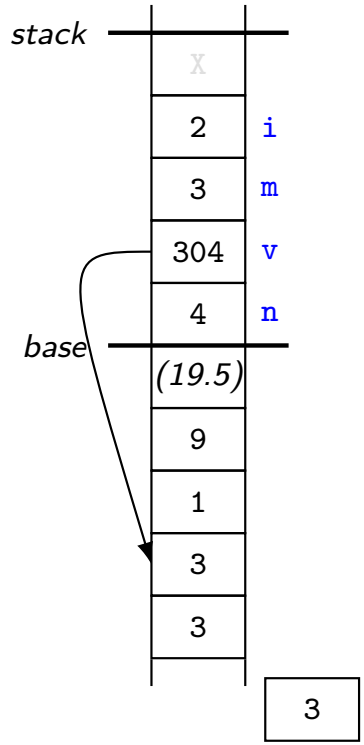

```
max(a,3) = 9
max(a,2) = 3
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```



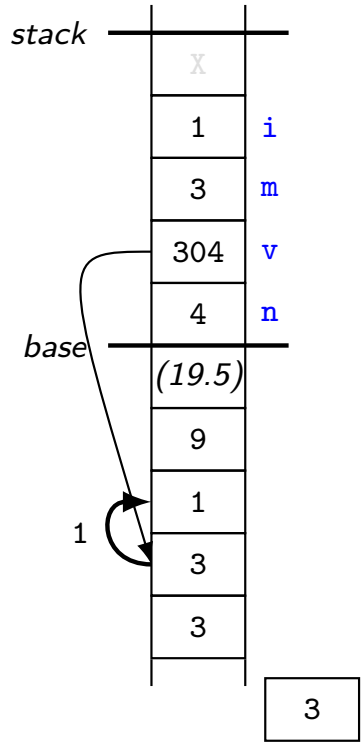

```
max(a,3) = 9
max(a,2) = 3
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```



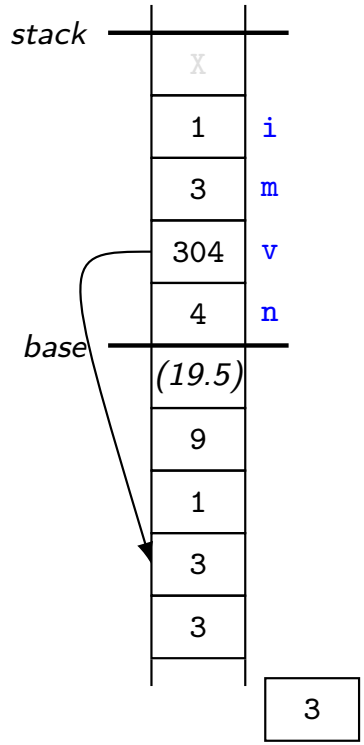

```
max(a,3) = 9
max(a,2) = 3
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```



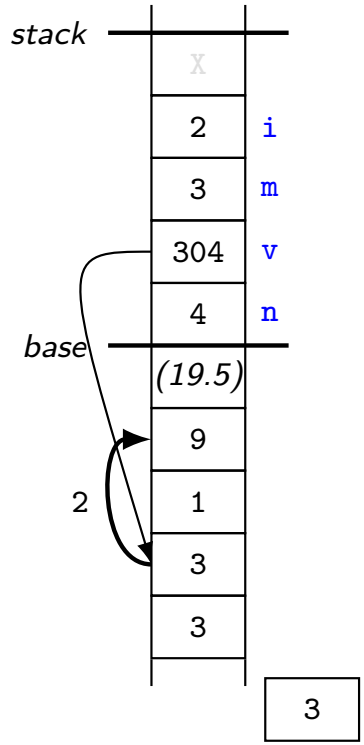

```
max(a,3) = 9
max(a,2) = 3
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```



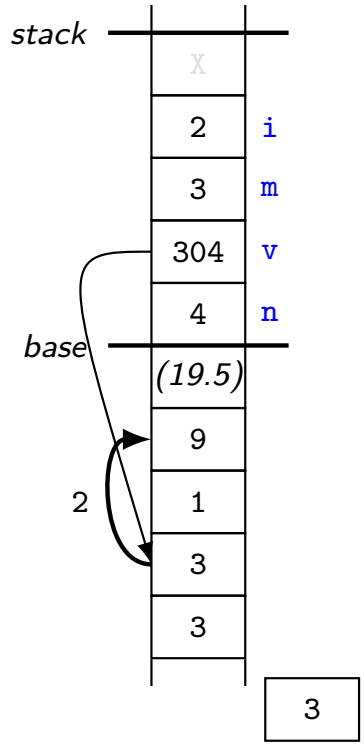

```
max(a,3) = 9
max(a,2) = 3
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```



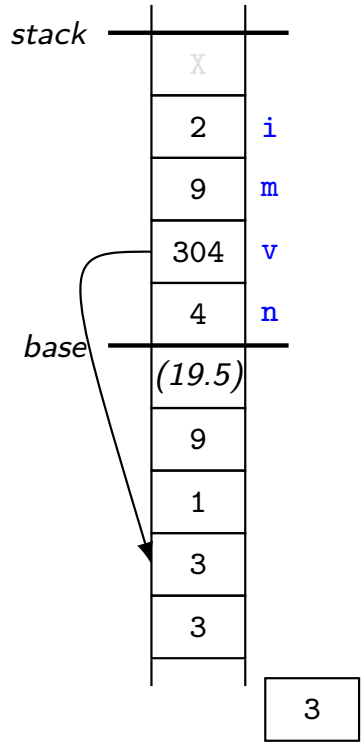

