

## Felder (Arrays) und Zeiger (Pointers) - Teil II

### Zeichen und Texte

- Texte haben wir schon gesehen:

```
std::cout << "Prime numbers in {2,...,999}:\n";
```

String-Literal

- Können wir auch "richtig" mit Texten arbeiten? Ja:

Zeichen: Wert des fundamentalen Typs `char`

Text: Feld mit zugrundeliegendem Typ `char`

### Der Typ `char` ("character")

- repräsentiert druckbare Zeichen (z.B. `'a'`) und *Steuerzeichen* (z.B. `'\n'`)

```
char c = 'a';
```

definiert Variable `c` vom Typ `char` mit Wert `'a'`;

Literal vom Typ `char`

### Der Typ `char` ("character")

- ist formal ein ganzzahliger Typ
  - Werte konvertierbar nach `int` / `unsigned int`
  - Alle arithmetischen Operatoren verfügbar (Nutzen zweifelhaft: was ist `'a'/'b'` ?)
  - Werte belegen meistens 8 Bit

Wertebereich:

`{-128,...,127}` oder `{0,...,255}`

### Der ASCII-Code

- definiert konkrete Konversionsregeln `char` → `int` / `unsigned int`
- wird von fast allen Plattformen benutzt

Zeichen → `{0,...,127}`

`'A', 'B', ..., 'Z'` → 65, 66, ..., 90

`'a', 'b', ..., 'z'` → 97, 98, ..., 122

### Der ASCII-Code

```
for (char c = 'a'; c <= 'z'; ++c)
```

```
std::cout << c;
```

Ausgabe: `abcdefghijklmnopqrstuvwxy`

Zeichen → `{0,...,127}`

`'A', 'B', ..., 'Z'` → 65, 66, ..., 90

`'a', 'b', ..., 'z'` → 97, 98, ..., 122

## Texte

- sind repräsentierbar als Felder mit zugrundeliegendem Typ `char`

```
char text[] = {'b', 'o', 'o', 'l'}
```

definiert ein Feld der Länge 4,  
das dem Text "bool" entspricht

## Texte

- sind repräsentierbar als Felder mit zugrundeliegendem Typ `char`

```
char text[] = {'b', 'o', 'o', 'l'}
```

- können auch durch String-Literale definiert werden

```
char text[] = "bool"
```

"kennt" seine Länge!

definiert ein Feld der Länge 5, das dem Text "bool" entspricht und *null-terminiert* ist (Extrazeichen '\0' wird am Ende angehängt)

## Anwendung: *String matching*

Finde das erste (oder alle) Vorkommen eines Musters (meist kurz) in einem gegebenen Text (meist lang)!

"Trivialer" Algorithmus:

```
Gallia est omnis divisa in partes tres
*
visa
```

## Anwendung: *String matching*

Finde das erste (oder alle) Vorkommen eines Musters (meist kurz) in einem gegebenen Text (meist lang)!

"Trivialer" Algorithmus:

```
Gallia est omnis divisa in partes tres
*
visa
```

## Anwendung: *String matching*

Finde das erste (oder alle) Vorkommen eines Musters (meist kurz) in einem gegebenen Text (meist lang)!

"Trivialer" Algorithmus:

```
Gallia est omnis divisa in partes tres
*
visa
```

## Anwendung: *String matching*

Finde das erste (oder alle) Vorkommen eines Musters (meist kurz) in einem gegebenen Text (meist lang)!

"Trivialer" Algorithmus:

```
Gallia est omnis divisa in partes tres
= (gefunden!)
visa
```

## Anwendung: *String matching*

```
#include<iostream>

int main ()
{
  // search string
  char s[] = "bool";

  // determine search string length m
  unsigned int m = 0;
  for (char* p = s; *p != '\0'; ++p) ++m;

  // cyclic text window of size m
  char* t = new char[m];

  unsigned int w = 0; // number of characters read so far
  unsigned int i = 0; // index where t logically starts
  ...
}
```

Muster "fest verdrahtet" in diesem Program (aber siehe *Details* im Skript)

## Anwendung: *String matching*

```
#include<iostream>

int main ()
{
  // search string
  char s[] = "bool";

  // determine search string length m
  unsigned int m = 0;
  for (char* p = s; *p != '\0'; ++p) ++m;

  // cyclic text window of size m
  char* t = new char[m];

  unsigned int w = 0; // number of characters read so far
  unsigned int i = 0; // index where t logically starts
  ...
}
```

Rechne die Musterlänge aus (das geht, weil s null-terminiert ist)

## Anwendung: *String matching*

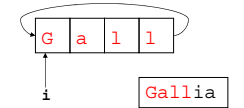
```
#include<iostream>

int main ()
{
  // search string
  char s[] = "bool";

  // determine search string length m
  unsigned int m = 0;
  for (char* p = s; *p != '\0'; ++p) ++m;

  // cyclic text window of size m
  char* t = new char[m];

  unsigned int w = 0; // number of characters read so far
  unsigned int i = 0; // index where t logically starts
  ...
}
```



## Anwendung: *String matching*

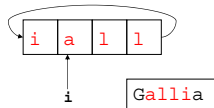
```
#include<iostream>

int main ()
{
  // search string
  char s[] = "bool";

  // determine search string length m
  unsigned int m = 0;
  for (char* p = s; *p != '\0'; ++p) ++m;

  // cyclic text window of size m
  char* t = new char[m];

  unsigned int w = 0; // number of characters read so far
  unsigned int i = 0; // index where t logically starts
  ...
}
```



## Anwendung: *String matching*

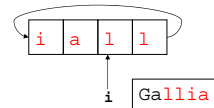
```
#include<iostream>

int main ()
{
  // search string
  char s[] = "bool";

  // determine search string length m
  unsigned int m = 0;
  for (char* p = s; *p != '\0'; ++p) ++m;

  // cyclic text window of size m
  char* t = new char[m];

  unsigned int w = 0; // number of characters read so far
  unsigned int i = 0; // index where t logically starts
  ...
}
```



