

# 11. Referenztypen

Referenztypen: Definition und Initialisierung, Pass By Value , Pass by Reference, Temporäre Objekte, Konstanten, Const-Referenzen

# Swap!

// POST: values of x and y are exchanged

```
void swap (int& x, int& y) {
```

```
    int t = x;
```

```
    x = y;
```

```
    y = t;
```

```
}
```

```
int main(){
```

```
    int a = 2;
```

```
    int b = 1;
```

```
    swap (a, b);
```

```
    assert (a == 1 && b == 2); // ok! 😊
```

```
}
```

# Referenztypen

- Wir können Funktionen in die Lage versetzen, die Werte ihrer Aufrufargumente zu ändern!

# Referenztypen

- Wir können Funktionen in die Lage versetzen, die Werte ihrer Aufrufargumente zu ändern!
- Kein neues Konzept auf der Funktionenseite, sondern eine neue Klasse von Typen

# Referenztypen

- Wir können Funktionen in die Lage versetzen, die Werte ihrer Aufrufargumente zu ändern!
- Kein neues Konzept auf der Funktionenseite, sondern eine neue Klasse von Typen

Referenztypen



# Referenztypen: Definition

*T*&

Gelesen als „*T*-Referenz“



Zugrundeliegender Typ

# Referenztypen: Definition

$T\&$

Gelesen als „ $T$ -Referenz“



Zugrundeliegender Typ

- $T\&$  hat den gleichen Wertebereich und gleiche Funktionalität wie  $T$ , ...

# Referenztypen: Definition

*T*&

Gelesen als „*T*-Referenz“



Zugrundeliegender Typ

- *T*& hat den gleichen Wertebereich und gleiche Funktionalität wie *T*, ...
- nur Initialisierung und Zuweisung funktionieren anders.



# Anakin Skywalker alias Darth Vader



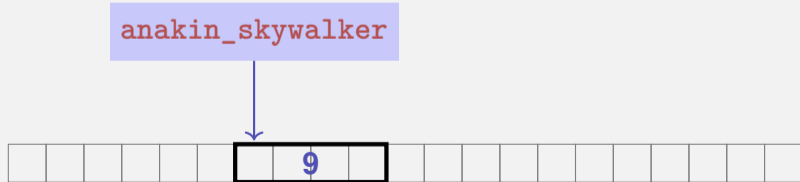
# Anakin Skywalker alias Darth Vader

```
int anakin_skywalker = 9;  
int& darth_vader = anakin_skywalker; // Alias  
darth_vader = 22;  
  
std::cout << anakin_skywalker;
```

# Anakin Skywalker alias Darth Vader

```
int anakin_skywalker = 9;  
int& darth_vader = anakin_skywalker; // Alias  
darth_vader = 22;
```

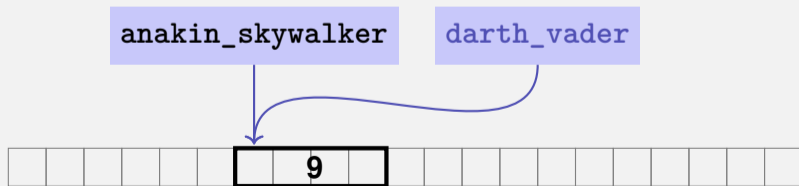
```
std::cout << anakin_skywalker;
```



# Anakin Skywalker alias Darth Vader

```
int anakin_skywalker = 9;  
int& darth_vader = anakin_skywalker; // Alias  
darth_vader = 22;
```

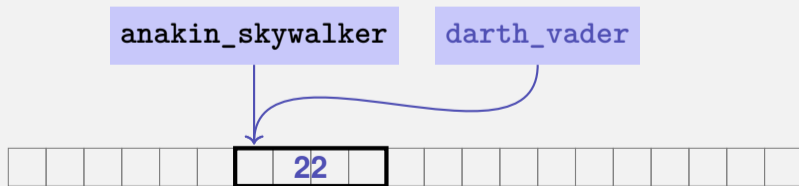
```
std::cout << anakin_skywalker;
```



# Anakin Skywalker alias Darth Vader

```
int anakin_skywalker = 9;  
int& darth_vader = anakin_skywalker; // Alias  
darth_vader = 22;
```