```
max(a,3) = 9
max(a,2) = 3
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```



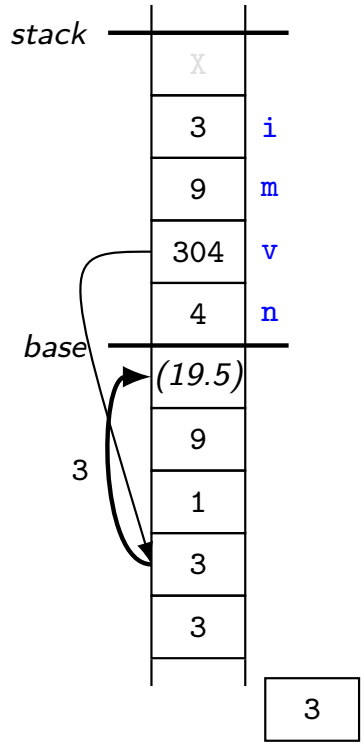

```
max(a,3) = 9
max(a,2) = 3
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```



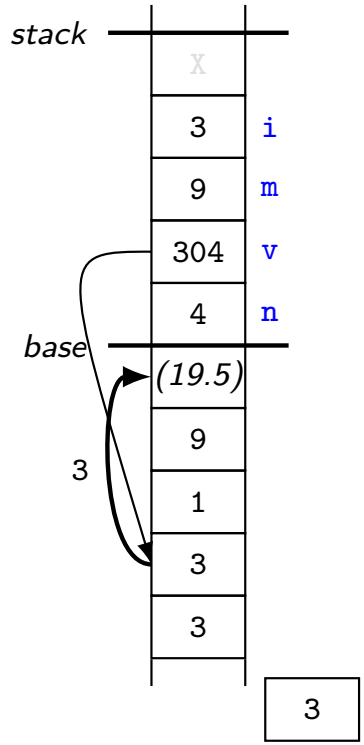

```
max(a,3) = 9
max(a,2) = 3
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```



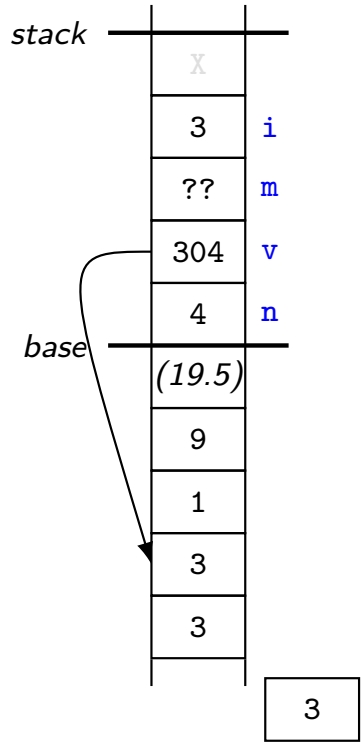

```
max(a,3) = 9
max(a,2) = 3
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```c
#include <stdio.h>
int max(int *v, int n)
{
    int m = v[0];
    for(int i = 1; i < n; i++) {
        if (v[i] > m) {
            m = v[i];
        }
    }
    return m;
}
int main(void)
{
    int m, a[3] = {3, 1, 9};
    m = max(a,3);
    printf("max(a,3) = %d\n", m);
    m = max(a,2);
    printf("max(a,2) = %d\n", m);
    m = max(a,4);
    printf("max(a,4) = %d\n", m);
    return 0;
}
```



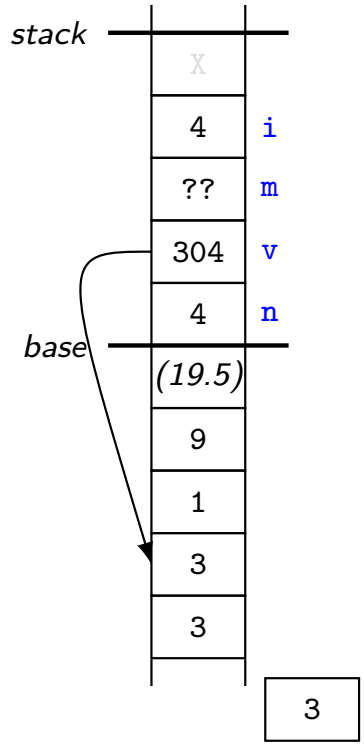

```
max(a,3) = 9
max(a,2) = 3
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```



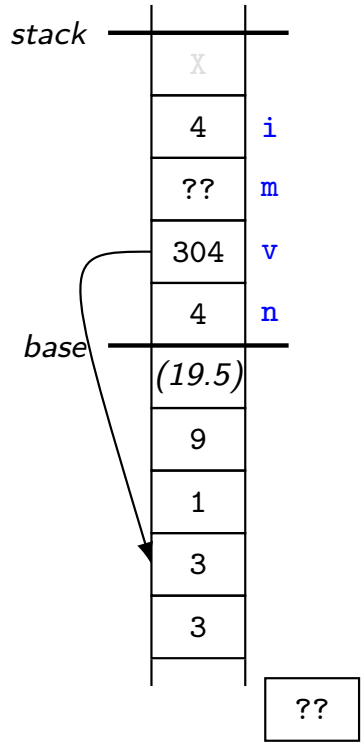