## Anwendung: *String matching*

```
...
// find pattern in the text being read from std::cin
std::cin >> std::noskipws; // don't skip whitespaces!
for (unsigned int j = 0; j < m;)
  // compare search string with window at j-th element
  if (w < m || s[j] != t[(i+j)%m])
    // input text still too short, or mismatch:
    // advance window by replacing first character
    if (std::cin >> t[i]) {
      std::cout << t[i];
      ++w; // one more character read
      j = 0; // restart with first characters
      i = (i+1)%m; // of string and window
    } else break; // no more characters in the input
  else ++j; // match: go to next character

std::cout << "\n";
delete[] t;
return 0;
}
```

Leerzeichen und Zeilenbrüche sollen *nicht* ignoriert werden

## Anwendung: String matching

```

...
// find pattern in the text being read from std::cin
std::cin >> std::noskipws; // don't skip whitespaces!

for (unsigned int j = 0; j < m;)
// compare search string with window at j-th element
if (w < m || s[j] != t[(i+j)%m])
// input test still too short, or mismatch:
// advance window by replacing first character
if (std::cin >> t[i]) {
std::cout << t[i];
++w; // one more character read
j = 0; // restart with first characters
i = (i+1)%m; // of string and window
} else break; // no more characters in the input
else ++j; // match: go to next character

std::cout << "\n";
delete[] t;
return 0;
}

```

w == 0

## Anwendung: String matching

```

...
// find pattern in the text being read from std::cin
std::cin >> std::noskipws; // don't skip whitespaces!

for (unsigned int j = 0; j < m;)
// compare search string with window at j-th element
if (w < m || s[j] != t[(i+j)%m])
// input test still too short, or mismatch:
// advance window by replacing first character
if (std::cin >> t[i]) {
std::cout << t[i];
++w; // one more character read
j = 0; // restart with first characters
i = (i+1)%m; // of string and window
} else break; // no more characters in the input
else ++j; // match: go to next character

std::cout << "\n";
delete[] t;
return 0;
}

```

w == 0

## Anwendung: String matching

```

...
// find pattern in the text being read from std::cin
std::cin >> std::noskipws; // don't skip whitespaces!

for (unsigned int j = 0; j < m;)
// compare search string with window at j-th element
if (w < m || s[j] != t[(i+j)%m])
// input test still too short, or mismatch:
// advance window by replacing first character
if (std::cin >> t[i]) {
std::cout << t[i];
++w; // one more character read
j = 0; // restart with first characters
i = (i+1)%m; // of string and window
} else break; // no more characters in the input
else ++j; // match: go to next character

std::cout << "\n";
delete[] t;
return 0;
}

```

w == 0

## Anwendung: String matching

Konversion  
nach bool  
(false wenn  
Eingabe-  
strom leer)

```

...
// find pattern in the text being read from std::cin
std::cin >> std::noskipws; // don't skip whitespaces!

for (unsigned int j = 0; j < m;)
// compare search string with window at j-th element
if (w < m || s[j] != t[(i+j)%m])
// input test still too short, or mismatch:
// advance window by replacing first character
if (std::cin >> t[i]) {
std::cout << t[i];
++w; // one more character read
j = 0; // restart with first characters
i = (i+1)%m; // of string and window
} else break; // no more characters in the input
else ++j; // match: go to next character

std::cout << "\n";
delete[] t;
return 0;
}

```

w == 0

## Anwendung: String matching

```

...
// find pattern in the text being read from std::cin
std::cin >> std::noskipws; // don't skip whitespaces!

for (unsigned int j = 0; j < m;)
// compare search string with window at j-th element
if (w < m || s[j] != t[(i+j)%m])
// input test still too short, or mismatch:
// advance window by replacing first character
if (std::cin >> t[i]) {
std::cout << t[i];
++w; // one more character read
j = 0; // restart with first characters
i = (i+1)%m; // of string and window
} else break; // no more characters in the input
else ++j; // match: go to next character

std::cout << "\n";
delete[] t;
return 0;
}

```

w == 1

## Anwendung: String matching

```

...
// find pattern in the text being read from std::cin
std::cin >> std::noskipws; // don't skip whitespaces!

for (unsigned int j = 0; j < m;)
// compare search string with window at j-th element
if (w < m || s[j] != t[(i+j)%m])
// input test still too short, or mismatch:
// advance window by replacing first character
if (std::cin >> t[i]) {
std::cout << t[i];
++w; // one more character read
j = 0; // restart with first characters
i = (i+1)%m; // of string and window
} else break; // no more characters in the input
else ++j; // match: go to next character

std::cout << "\n";
delete[] t;
return 0;
}

```

w == 1

## Anwendung: String matching

```

...
// find pattern in the text being read from std::cin
std::cin >> std::noskipws; // don't skip whitespaces!
for (unsigned int j = 0; j < m;)
// compare search string with window at j-th element
if (w < m || s[j] != t[(i+j)%m])
// input test still too short, or mismatch:
// advance window by replacing first character
if (std::cin >> t[i]) {
std::cout << t[i];
++w; // one more character read
j = 0; // restart with first characters
i = (i+1)%m; // of string and window
} else break; // no more characters in the input
else ++j; // match: go to next character

std::cout << "\n";
delete[] t;
return 0;
}

```

**w == 2**

## Anwendung: String matching

```

...
// find pattern in the text being read from std::cin
std::cin >> std::noskipws; // don't skip whitespaces!
for (unsigned int j = 0; j < m;)
// compare search string with window at j-th element
if (w < m || s[j] != t[(i+j)%m])
// input test still too short, or mismatch:
// advance window by replacing first character
if (std::cin >> t[i]) {
std::cout << t[i];
++w; // one more character read
j = 0; // restart with first characters
i = (i+1)%m; // of string and window
} else break; // no more characters in the input
else ++j; // match: go to next character

std::cout << "\n";
delete[] t;
return 0;
}

```

**w == 3**

## Anwendung: String matching

```

...
// find pattern in the text being read from std::cin
std::cin >> std::noskipws; // don't skip whitespaces!
for (unsigned int j = 0; j < m;)
// compare search string with window at j-th element
if (w < m || s[j] != t[(i+j)%m])
// input test still too short, or mismatch:
// advance window by replacing first character
if (std::cin >> t[i]) {
std::cout << t[i];
++w; // one more character read
j = 0; // restart with first characters
i = (i+1)%m; // of string and window
} else break; // no more characters in the input
else ++j; // match: go to next character

std::cout << "\n";
delete[] t;
return 0;
}