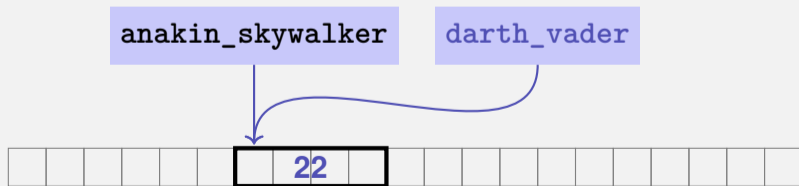
```
std::cout << anakin_skywalker;
```



# Anakin Skywalker alias Darth Vader

```
int anakin_skywalker = 9;  
int& darth_vader = anakin_skywalker; // Alias  
darth_vader = 22;  
std::cout << anakin_skywalker;
```

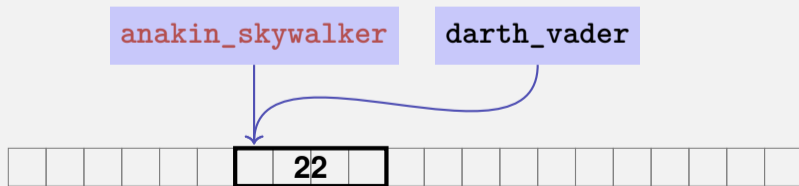
Zuweisung an den L-Wert hinter dem Alias



# Anakin Skywalker alias Darth Vader

```
int anakin_skywalker = 9;  
int& darth_vader = anakin_skywalker; // Alias  
darth_vader = 22;
```

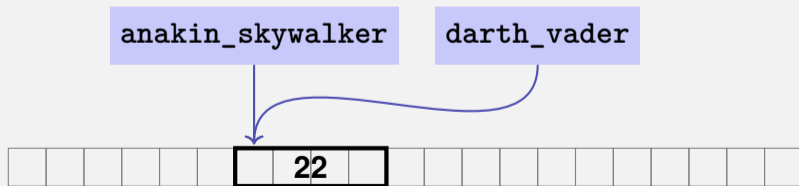
```
std::cout << anakin_skywalker; // 22
```



# Anakin Skywalker alias Darth Vader

```
int anakin_skywalker = 9;  
int& darth_vader = anakin_skywalker; // Alias  
darth_vader = 22;
```

```
std::cout << anakin_skywalker; // 22
```





# Referenztypen: Initialisierung & Zuweisung

```
int& darth_vader = anakin_skywalker;
```

- Eine Variable mit **Referenztyp** (eine *Referenz*) kann nur mit einem **L-Wert** initialisiert werden.

# Referenztypen: Initialisierung & Zuweisung

```
int& darth_vader = anakin_skywalker;
```

- Eine Variable mit Referenztyp (eine *Referenz*) kann nur mit einem L-Wert initialisiert werden.
- Die Variable wird dabei ein *Alias* des L-Werts (ein anderer Name für das referenzierte Objekt).

# Referenztypen: Initialisierung & Zuweisung

```
int& darth_vader = anakin_skywalker;  
darth_vader = 22; // anakin_skywalker = 22
```

- Eine Variable mit Referenztyp (eine *Referenz*) kann nur mit einem L-Wert initialisiert werden.
- Die Variable wird dabei ein *Alias* des L-Werts (ein anderer Name für das referenzierte Objekt).
- Zuweisung an die Referenz erfolgt an das **Objekt** hinter dem Alias.

# Referenztypen: Realisierung

Intern wird ein Wert vom Typ  $T\&$  durch die Adresse eines Objekts vom Typ  $T$  repräsentiert.

```
int& j; // Fehler: j muss Alias von irgendetwas sein
```

# Referenztypen: Realisierung

Intern wird ein Wert vom Typ  $T\&$  durch die Adresse eines Objekts vom Typ  $T$  repräsentiert.

```
int& j; // Fehler: j muss Alias von irgendetwas sein
```

```
int& k = 5; // Fehler: Das Literal 5 hat keine Adresse
```

# Pass by Reference

```
void increment (int& i)
{
    ++i;
}

...
int j = 5;
increment (j);
std::cout << j << "\n"; // 6
```

