Programmering i C

Lektion 4

18. september 2009

Pointers

Husk: “En variabel er en navngiven plads i computerens lager.”

En pointer er en “pegepind” der *peger* på denne plads.

Husk: “En variabel er en navngiven plads i computerens lager.”

En pointer er en “pegepind” der *peger* på denne plads.

Declaring a pointer:

```
int* ptr_example; // Declares a pointer to an int.
```

Husk: “En variabel er en navngiven plads i computerens lager.”

En pointer er en “pegepind” der *peger* på denne plads.

Declaring a pointer:

```
int* ptr_example; // Declares a pointer to an int.
```

Getting the address of a variable:

```
int my_int = 3;
ptr_example = &my_int. // makes ptr_example point to
                       // the address of my_int.
```

Husk: “En variabel er en navngiven plads i computerens lager.”

En pointer er en “pegepind” der *peger* på denne plads.

Declaring a pointer:

```
int* ptr_example; // Declares a pointer to an int.
```

Getting the address of a variable:

```
int my_int = 3;
ptr_example = &my_int. // makes ptr_example point to
                       // the address of my_int.
```

Dereferencing:

```
*ptr_example = 2; // Sets the value of the data
                  // pointed to by ptr_example.
```

- &j betegner *adressen* af variablen j
- ∗pti betegner den *værdi*, som pti *peger på*

$\Rightarrow$ ∗&i er det samme som i (og &∗pti er det samme som pti)

- ∗ = dereference, & = reference

Eksempel:

```
int* ptr_example; // Declares a pointer to an int.
int *ptr_example; // Declares a pointer to an int.
int* ptr2, ptr3;

int main (void) {
    ptr3 = 5;
    ptr2 = 5; // gives warning
}
```

- `*s should be sticky.`

Eksempel:

```
int* ptr_example; // Declares a pointer to an int.
int *ptr_example; // Declares a pointer to an int.
int *ptr2, ptr3;

int main (void) {
    ptr3 = 5;
    ptr2 = (int*) 5; // using a cast
                     // still not a good idea.
}
```

- &j betegner *adressen* af variablen j
- ∗pti betegner den *værdi*, som pti *peger på*

$\Rightarrow$ ∗&i er det samme som i (og &∗pti er det samme som pti)

- ∗ = dereference, & = reference

Problem: Funktioner i C kan ikke ændre på deres parametre (og give ændringer tilbage til hovedprogrammet) – værdiparametre.

Problem: Funktioner i C kan ikke ændre på deres parametre (og give ændringer tilbage til hovedprogrammet) – værdiparametre.

Løsning: Kald funktionen med pointers som parametre:

Problem: Funktioner i C kan ikke ændre på deres parametre (og give ændringer tilbage til hovedprogrammet) – værdiparametre.

Løsning: Kald funktionen med pointers som parametre:
Eksempel: en funktion der bytter om på to heltal:

```
void swap(int *x, int *y) {
  int tmp;
  tmp = *x;
  *x = *y;
  *y = tmp;
}
```

Bemærk at swap ikke laver om på de to pointers; kun på de *værdier* de peger på! [swap.c]

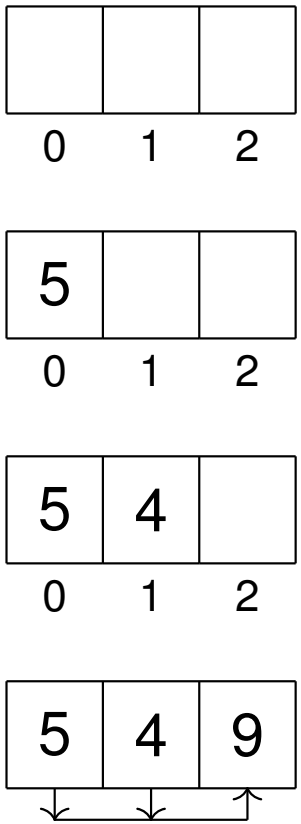
Eksempel: en funktion der bytter om på to heltal:

```
int main(void) {
  int a = 3, b = 7;

  printf("Before: %d %d\n", a, b);
  swap(&a, &b);
  printf("After:  %d %d\n", a, b);

  return 0;
}

void swap(int *x, int *y) {
  int tmp;
  tmp = *x;
  *x = *y;
  *y = tmp;
}
```

Arrays

Et array er en tabel af variable *af samme type* der kan tilgås via deres indeks.

```
int tal[3];

tal[0]=5;

tal[1]=4;

tal[2]=tal[0]+tal[1];
```