```

**w == 4**

## Anwendung: String matching

```

...
// find pattern in the text being read from std::cin
std::cin >> std::noskipws; // don't skip whitespaces!
for (unsigned int j = 0; j < m;)
// compare search string with window at j-th element
if (w < m || s[j] != t[(i+j)%m])
// input test still too short, or mismatch:
// advance window by replacing first character
if (std::cin >> t[i]) {
std::cout << t[i];
++w; // one more character read
j = 0; // restart with first characters
i = (i+1)%m; // of string and window
} else break; // no more characters in the input
else ++j; // match: go to next character

std::cout << "\n";
delete[] t;
return 0;
}

```

**w == 4**

## Anwendung: String matching

```

...
// find pattern in the text being read from std::cin
std::cin >> std::noskipws; // don't skip whitespaces!
for (unsigned int j = 0; j < m;)
// compare search string with window at j-th element
if (w < m || s[j] != t[(i+j)%m])
// input test still too short, or mismatch:
// advance window by replacing first character
if (std::cin >> t[i]) {
std::cout << t[i];
++w; // one more character read
j = 0; // restart with first characters
i = (i+1)%m; // of string and window
} else break; // no more characters in the input
else ++j; // match: go to next character

std::cout << "\n";
delete[] t;
return 0;
}

```

**w == 5**

## Anwendung: String matching

```

...
// find pattern in the text being read from std::cin
std::cin >> std::noskipws; // don't skip whitespaces!
for (unsigned int j = 0; j < m;)
// compare search string with window at j-th element
if (w < m || s[j] != t[(i+j)%m])
// input test still too short, or mismatch:
// advance window by replacing first character
if (std::cin >> t[i]) {
std::cout << t[i];
++w; // one more character read
j = 0; // restart with first characters
i = (i+1)%m; // of string and window
} else break; // no more characters in the input
else ++j; // match: go to next character

std::cout << "\n";
delete[] t;
return 0;
}

```

**w == 23**

## Anwendung: String matching

```
...
// find pattern in the text being read from std::cin
std::cin >> std::noskipws; // don't skip whitespaces!
for (unsigned int j = 0; j < m;)
// compare search string with window at j-th element
if (w < m || s[j] != t[(i+j)%m])
// input text still too short, or mismatch:
// advance window by replacing first character
if (std::cin >> t[i]) {
std::cout << t[i];
++w; // one more character read
j = 0; // restart with first characters
i = (i+1)%m; // of string and window
} else break; // no more characters in the input
else ++j; // match: go to next character

std::cout << "\n";
delete[] t;
return 0;
}
w == 23
```

## Anwendung: String matching

```
...
// find pattern in the text being read from std::cin
std::cin >> std::noskipws; // don't skip whitespaces!
for (unsigned int j = 0; j < m;)
// compare search string with window at j-th element
if (w < m || s[j] != t[(i+j)%m])
// input text still too short, or mismatch:
// advance window by replacing first character
if (std::cin >> t[i]) {
std::cout << t[i];
++w; // one more character read
j = 0; // restart with first characters
i = (i+1)%m; // of string and window
} else break; // no more characters in the input
else ++j; // match: go to next character

std::cout << "\n";
delete[] t;
return 0;
}
w == 23
```

## Anwendung: String matching

```
...
// find pattern in the text being read from std::cin
std::cin >> std::noskipws; // don't skip whitespaces!
for (unsigned int j = 0; j < m;)
// compare search string with window at j-th element
if (w < m || s[j] != t[(i+j)%m])
// input text still too short, or mismatch:
// advance window by replacing first character
if (std::cin >> t[i]) {
std::cout << t[i];
++w; // one more character read
j = 0; // restart with first characters
i = (i+1)%m; // of string and window
} else break; // no more characters in the input
else ++j; // match: go to next character

std::cout << "\n";
delete[] t;
return 0;
}
w == 23
```

## Anwendung: String matching

```
...
// find pattern in the text being read from std::cin
std::cin >> std::noskipws; // don't skip whitespaces!
for (unsigned int j = 0; j < m;)
// compare search string with window at j-th element
if (w < m || s[j] != t[(i+j)%m])
// input text still too short, or mismatch:
// advance window by replacing first character
if (std::cin >> t[i]) {
std::cout << t[i];
++w; // one more character read
j = 0; // restart with first characters
i = (i+1)%m; // of string and window
} else break; // no more characters in the input
else ++j; // match: go to next character

std::cout << "\n";
delete[] t;
return 0;
}
w == 23
```

## Anwendung: String matching

```
...
// find pattern in the text being read from std::cin
std::cin >> std::noskipws; // don't skip whitespaces!
for (unsigned int j = 0; j < m;)
// compare search string with window at j-th element
if (w < m || s[j] != t[(i+j)%m])
// input text still too short, or mismatch:
// advance window by replacing first character
if (std::cin >> t[i]) {
std::cout << t[i];
++w; // one more character read
j = 0; // restart with first characters
i = (i+1)%m; // of string and window
} else break; // no more characters in the input
else ++j; // match: go to next character

std::cout << "\n";
delete[] t;
return 0;
}
w == 23
```

## Anwendung: String matching

```
...
// find pattern in the text being read from std::cin
std::cin >> std::noskipws; // don't skip whitespaces!
for (unsigned int j = 0; j < m;)
// compare search string with window at j-th element
if (w < m || s[j] != t[(i+j)%m])
// input text still too short, or mismatch:
// advance window by replacing first character
if (std::cin >> t[i]) {
std::cout << t[i];
++w; // one more character read
j = 0; // restart with first characters
i = (i+1)%m; // of string and window
} else break; // no more characters in the input
else ++j; // match: go to next character

std::cout << "\n";
delete[] t;
return 0;
}
w == 23
```

## Anwendung: *String matching*

```

...
// find pattern in the text being read from std::cin
std::cin >> std::noskipws; // don't skip whitespaces!
for (unsigned int j = 0; j < m; j++)
    if (w < m || s[j] != t[(i+j)%m])
        continue;
    // compare search string with window at j-th element
    // input test still too short, or mismatch:
    // advance window by replacing first character
    if (std::cin >> t[i]) {
        std::cout << t[i];
        ++w; // one more character read
        j = 0; // restart with first characters
        i = (i+1)%m; // of string and window
    } else break; // no more characters in the input
    else ++j; // match: go to next character

std::cout << "\n";
delete[] t;
return 0;
}
w == 23
    
```