```
max(a,3) = 9
max(a,2) = 3
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```c
#include <stdio.h>
int max(int *v, int n)
{
    int m = v[0];
    for(int i = 1; i < n; i++) {
        if (v[i] > m) {
            m = v[i];
        }
    }
    return m;
}
int main(void)
{
    int m, a[3] = {3, 1, 9};
    m = max(a,3);
    printf("max(a,3) = %d\n", m);
    m = max(a,2);
    printf("max(a,2) = %d\n", m);
    m = max(a,4);
    printf("max(a,4) = %d\n", m);
    return 0;
}
```



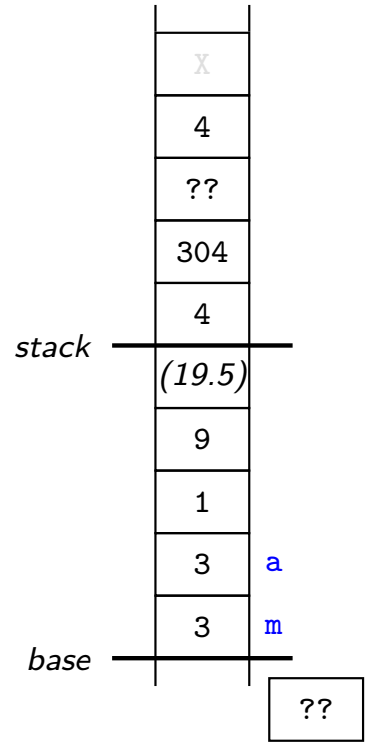

```
max(a,3) = 9
max(a,2) = 3
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```



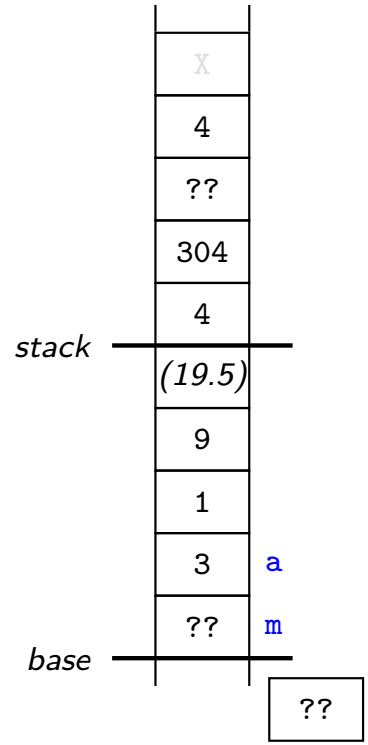

```
max(a,3) = 9
max(a,2) = 3
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```



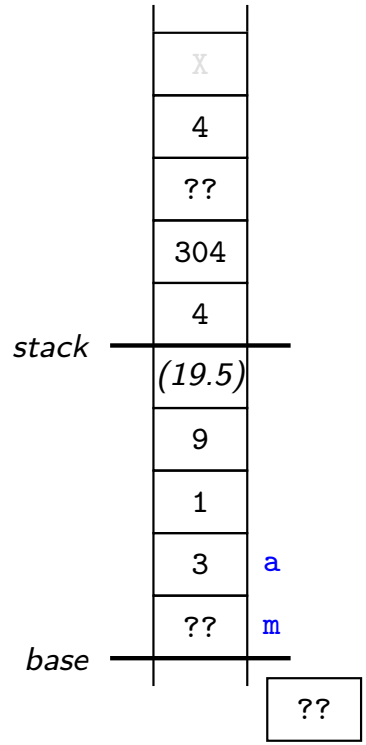

```
max(a,3) = 9
max(a,2) = 3
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```



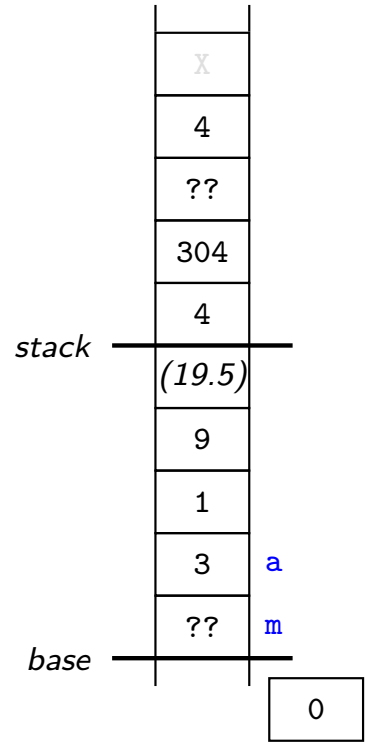

```
max(a,3) = 9
max(a,2) = 3
max(a,4) = ??
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```



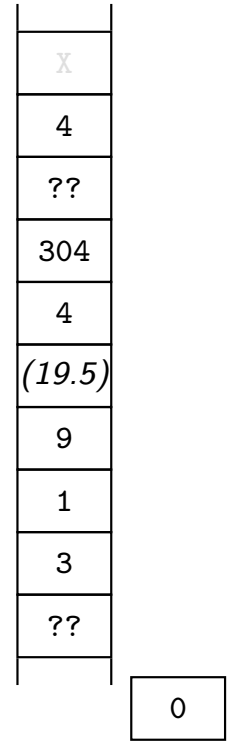