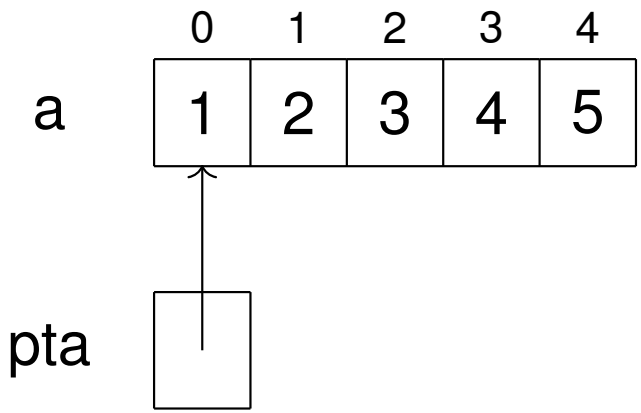
- et array skal deklareres med angivelse af *type*, og helst også *størrelse*: *type* a[*N*]
- laveste indeks er 0, højeste er $N - 1$
- indgangene lagres *umiddelbart efter hinanden*

$\Rightarrow$ &a[k] == &a[0] + k∗sizeof(*type*)

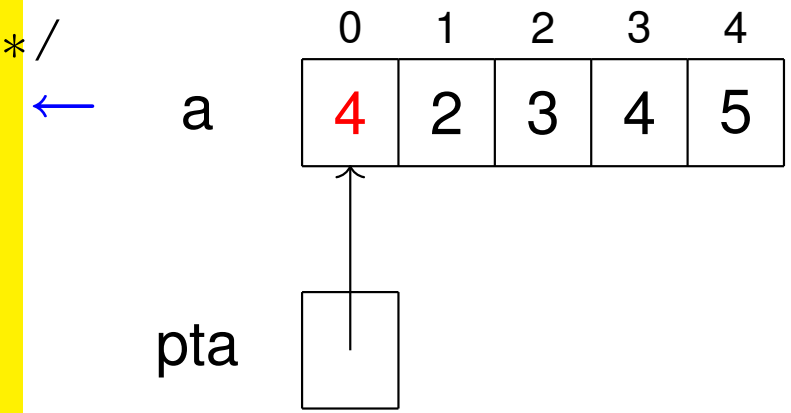
I C er et array det samme som en konstant pointer til dets første indgang:

```c
#include <stdio.h>

int main (void) { /* array-pt.c */
  int a[3], i;

  *a = 5;
  *(a + 1) = 4;
  *(a + 2) = *a + *(a + 1);

  for (i = 0; i < 3; i++) {
    printf( "%d: %d\n", i, a[i]);
  }

  return 0;
}
```

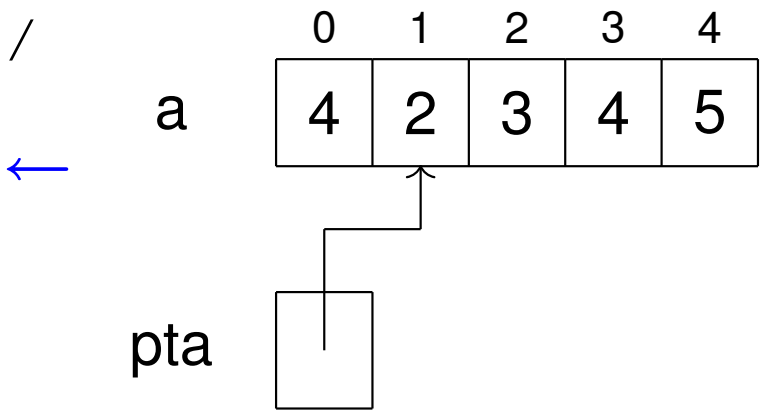
```c
#include <stdio.h>

/* array-pt-2.c */
int main( void) {
  int a[5]= {1, 2, 3, 4, 5};
  int *pta, i;

   pta = a; /* or, pta= &a[0]; */
  *pta = 4;
   pta++;
  *pta = *(pta - 1) * 2;
   pta += 3;
  (*pta)++;
  printf("index: %d\n", pta - a);

  for(i = 0; i < 5; i++) {
    printf("a[%d]: %d\n",i,a[i]);
  }
  return 0;
}
```