## Anwendung: *String matching*

```

...
// find pattern in the text being read from std::cin
std::cin >> std::noskipws; // don't skip whitespaces!
for (unsigned int j = 0; j < m; j++)
    if (w < m || s[j] != t[(i+j)%m])
        continue;
    // compare search string with window at j-th element
    // input test still too short, or mismatch:
    // advance window by replacing first character
    if (std::cin >> t[i]) {
        std::cout << t[i];
        ++w; // one more character read
        j = 0; // restart with first characters
        i = (i+1)%m; // of string and window
    } else break; // no more characters in the input
    else ++j; // match: go to next character

std::cout << "\n";
delete[] t;
return 0;
}
w == 23
    
```

Fertig! Die ersten 23 Text-  
zeichen wurden ausgegeben

## Anwendung: *String matching*

- Aufruf des Programms z.B. durch

```
./string_matching < eratosthenes.C
```

Eingabe wird nicht von der Tastatur, sondern aus der  
angegebenen Datei genommen (Umlenkung der Eingabe)

## Anwendung: *String matching*

- Aufruf des Programms z.B. durch

```
./string_matching < eratosthenes.C
```

- Ausgabe:

```

// Program: eratosthenes.C
// Calculate prime numbers in {2,...,999} using
// Eratosthenes' sieve.

#include <iostream>

int main()
{
    // definition and initialization: provides us with
    // Booleans crossed_out[0]...crossed_out[999]
    bool
    
```

## Anwendung: *String matching*

- Aufruf des Programms z.B. durch

```
./string_matching < eratosthenes.C >
match.out
```

Ausgabe wird nicht auf den  
Bildschirm geschrieben,  
sondern in die angegebene  
Datei (Umlenkung der  
Ausgabe)

## Anwendung: *String matching*

- Aufruf des Programms z.B. durch

```
./string_matching < eratosthenes.C >
match.out
```

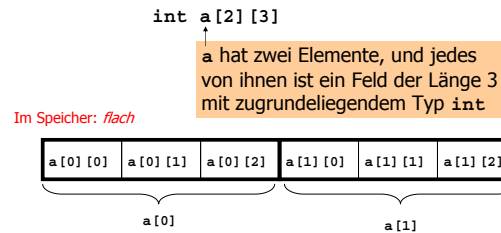
- Der triviale Algorithmus ist meistens  
schnell, aber nicht immer (Übung)
- *Knuth-Morris-Pratt*-Algorithmus ist  
immer schnell

## Mehrdimensionale Felder

- sind Felder von Feldern
- dienen zum Speichern von *Tabellen*, *Matrizen*,...

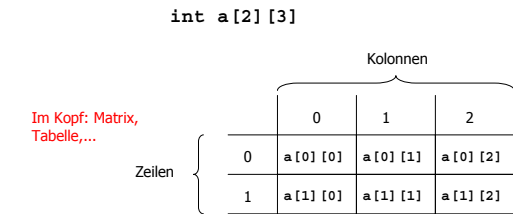
## Mehrdimensionale Felder

- sind Felder von Feldern



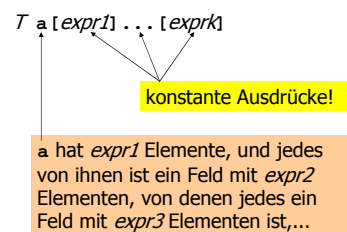
## Mehrdimensionale Felder

- sind Felder von Feldern



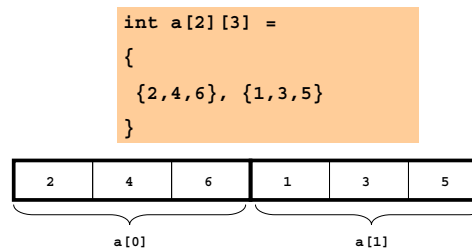
## Mehrdimensionale Felder

- sind Felder von Feldern von Feldern...



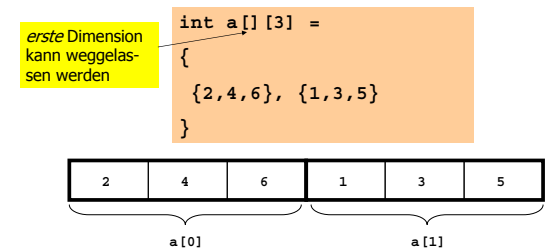
## Mehrdimensionale Felder

Initialisierung:



## Mehrdimensionale Felder

Initialisierung:

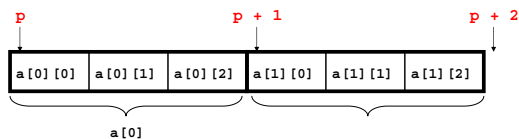




## Zeiger auf Felder

Wie iteriert man natürlich über ein mehrdimensionales Feld?

```
int a[2][3];
int (*p)[3] = a; // Zeiger auf erstes Element
```



## Zeiger auf Felder

Wie iteriert man natürlich über ein mehrdimensionales Feld?

```
int a[2][3];
int (*p)[3] = a; // Zeiger auf erstes Element
```

Implizite Typdefinition: \*p ist vom Typ int[3], also ist p ein Zeiger auf int[3]

## Zeiger auf Felder

Wie iteriert man natürlich über ein mehrdimensionales Feld?

```
int a[2][3];
int (*p)[3] = a; // Zeiger auf erstes Element
```

Ohne Klammern: p ist ein Feld von 3 Zeigern auf int

```
int *p[3]; // int* p[3]
```

## Zeiger auf Felder

Wie iteriert man natürlich über ein mehrdimensionales Feld?

```
int a[2][3];
int (*p)[3] = a; // Zeiger auf erstes Element
```

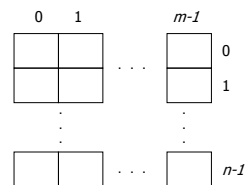
Ohne Klammern: p ist ein Feld von 3 Zeigern auf int

```
int *p[3]
```

Solche Syntax-"Gemeinheiten" muss man nicht sich nicht merken, nur bei Bedarf nachschauen (Skript)!