```
max(a,3) = 9
max(a,2) = 3
max(a,4) = ??
```

Fält som parametrar (3)

- Vi måste skicka med storleken på fältet till funktionen
- Annars finns risken att vi indexerar utanför fältet

code/find-max.c

```
1  #include <stdio.h>
2  int max(int *v, int n)
3  {
4      int m = v[0];
5      for(int i = 1; i < n; i++) {
6          if (v[i] > m) {
7              m = v[i];
8          }
9      }
10     return m;
11 }
12 int main(void)
13 {
14     int m, a[3] = {3, 1, 9};
15     m = max(a,3);
16     printf("max(a,3) = %d\n", m);
17     m = max(a,2);
18     printf("max(a,2) = %d\n", m);
19     m = max(a,4);
20     printf("max(a,4) = %d\n", m);
21     return 0;
22 }
```



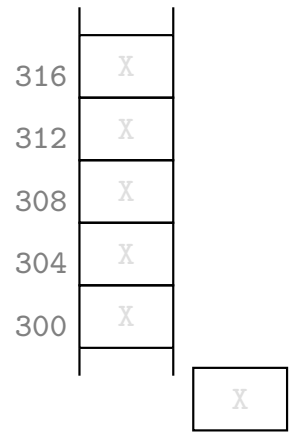

```
max(a,3) = 9
max(a,2) = 3
max(a,4) = ??
```

Pekare som returvärden (1)

- Vad händer när denna kod körs?

code/ptr-ret.c

```
1  #include <stdio.h>
2  int *dummy1(void)
3  {
4      int a[1] = {1};
5      return a;
6  }
7  int *dummy2(void)
8  {
9      int n = 2;
10     return &n;
11 }
12 int main(void)
13 {
14     int *a, *b;
15     a = dummy1();
16     printf("a = %d\n", a[0]);
17     b = dummy2();
18     printf("a = %d, b = %d\n", a[0], b[0]);
19     return 0;
20 }
```

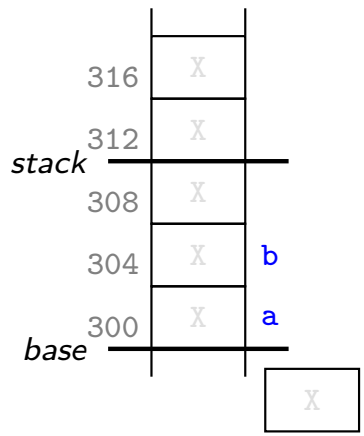


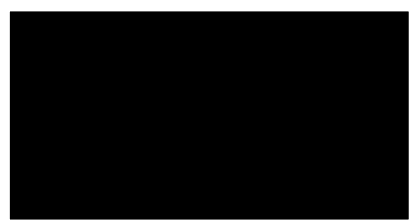
Pekare som returvärden (1)

- Vad händer när denna kod körs?

code/ptr-ret.c

```
1   #include <stdio.h>
2   int *dummy1(void)
3   {
4       int a[1] = {1};
5       return a;
6   }
7   int *dummy2(void)
8   {
9       int n = 2;
10      return &n;
11  }
12  int main(void)
13  {
14      int *a, *b;
15      a = dummy1();
16      printf("a = %d\n", a[0]);
17      b = dummy2();
18      printf("a = %d, b = %d\n", a[0], b[0]);
19      return 0;
20  }
```



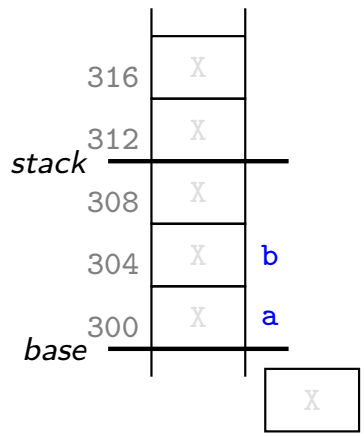


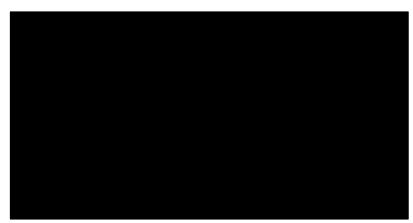
Pekare som returvärden (1)

- Vad händer när denna kod körs?

code/ptr-ret.c

```
1  #include <stdio.h>
2  int *dummy1(void)
3  {
4      int a[1] = {1};
5      return a;
6  }
7  int *dummy2(void)
8  {
9      int n = 2;
10     return &n;
11 }
12 int main(void)
13 {
14     int *a, *b;
15     a = dummy1();
16     printf("a = %d\n", a[0]);
17     b = dummy2();
18     printf("a = %d, b = %d\n", a[0], b[0]);
19     return 0;
20 }
```



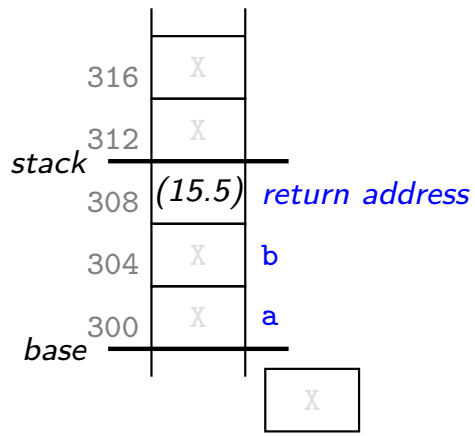


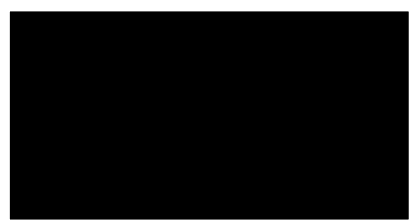
Pekare som returvärden (1)

- Vad händer när denna kod körs?

code/ptr-ret.c