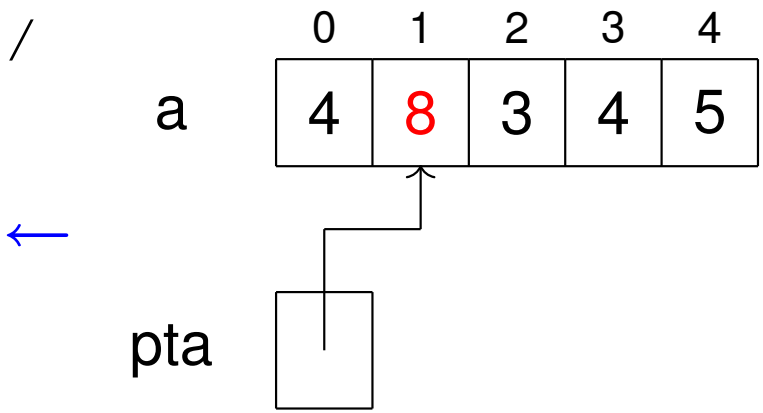
```c
#include <stdio.h>

/* array-pt-2.c */
int main( void) {
  int a[5]= {1, 2, 3, 4, 5};
  int *pta, i;

   pta = a; /* or, pta= &a[0]; */
  *pta = 4;
   pta++;
  *pta = *(pta - 1) * 2;
   pta += 3;
  (*pta)++;
  printf("index: %d\n", pta - a);

  for(i = 0; i < 5; i++) {
    printf("a[%d]: %d\n",i,a[i]);
  }
  return 0;
}
```



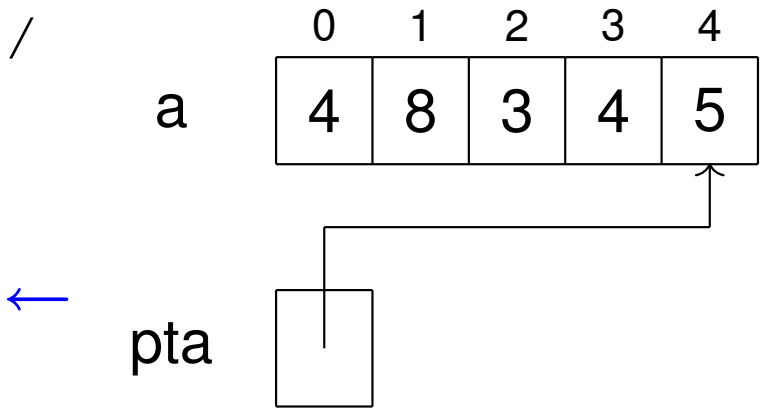
```c
#include <stdio.h>

/* array-pt-2.c */
int main( void) {
  int a[5]= {1, 2, 3, 4, 5};
  int *pta, i;

   pta = a; /* or, pta= &a[0]; */
  *pta = 4;
   pta++;
  *pta = *(pta - 1) * 2;
   pta += 3;
  (*pta)++;
  printf("index: %d\n", pta - a);

  for(i = 0; i < 5; i++) {
    printf("a[%d]: %d\n",i,a[i]);
  }
  return 0;
}
```



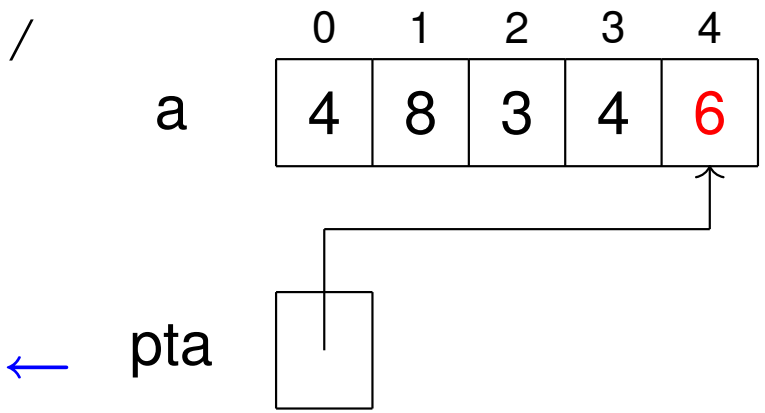
```c
#include <stdio.h>

/* array-pt-2.c */
int main( void) {
  int a[5]= {1, 2, 3, 4, 5};
  int *pta, i;

   pta = a; /* or, pta= &a[0]; */
  *pta = 4;
   pta++;
  *pta = *(pta - 1) * 2;
   pta += 3;
  (*pta)++;
  printf("index: %d\n", pta - a);

  for(i = 0; i < 5; i++) {
    printf("a[%d]: %d\n",i,a[i]);
  }
  return 0;
}
```