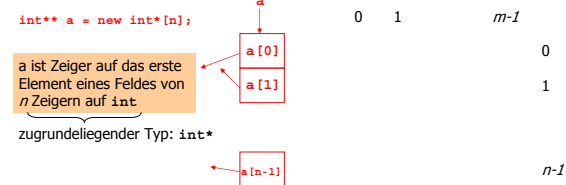
## Felder von Zeigern

Wie bekommen wir mehrdimensionale Felder mit variablen Dimensionen?



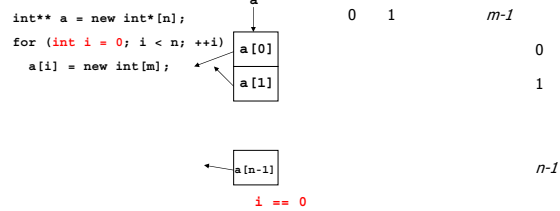
## Felder von Zeigern

Wie bekommen wir mehrdimensionale Felder mit variablen Dimensionen?



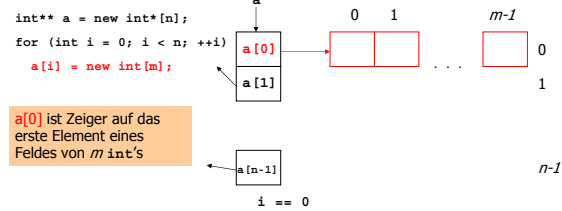
## Felder von Zeigern

- Wie bekommen wir mehrdimensionale Felder mit variablen Dimensionen?



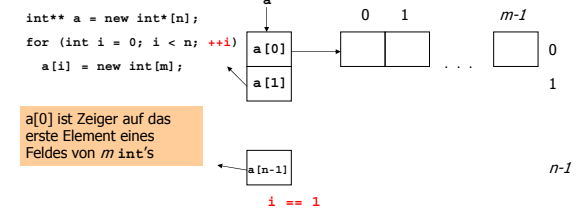
## Felder von Zeigern

- Wie bekommen wir mehrdimensionale Felder mit variablen Dimensionen?



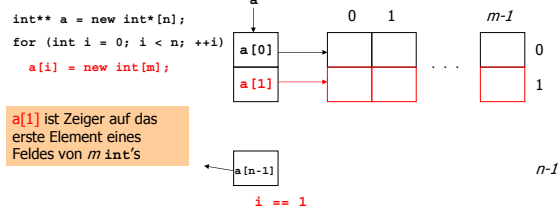
## Felder von Zeigern

- Wie bekommen wir mehrdimensionale Felder mit variablen Dimensionen?



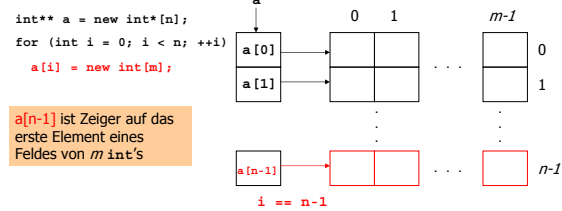
## Felder von Zeigern

- Wie bekommen wir mehrdimensionale Felder mit variablen Dimensionen?



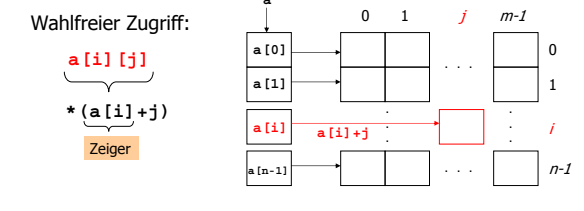
## Felder von Zeigern

- Wie bekommen wir mehrdimensionale Felder mit variablen Dimensionen?



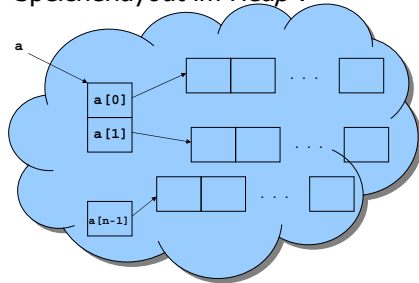
## Felder von Zeigern

- Wie bekommen wir mehrdimensionale Felder mit variablen Dimensionen?



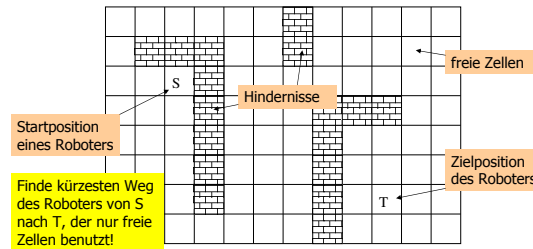
## Felder von Zeigern

- Speicherlayout im *Heap*:



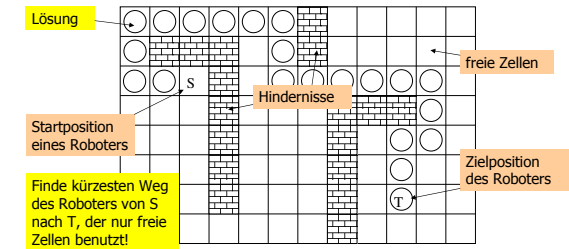
## Anwendung: kürzeste Wege

Fabrik-Halle ( $n \times m$  quadratische Zellen)



## Anwendung: kürzeste Wege

Fabrik-Halle ( $n \times m$  quadratische Zellen)



## Ein (scheinbar) anderes Problem

Finde die *Längen* der kürzesten Wege zu *allen* möglichen Zielen!

Das löst auch das Original-Problem: starte in T; folge einem Weg mit (jeweils um eins) *sinken-*den Längen!

4	5	6	7	8	9	15	16	17	18	19
3	2	1	0	10	11	14	15	16	17	18
2	1	0	10	11	12	13	14	15	16	17
3	2	1	11	12	13	14	15	16	17	18
4	3	2	10	11	12	13	14	15	16	17
5	4	3	9	10	11	12	13	14	15	16
6	5	4	8	9	10	11	12	13	14	15
7	6	5	7	8	9	10	11	12	13	14

Zielposition des Roboters; kürzester Weg hat Länge 21

## Ein (scheinbar) anderes Problem

Finde die *Längen* der kürzesten Wege zu *allen* möglichen Zielen!

Das löst auch das Original-Problem: starte in T; folge einem Weg mit (jeweils um eins) *sinken-*den Längen!