```
1  #include <stdio.h>
2  int *dummy1(void)
3  {
4      int a[1] = {1};
5      return a;
6  }
7  int *dummy2(void)
8  {
9      int n = 2;
10     return &n;
11 }
12 int main(void)
13 {
14     int *a, *b;
15     a = dummy1();
16     printf("a = %d\n", a[0]);
17     b = dummy2();
18     printf("a = %d, b = %d\n", a[0], b[0]);
19     return 0;
20 }
```



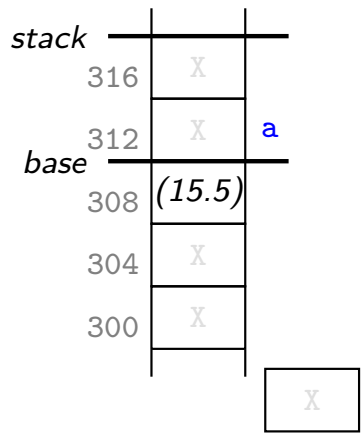


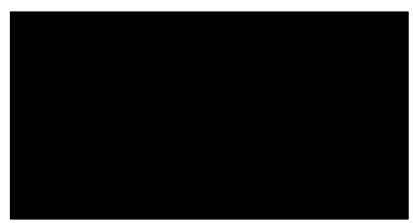
Pekare som returvärden (1)

- Vad händer när denna kod körs?

code/ptr-ret.c

```
1  #include <stdio.h>
2  int *dummy1(void)
3  {
4      int a[1] = {1};
5      return a;
6  }
7  int *dummy2(void)
8  {
9      int n = 2;
10     return &n;
11 }
12 int main(void)
13 {
14     int *a, *b;
15     a = dummy1();
16     printf("a = %d\n", a[0]);
17     b = dummy2();
18     printf("a = %d, b = %d\n", a[0], b[0]);
19     return 0;
20 }
```



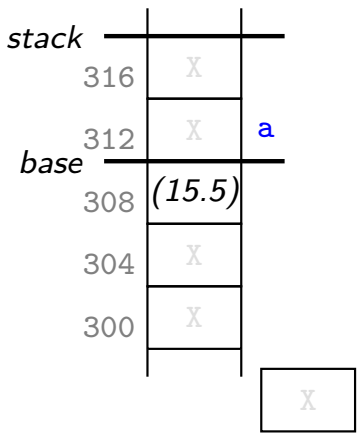


Pekare som returvärden (1)

- Vad händer när denna kod körs?

code/ptr-ret.c

```
1  #include <stdio.h>
2  int *dummy1(void)
3  {
4      int a[1] = {1};
5      return a;
6  }
7  int *dummy2(void)
8  {
9      int n = 2;
10     return &n;
11 }
12 int main(void)
13 {
14     int *a, *b;
15     a = dummy1();
16     printf("a = %d\n", a[0]);
17     b = dummy2();
18     printf("a = %d, b = %d\n", a[0], b[0]);
19     return 0;
20 }
```

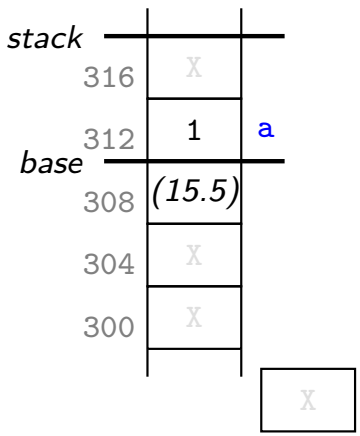


Pekare som returvärden (1)

- Vad händer när denna kod körs?

code/ptr-ret.c

```
1  #include <stdio.h>
2  int *dummy1(void)
3  {
4      int a[1] = {1};
5      return a;
6  }
7  int *dummy2(void)
8  {
9      int n = 2;
10     return &n;
11 }
12 int main(void)
13 {
14     int *a, *b;
15     a = dummy1();
16     printf("a = %d\n", a[0]);
17     b = dummy2();
18     printf("a = %d, b = %d\n", a[0], b[0]);
19     return 0;
20 }
```

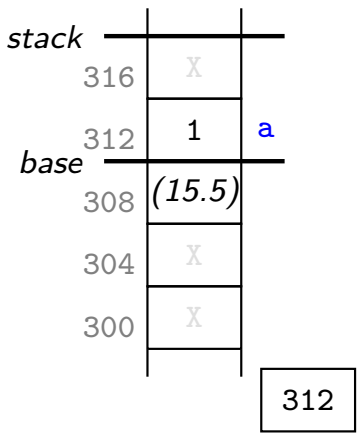


Pekare som returvärden (1)

- Vad händer när denna kod körs?

code/ptr-ret.c