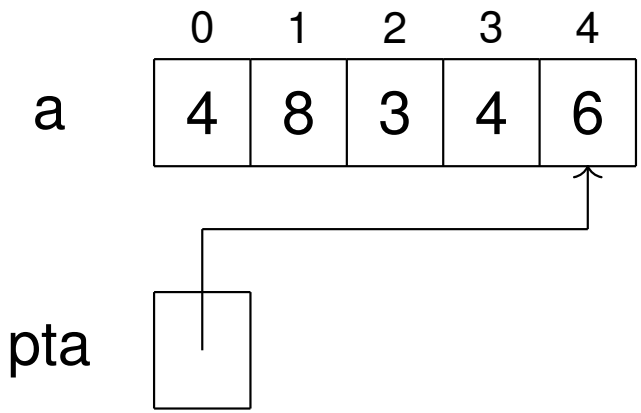
```c
#include <stdio.h>

/* array-pt-2.c */
int main( void) {
  int a[5]= {1, 2, 3, 4, 5};
  int *pta, i;

   pta = a; /* or, pta= &a[0]; */
  *pta = 4;
   pta++;
  *pta = *(pta - 1) * 2;
   pta += 3;
  (*pta)++;
  printf("index: %d\n", pta - a);

  for(i = 0; i < 5; i++) {
    printf("a[%d]: %d\n",i,a[i]);
  }
  return 0;
}
```



```c
#include <stdio.h>

/* array-pt-2.c */
int main( void) {
  int a[5]= {1, 2, 3, 4, 5};
  int *pta, i;

   pta = a; /* or, pta= &a[0]; */
  *pta = 4;
   pta++;
  *pta = *(pta - 1) * 2;
   pta += 3;
  (*pta)++;
  printf("index: %d\n", pta - a);

  for(i = 0; i < 5; i++) {
    printf("a[%d]: %d\n",i,a[i]);
  }
  return 0;
}
```



```c
#include <stdio.h>

/* array-pt-2.c */
int main( void) {
  int a[5]= {1, 2, 3, 4, 5};
  int *pta, i;

   pta = a; /* or, pta= &a[0]; */
  *pta = 4;
   pta++;
  *pta = *(pta - 1) * 2;
   pta += 3;
  (*pta)++;
  printf("index: %d\n", pta - a);

  for(i = 0; i < 5; i++) {
    printf("a[%d]: %d\n",i,a[i]);
  }
  return 0;
}
```



Pas på! C ser ikke efter om et indeks man forsøger at tilgå ligger indenfor arrayets grænser:

```c
#include <stdio.h>

int main(void) { /* array-bad.c */
  int a[3];

  /* Menigsløst resultat */
  printf("%d\n", a[3]);

  /* FARLIGT! */
  /*  a[3]= 17; */

  return 0;
}
```

Programmet skriver i et hukommelsesområde det ikke har reserveret! I bedste tilfælde er det kun programmet der crasher ...

Strenge

En streng i C er et *nulafsluttet* array af chars:

```
char s[]={ 'A', 'a', 'l', 'b', 'o', 'r', 'g', '\0' };
```

eller tilsvarende, en *pointer* til char:

```
char *s;
s= "Aalborg";
```

Følgende initialisering går også:

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

Men som *assignment* er den gal:

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

[streng-init.c]

Lav alle forekomster af 'a' om til 'i':

```
#include <stdio.h>

int main( void) { /* abrakadabra.c */
  char s[]= "abrakadabra";  /* virker */
  /* char *s= "abrakadabra"; */ /* virker IKKE */
  char *p;

  printf("%s\n", s);
  p = s;
  while (*p != '\0') {
    if (*p == 'a') {
      *p= 'i';
    }
    p++;
  }

  printf("%s\n", s);
  return 0;
}
```

- en streng kan defineres som et array af char eller en pointer til char
- begge er *nulafsluttet*: sidste indgang er '\0' ("*sentinel*")
- i strenge der er defineret som et array, kan tegnene ændres
- i strenge der er defineret som en pointer, kan tegnene *ikke* ændres
- tegnet 'a' er forskellig fra strengen "a":
  'a'= 97 "a"=['a','\0']
- den tomme streng: ""=['\0']

Biblioteket `string.h` leverer funktioner til håndtering af strenge:

- **int** strcmp(**char** *s, **char** *t)
  sammenligner s og t i *leksikografisk* orden
  < 0: s kommer *før* t
  $= 0$: s er *lig med* t
  > 0: s kommer *efter* t
- **unsigned int** strlen(**char** *s)
  returnerer antallet af tegn i s (minus '\0')
- **char** *strcpy(**char** *s, **char** *t)
  kopierer t til s
  returnerer en pointer til s
  Pas på: Hvis der ikke er plads nok i s, går det galt!
- **char** *strcat(**char** *s, **char** *t)
  tilføjer t til slutningen af s
  returnerer en pointer til s
  samme kommentar som for strcpy
- *og en del flere*

[streng-eks.c]