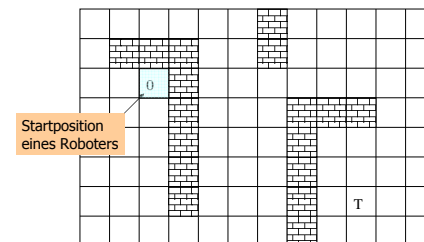
4	5	6	7	8	9	15	16	17	18	19
3	2	1	0	10	11	14	15	16	17	18
2	1	0	10	11	12	13	14	15	16	17
3	2	1	11	12	13	14	15	16	17	18
4	3	2	10	11	12	13	14	15	16	17
5	4	3	9	10	11	12	13	14	15	16
6	5	4	8	9	10	11	12	13	14	15
7	6	5	7	8	9	10	11	12	13	14

Weglänge 21 (optimal)

Zielposition des Roboters; kürzester Weg hat Länge 21

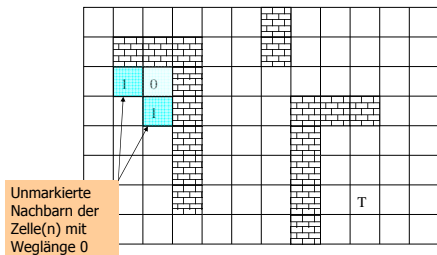
## Markierung aller Zellen mit ihren Weglängen

Schritt 0: Alle Zellen mit Weglänge 0:



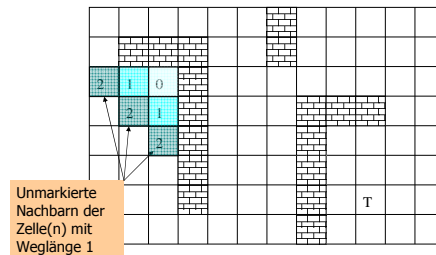
## Markierung aller Zellen mit ihren Weglängen

Schritt 1: Alle Zellen mit Weglänge 1:



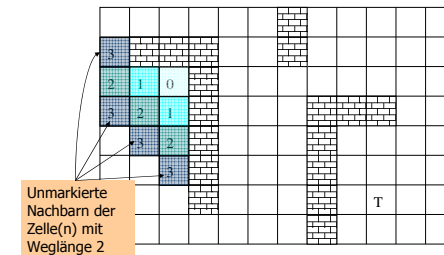
## Markierung aller Zellen mit ihren Weglängen

Schritt 2: Alle Zellen mit Weglänge 2:



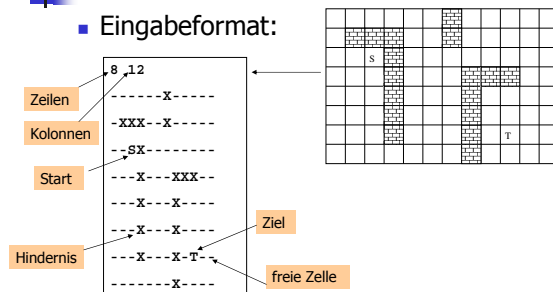
## Markierung aller Zellen mit ihren Weglängen

Schritt 3: Alle Zellen mit Weglänge 3:



## Das Kürzeste-Wege-Programm

■ Eingabeformat:



## Das Kürzeste-Wege-Programm

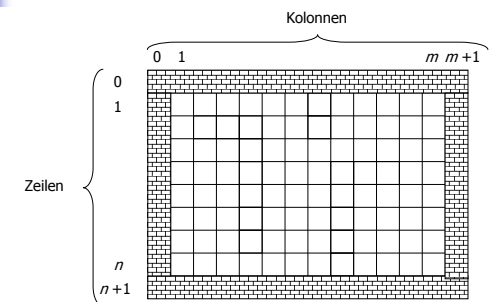
■ Einlesen der Dimensionen und Bereitstellung eines zweidimensionalen Feldes für die Weglängen

```
int n; std::cin >> n; // number of rows
int m; std::cin >> m; // number of columns

// dynamically allocate twodimensional array of dimensions
// (n+2) x (m+2) to hold the floor plus extra walls around
int** floor = new int*[n+2];
for (int r=0; r<n+2; ++r)
    floor[r] = new int[m+2];
```

Wächter (sentinels)

## Das Kürzeste-Wege-Programm



## Das Kürzeste-Wege-Programm

- Einlesen der Hallenbelegung und Initialisierung der Längen

```
int tr = 0;
int tc = 0;
for (int r=1; r<n+1; ++r)
  for (int c=1; c<m+1; ++c) {
    char entry = '-';
    std::cin >> entry;
    if (entry == 'S') floor[x][c] = 0;
    else if (entry == 'T') floor[tr = x][tc = c] = -1;
    else if (entry == 'X') floor[x][c] = -2;
    else if (entry == '-') floor[x][c] = -1;
  }
```

Zielkoordinaten (Zeilen-/Kolonnenindex)

## Das Kürzeste-Wege-Programm

- Einlesen der Hallenbelegung und Initialisierung der Längen

```
int tr = 0;
int tc = 0;
for (int r=1; r<n+1; ++r)
  for (int c=1; c<m+1; ++c) {
    char entry = '-';
    std::cin >> entry;
    if (entry == 'S') floor[x][c] = 0;
    else if (entry == 'T') floor[tr = x][tc = c] = -1;
    else if (entry == 'X') floor[x][c] = -2;
    else if (entry == '-') floor[x][c] = -1;
  }
```

lies die Eingabe Zeile für Zeile (z.B. durch Umlenkung der Eingabe auf die Datei `shortest_path0.dat`)

## Das Kürzeste-Wege-Programm

- Einlesen der Hallenbelegung und Initialisierung der Längen

```
int tr = 0;
int tc = 0;
for (int r=1; r<n+1; ++r)
  for (int c=1; c<m+1; ++c) {
    char entry = '-';
    std::cin >> entry;
    if (entry == 'S') floor[x][c] = 0;
    else if (entry == 'T') floor[tr = r][tc = c] = -1;
    else if (entry == 'X') floor[x][c] = -2;
    else if (entry == '-') floor[x][c] = -1;
  }
```

Eingabezeichen in Zeile  $r \in \{1, \dots, n\}$  und Kolonne  $c \in \{1, \dots, m\}$ .