```
1  #include <stdio.h>
2  int *dummy1(void)
3  {
4      int a[1] = {1};
5      return a;
6  }
7  int *dummy2(void)
8  {
9      int n = 2;
10     return &n;
11 }
12 int main(void)
13 {
14     int *a, *b;
15     a = dummy1();
16     printf("a = %d\n", a[0]);
17     b = dummy2();
18     printf("a = %d, b = %d\n", a[0], b[0]);
19     return 0;
20 }
```

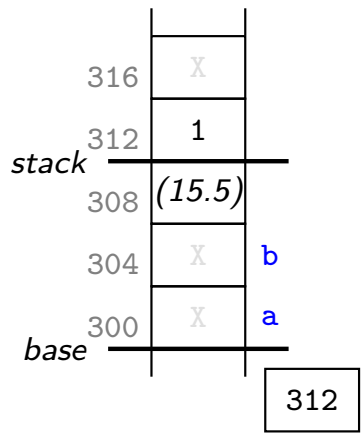


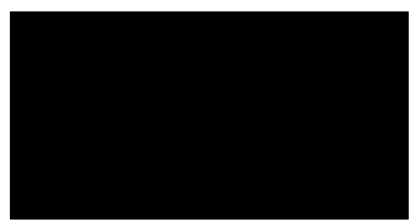
Pekare som returvärden (1)

- Vad händer när denna kod körs?

code/ptr-ret.c

```
1  #include <stdio.h>
2  int *dummy1(void)
3  {
4      int a[1] = {1};
5      return a;
6  }
7  int *dummy2(void)
8  {
9      int n = 2;
10     return &n;
11 }
12 int main(void)
13 {
14     int *a, *b;
15     a = dummy1();
16     printf("a = %d\n", a[0]);
17     b = dummy2();
18     printf("a = %d, b = %d\n", a[0], b[0]);
19     return 0;
20 }
```



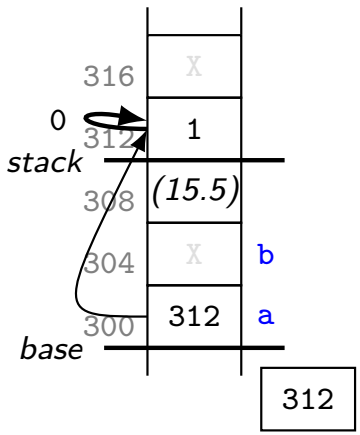


Pekare som returvärden (1)

- Vad händer när denna kod körs?

code/ptr-ret.c

```
1  #include <stdio.h>
2  int *dummy1(void)
3  {
4      int a[1] = {1};
5      return a;
6  }
7  int *dummy2(void)
8  {
9      int n = 2;
10     return &n;
11 }
12 int main(void)
13 {
14     int *a, *b;
15     a = dummy1();
16     printf("a = %d\n", a[0]);
17     b = dummy2();
18     printf("a = %d, b = %d\n", a[0], b[0]);
19     return 0;
20 }
```

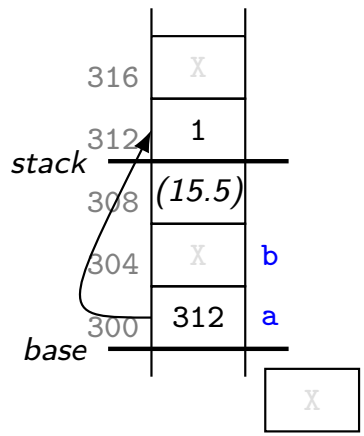


Pekare som returvärden (1)

- Vad händer när denna kod körs?

code/ptr-ret.c

```
1  #include <stdio.h>
2  int *dummy1(void)
3  {
4      int a[1] = {1};
5      return a;
6  }
7  int *dummy2(void)
8  {
9      int n = 2;
10     return &n;
11 }
12 int main(void)
13 {
14     int *a, *b;
15     a = dummy1();
16     printf("a = %d\n", a[0]);
17     b = dummy2();
18     printf("a = %d, b = %d\n", a[0], b[0]);
19     return 0;
20 }
```



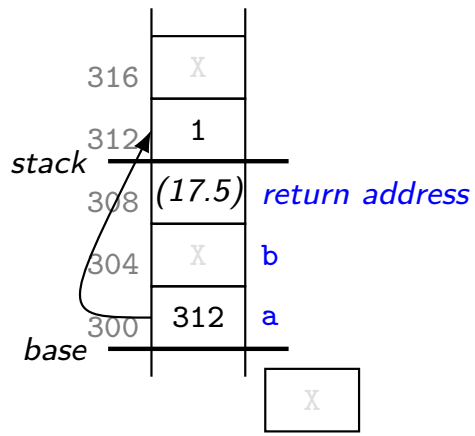

```
a = 1
```

Pekare som returvärden (1)

- Vad händer när denna kod körs?

code/ptr-ret.c

```
1  #include <stdio.h>
2  int *dummy1(void)
3  {
4      int a[1] = {1};
5      return a;
6  }
7  int *dummy2(void)
8  {
9      int n = 2;
10     return &n;
11 }
12 int main(void)
13 {
14     int *a, *b;
15     a = dummy1();
16     printf("a = %d\n", a[0]);
17     b = dummy2();
18     printf("a = %d, b = %d\n", a[0], b[0]);
19     return 0;
20 }
```



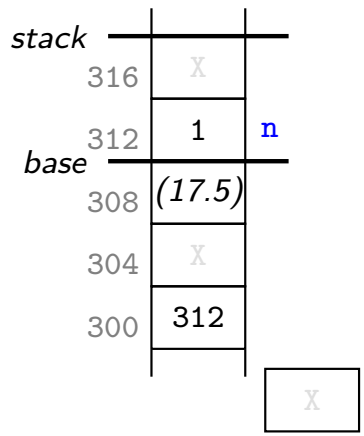

```
a = 1
```

Pekare som returvärden (1)

- Vad händer när denna kod körs?

code/ptr-ret.c

```
1  #include <stdio.h>
2  int *dummy1(void)
3  {
4      int a[1] = {1};
5      return a;
6  }
7  int *dummy2(void)
8  {
9      int n = 2;
10     return &n;
11 }
12 int main(void)
13 {
14     int *a, *b;
15     a = dummy1();
16     printf("a = %d\n", a[0]);
17     b = dummy2();
18     printf("a = %d, b = %d\n", a[0], b[0]);
19     return 0;
20 }
```



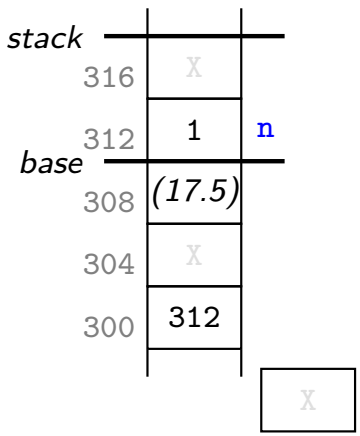

```
a = 1
```

Pekare som returvärden (1)

- Vad händer när denna kod körs?

code/ptr-ret.c

```
1  #include <stdio.h>
2  int *dummy1(void)
3  {
4      int a[1] = {1};
5      return a;
6  }
7  int *dummy2(void)
8  {
9      int n = 2;
10     return &n;
11 }
12 int main(void)
13 {
14     int *a, *b;
15     a = dummy1();
16     printf("a = %d\n", a[0]);
17     b = dummy2();
18     printf("a = %d, b = %d\n", a[0], b[0]);
19     return 0;
20 }
```



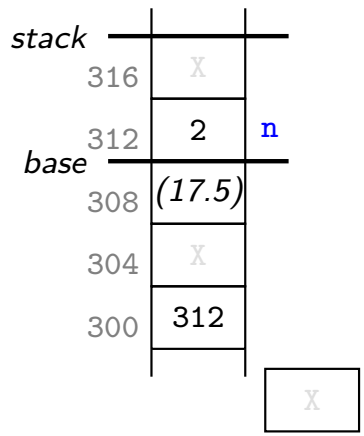

```
a = 1
```

Pekare som returvärden (1)

- Vad händer när denna kod körs?

code/ptr-ret.c

```
1  #include <stdio.h>
2  int *dummy1(void)
3  {
4      int a[1] = {1};
5      return a;
6  }
7  int *dummy2(void)
8  {
9      int n = 2;
10     return &n;
11 }
12 int main(void)
13 {
14     int *a, *b;
15     a = dummy1();
16     printf("a = %d\n", a[0]);
17     b = dummy2();
18     printf("a = %d, b = %d\n", a[0], b[0]);
19     return 0;
20 }
```



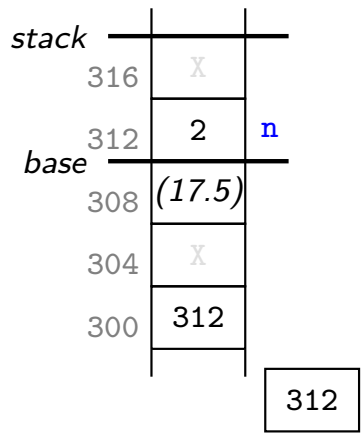

```
a = 1
```

Pekare som returvärden (1)

- Vad händer när denna kod körs?

code/ptr-ret.c

```
1  #include <stdio.h>
2  int *dummy1(void)
3  {
4      int a[1] = {1};
5      return a;
6  }
7  int *dummy2(void)
8  {
9      int n = 2;
10     return &n;
11 }
12 int main(void)
13 {
14     int *a, *b;
15     a = dummy1();
16     printf("a = %d\n", a[0]);
17     b = dummy2();
18     printf("a = %d, b = %d\n", a[0], b[0]);
19     return 0;
20 }
```



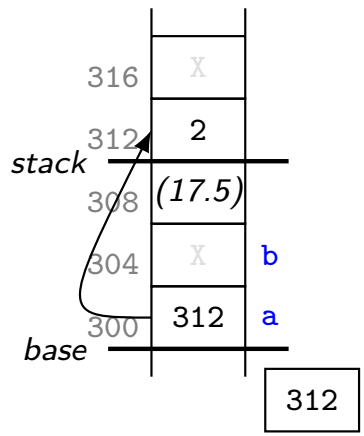

```
a = 1
```

Pekare som returvärden (1)

- Vad händer när denna kod körs?

code/ptr-ret.c