## Das Kürzeste-Wege-Programm

- Einlesen der Hallenbelegung und Initialisierung der Längen

```
int tr = 0;
int tc = 0;
for (int r=1; r<n+1; ++r)
  for (int c=1; c<m+1; ++c) {
    char entry = '-';
    std::cin >> entry;
    if (entry == 'S') floor[x][c] = 0;
    else if (entry == 'T') floor[tr = x][tc = c] = -1;
    else if (entry == 'X') floor[x][c] = -2;
    else if (entry == '-') floor[x][c] = -1;
  }
```

Länge bereits bekannt

## Das Kürzeste-Wege-Programm

- Einlesen der Hallenbelegung und Initialisierung der Längen

```
int tr = 0;
int tc = 0;
for (int r=1; r<n+1; ++r)
  for (int c=1; c<m+1; ++c) {
    char entry = '-';
    std::cin >> entry;
    if (entry == 'S') floor[x][c] = 0;
    else if (entry == 'T') floor[tr = x][tc = c] = -1;
    else if (entry == 'X') floor[x][c] = -2;
    else if (entry == '-') floor[x][c] = -1;
  }
```

Ziel: setze Koordinaten

-1: Länge noch unbekannt

## Das Kürzeste-Wege-Programm

- Einlesen der Hallenbelegung und Initialisierung der Längen

```
int tr = 0;
int tc = 0;
for (int r=1; r<n+1; ++r)
  for (int c=1; c<m+1; ++c) {
    char entry = '-';
    std::cin >> entry;
    if (entry == 'S') floor[x][c] = 0;
    else if (entry == 'T') floor[tr = r][tc = c] = -1;
    else if (entry == 'X') floor[x][c] = -2;
    else if (entry == '-') floor[x][c] = -1;
  }
```

-2: Länge nicht relevant (Hindernis)

## Das Kürzeste-Wege-Programm

- Einlesen der Hallenbelegung und Initialisierung der Längen

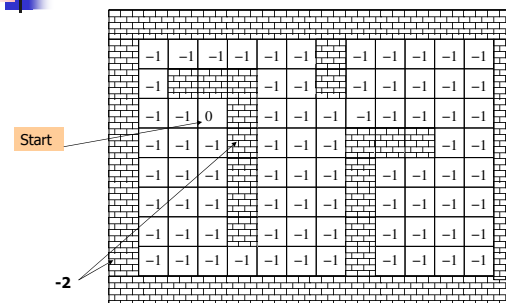
```
int tr = 0;
int tc = 0;
for (int r=1; r<n+1; ++r)
  for (int c=1; c<m+1; ++c) {
    char entry = '-';
    std::cin >> entry;
    if (entry == 'S') floor[r][c] = 0;
    else if (entry == 'T') floor[tr][tc] = c = -1;
    else if (entry == 'X') floor[r][c] = -2;
    else if (entry == '-') floor[r][c] = -1; ← -1: Länge noch unbekannt (freie Zelle)
  }
```

## Das Kürzeste-Wege-Programm

- Hinzufügen der umschliessenden "Wände"

```
for (int r=0; r<n+2; ++r)
  floor[r][0] = floor[r][m+1] = -2; ← Kolonnen 0 und m+1
for (int c=0; c<m+2; ++c)
  floor[0][c] = floor[n+1][c] = -2; ← Zeilen 0 und n+1
```

## Das Kürzeste-Wege-Programm



## Das Kürzeste-Wege-Programm

```
for (int i=1; ++i) {
  bool progress = false;
  for (int r=1; r<n+1; ++r)
    for (int c=1; c<m+1; ++c) {
      if (floor[r][c] != -1) continue;
      if (floor[r-1][c] == i-1 || floor[r+1][c] == i-1 ||
          floor[r][c-1] == i-1 || floor[r][c+1] == i-1) {
        floor[r][c] = i; // label cell with i
        progress = true;
      }
    }
  if (!progress) break;
}
```

Hauptschleife: finde und markiere alle Zellen mit Weglängen i=1,2,3...

## Das Kürzeste-Wege-Programm

```
for (int i=1; ++i) {
  bool progress = false; ← Haben wir für dieses i eine Zelle gefunden?
  for (int r=1; r<n+1; ++r)
    for (int c=1; c<m+1; ++c) {
      if (floor[r][c] != -1) continue;
      if (floor[r-1][c] == i-1 || floor[r+1][c] == i-1 ||
          floor[r][c-1] == i-1 || floor[r][c+1] == i-1) {
        floor[r][c] = i; // label cell with i
        progress = true;
      }
    }
  if (!progress) break;
}
```

## Das Kürzeste-Wege-Programm

```
for (int i=1; ++i) {
  bool progress = false;
  for (int r=1; r<n+1; ++r) ← Iteriere über alle "echten" Zellen
    for (int c=1; c<m+1; ++c) {
      if (floor[r][c] != -1) continue;
      if (floor[r-1][c] == i-1 || floor[r+1][c] == i-1 ||
          floor[r][c-1] == i-1 || floor[r][c+1] == i-1) {
        floor[r][c] = i; // label cell with i
        progress = true;
      }
    }
  if (!progress) break;
}
```

## Das Kürzeste-Wege-Programm

```

for (int i=1;; ++i) {
    bool progress = false;
    for (int r=1; r<n+1; ++r)
        for (int c=1; c<m+1; ++c) {
            if (floor[r][c] != -1) continue;
            if (floor[r-1][c] == i-1 || floor[r+1][c] == i-1 ||
                floor[r][c-1] == i-1 || floor[r][c+1] == i-1) {
                floor[r][c] = i; // label cell with i
                progress = true;
            }
        }
    if (!progress) break;
}

```

Betrachte Zelle in Zeile  $r \in \{1, \dots, n\}$  und Kolonne  $c \in \{1, \dots, m\}$ :

**Fall 1:** Hindernis, oder schon markiert

## Das Kürzeste-Wege-Programm

```

for (int i=1;; ++i) {
    bool progress = false;
    for (int r=1; r<n+1; ++r)
        for (int c=1; c<m+1; ++c) {
            if (floor[r][c] != -1) continue;
            if (floor[r-1][c] == i-1 || floor[r+1][c] == i-1 ||
                floor[r][c-1] == i-1 || floor[r][c+1] == i-1) {
                floor[r][c] = i; // label cell with i
                progress = true;
            }
        }
    if (!progress) break;
}

```

Betrachte Zelle in Zeile  $r \in \{1, \dots, n\}$  und Kolonne  $c \in \{1, \dots, m\}$ :

**Fall 2:** Ein Nachbar hat bereits Weglänge  $i-1$

Durch die Wächter hat jede "echte" Zelle 4 Nachbarn!