```
1  #include <stdio.h>
2  int *dummy1(void)
3  {
4      int a[1] = {1};
5      return a;
6  }
7  int *dummy2(void)
8  {
9      int n = 2;
10     return &n;
11 }
12 int main(void)
13 {
14     int *a, *b;
15     a = dummy1();
16     printf("a = %d\n", a[0]);
17     b = dummy2();
18     printf("a = %d, b = %d\n", a[0], b[0]);
19     return 0;
20 }
```



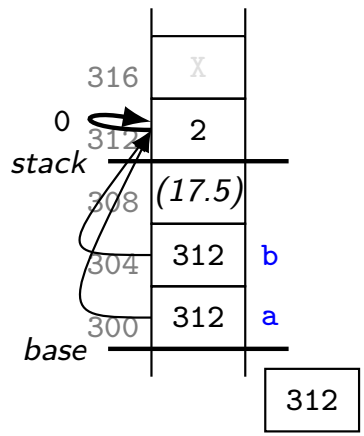

```
a = 1
```

Pekare som returvärden (1)

- Vad händer när denna kod körs?

code/ptr-ret.c

```
1  #include <stdio.h>
2  int *dummy1(void)
3  {
4      int a[1] = {1};
5      return a;
6  }
7  int *dummy2(void)
8  {
9      int n = 2;
10     return &n;
11 }
12 int main(void)
13 {
14     int *a, *b;
15     a = dummy1();
16     printf("a = %d\n", a[0]);
17     b = dummy2();
18     printf("a = %d, b = %d\n", a[0], b[0]);
19     return 0;
20 }
```



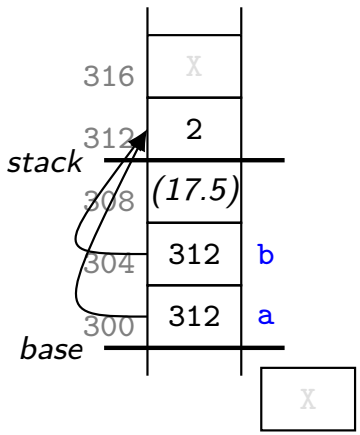

```
a = 1
```

Pekare som returvärden (1)

- Vad händer när denna kod körs?

code/ptr-ret.c

```
1  #include <stdio.h>
2  int *dummy1(void)
3  {
4      int a[1] = {1};
5      return a;
6  }
7  int *dummy2(void)
8  {
9      int n = 2;
10     return &n;
11 }
12 int main(void)
13 {
14     int *a, *b;
15     a = dummy1();
16     printf("a = %d\n", a[0]);
17     b = dummy2();
18     printf("a = %d, b = %d\n", a[0], b[0]);
19     return 0;
20 }
```



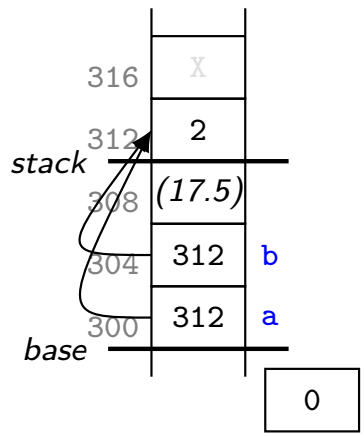

```
a = 1
a = 2, b = 2
```

Pekare som returvärden (1)

- Vad händer när denna kod körs?

code/ptr-ret.c

```c
#include <stdio.h>
int *dummy1(void)
{
    int a[1] = {1};
    return a;
}
int *dummy2(void)
{
    int n = 2;
    return &n;
}
int main(void)
{
    int *a, *b;
    a = dummy1();
    printf("a = %d\n", a[0]);
    b = dummy2();
    printf("a = %d, b = %d\n", a[0], b[0]);
    return 0;
}
```



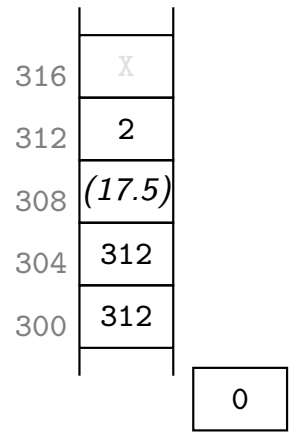

```
a = 1
a = 2, b = 2
```

Pekare som returvärden (1)

- Vad händer när denna kod körs?

code/ptr-ret.c

```c
#include <stdio.h>
int *dummy1(void)
{
    int a[1] = {1};
    return a;
}
int *dummy2(void)
{
    int n = 2;
    return &n;
}
int main(void)
{
    int *a, *b;
    a = dummy1();
    printf("a = %d\n", a[0]);
    b = dummy2();
    printf("a = %d, b = %d\n", a[0], b[0]);
    return 0;
}
```




```
a = 1
a = 2, b = 2
```

Pekare som returvärden (2)

- Funktioner som returnerar värden gör det genom att kopiera värdet
- Om returvärdet är en pekare (adress) kan vi få en pekare som pekar till fel del av stacken
- När funktionen returnerar frigörs minnet som funktionen använt till lokala variabler
- Nästa funktion som anropas får tillgång till samma minne och kan alltså ändra i det, vilket kan få oönskade sidoeffekter
- Vi skall alltså aldrig returnera en adress till en lokal variabel!
- Vill vi använda funktioner till att allokera minne till fält så finns en speciell teknik
  - Dynamisk minnesallokering (senare kurser)

Tips

- Det är lätt att det blir typfel vid aktuella och formella parametrar
  - Kompilatorn ger ofta varningar
  - Tips: Ta de enkla program som vi tittat på i dag, skriv in och experimentera med dem
- Börja med att få något litet att fungera
  - Kan vara kod från föreläsning, övningsuppgift, bok, etc. eller från tom fil
  - Modifiera, lägg till
  - Kompilera och testa i varje steg