## Das Kürzeste-Wege-Programm

```

for (int i=1;; ++i) {
    bool progress = false;
    for (int r=1; r<n+1; ++r)
        for (int c=1; c<m+1; ++c) {
            if (floor[r][c] != -1) continue;
            if (floor[r-1][c] == i-1 || floor[r+1][c] == i-1 ||
                floor[r][c-1] == i-1 || floor[r][c+1] == i-1) {
                floor[r][c] = i; // label cell with i
                progress = true;
            }
        }
    if (!progress) break;
}

```

Betrachte Zelle in Zeile  $r \in \{1, \dots, n\}$  und Kolonne  $c \in \{1, \dots, m\}$ :

**Fall 2:** Ein Nachbar hat bereits Weglänge  $i-1$

Markiere Zelle mit  $i$

## Das Kürzeste-Wege-Programm

```

for (int i=1;; ++i) {
    bool progress = false;
    for (int r=1; r<n+1; ++r)
        for (int c=1; c<m+1; ++c) {
            if (floor[r][c] != -1) continue;
            if (floor[r-1][c] == i-1 || floor[r+1][c] == i-1 ||
                floor[r][c-1] == i-1 || floor[r][c+1] == i-1) {
                floor[r][c] = i; // label cell with i
                progress = true;
            }
        }
    if (!progress) break;
}

```

Falls keine Zelle mehr markiert werden konnte, sind wir fertig (sonst geht's weiter mit  $i+1$ )

## Das Kürzeste-Wege-Programm

- Markieren des kürzesten Weges durch "Rückwärtslaufen" vom Ziel zum Start

```

int r = tr; int c = tc; // Zielzelle
while (floor[r][c] > 0) {
    int d = floor[r][c] - 1;
    floor[r][c] = -3;
    if (floor[r-1][c] == d) --r;
    else if (floor[r+1][c] == d) ++r;
    else if (floor[r][c-1] == d) --c;
    else ++c; // (floor[r][c+1] == d)
}

```

## Das Kürzeste-Wege-Programm

- Markieren des kürzesten Weges durch "Rückwärtslaufen" vom Ziel zum Start

```

int r = tr; int c = tc;
while (floor[r][c] > 0) {
    int d = floor[r][c] - 1;
    floor[r][c] = -3;
    if (floor[r-1][c] == d) --r;
    else if (floor[r+1][c] == d) ++r;
    else if (floor[r][c-1] == d) --c;
    else ++c; // (floor[r][c+1] == d)
}

```

Solange Startzelle noch nicht erreicht...

## Das Kürzeste-Wege-Programm

- Markieren des kürzesten Weges durch "Rückwärtslaufen" vom Ziel zum Start

```
int r = tr; int c = tc;
while (floor[r][c] > 0) {
    int d = floor[r][c] - 1;
    floor[r][c] = -3;
    if (floor[r-1][c] == d) --r;
    else if (floor[r+1][c] == d) ++r;
    else if (floor[r][c-1] == d) --c;
    else ++c; // (floor[r][c+1] == d)
}
```

d = um eins kleinere Weglänge

## Das Kürzeste-Wege-Programm

- Markieren des kürzesten Weges durch "Rückwärtslaufen" vom Ziel zum Start

```
int r = tr; int c = tc;
while (floor[r][c] > 0) {
    int d = floor[r][c] - 1;
    floor[r][c] = -3;
    if (floor[r-1][c] == d) --r;
    else if (floor[r+1][c] == d) ++r;
    else if (floor[r][c-1] == d) --c;
    else ++c; // (floor[r][c+1] == d)
}
```

Markiere Zelle (mit -3 zur Unterscheidung): sie liegt auf dem kürzesten Weg

## Das Kürzeste-Wege-Programm

- Markieren des kürzesten Weges durch "Rückwärtslaufen" vom Ziel zum Start

```
int r = tr; int c = tc;
while (floor[r][c] > 0) {
    int d = floor[r][c] - 1;
    floor[r][c] = -3;
    if (floor[r-1][c] == d) --r;
    else if (floor[r+1][c] == d) ++r;
    else if (floor[r][c-1] == d) --c;
    else ++c; // (floor[r][c+1] == d)
}
```

Gehe zu einer Nachbarzelle mit Weglänge d und wiederhole...

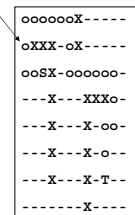
## Das Kürzeste-Wege-Programm

-3	-3	-3	-3	-3	-3	15	16	17	18	19	
-3			9	-3		14	15	16	17	18	
-3	-3	0		10	-3	-3	-3	-3	-3	17	
3	2	1		11	12	13			-3	18	
4	3	2		10	11	12		20	-3	19	
5	4	3		9	10	11		21	-3	20	
6	5	4		8	9	10		22	-3	21	
7	6	5	6	7	8	9		23	22	21	22

## Das Kürzeste-Wege-Programm

- Ausgabe: auf dem kürzesten Weg

```
for (int r=1; r<n+1; ++r) {
    for (int c=1; c<n+1; ++c)
        if (floor[r][c] == 0) std::cout << 'S';
        else if (r == tr && c == tc) std::cout << 'T';
        else if (floor[r][c] == -3) std::cout << 'o';
        else if (floor[r][c] == -2) std::cout << 'X';
        else std::cout << '-';
    std::cout << "\n";
}
```



## Das Kürzeste-Wege-Programm

- Last, but not least: lösche die auf dem Heap bereitgestellten Felder

```
for (int r=0; r<n+2; ++r)
    delete[] floor[r];
delete[] floor;
```

Feld für die Zeile mit Index r

Feld von Zeigern auf die n+2 Zeilen





## Das Kürzeste-Wege-Programm

- Das Programm kann recht langsam sein, weil für jedes  $i$  *alle* Zellen durchlaufen werden
- Verbesserung: durchlaufe jeweils nur die Nachbarn der Zellen mit Markierung  $i-1$
- *Challenge* : **der leichteste bisher...**