# Pass by Reference

```
void increment (int& i)
{
    ++i;
}
...
int j = 5;
increment (j);
std::cout << j << "\n"; // 6
```



# Pass by Reference

```
void increment (int& i) ← Initialisierung der formalen Argumente  
{ // i wird Alias des Aufrufarguments  
    ++i;  
}  
...  
int j = 5;  
increment (j);  
std::cout << j << "\n"; // 6
```





# Pass by Reference

```
void increment (int& i)
{
    ++i;
}
...
int j = 5;
increment (j);
std::cout << j << "\n"; // 6
```



# Pass by Reference

```
void increment (int& i)
{
    ++i;
}
...
int j = 5;
increment (j);
std::cout << j << "\n"; // 6
```



# Pass by Reference

Formales Argument hat Referenztyp:

⇒ **Pass by Reference**

Formales Argument wird (intern) mit der *Adresse* des Aufrufarguments (L-Wert) initialisiert und wird damit zu einem *Alias*.

# Pass by Value

Formales Argument hat keinen Referenztyp:

⇒ **Pass by Value**

Formales Argument wird mit dem *Wert* des Aufrufarguments (R-Wert) initialisiert und wird damit zu einer *Kopie*.

# Referenzen im Kontext von `intervals_intersect`

```
// PRE: [a1, b1], [a2, b2] are (generalized) intervals,
// POST: returns true if [a1, b1], [a2, b2] intersect, in which case
//       [l, h] contains the intersection of [a1, b1], [a2, b2]
bool intervals_intersect (int& l, int& h,
                          int a1, int b1, int a2, int b2) {
    sort (a1, b1);
    sort (a2, b2);
    l = std::max (a1, a2); // Zuweisungen
    h = std::min (b1, b2); // via Referenzen
    return l <= h;
}
...
int lo = 0; int hi = 0;
if (intervals_intersect (lo, hi, 0, 2, 1, 3)) // Initialisierung
    std::cout << "[" << lo << "," << hi << "]" << "\n"; // [1,2]
```



# Referenzen im Kontext von `intervals_intersect`

```
// POST: a <= b
void sort (int& a, int& b) {
    if (a > b)
        std::swap (a, b); // Initialisierung ("Durchreichen" von a, b)
}
```

```
bool intervals_intersect (int& l, int& h,
                          int a1, int b1, int a2, int b2) {
    sort (a1, b1); // Initialisierung
    sort (a2, b2); // Initialisierung
    l = std::max (a1, a2);
    h = std::min (b1, b2);
    return l <= h;
}
```

# Return by Value / Reference

- Auch der Rückgabebetyp einer Funktion kann ein Referenztyp sein ( return by reference )

# Return by Value / Reference

- Auch der Rückgabebetyp einer Funktion kann ein Referenztyp sein ( return by reference )
- In diesem Fall ist der Funktionsaufruf selbst ein L-Wert



# Return by Value / Reference

- Auch der Rückgabebetyp einer Funktion kann ein Referenztyp sein ( return by reference )
- In diesem Fall ist der Funktionsaufruf selbst ein L-Wert

# Return by Value / Reference

- Auch der Rückgabetypp einer Funktion kann ein Referenztyp sein ( return by reference )
- In diesem Fall ist der Funktionsausruf selbst ein L-Wert

```
int& increment (int& i)
{
    return ++i;
}
```

# Return by Value / Reference

- Auch der Rückgabebetyp einer Funktion kann ein Referenztyp sein ( return by reference )
- In diesem Fall ist der Funktionsaufruf selbst ein L-Wert

```
int& increment (int& i)
{
    return ++i;
}
```

Exakt die Semantik des Prä-Inkrement

# Temporäre Objekte

Was ist hier falsch?

```
int& foo (int i)
{
    return i;
}
```

# Temporäre Objekte

Was ist hier falsch?

```
int& foo (int i)
{
    return i;
}
```

Rückgabewert vom Typ `int&` wird Alias des formalen Arguments, dessen Speicherdauer aber nach Auswertung des Funktionsaufrufes endet.

# Temporäre Objekte

Was ist hier falsch?

```
int& foo (int i)
{
    return i;
}
```

Rückgabewert vom Typ `int&` wird Alias des formalen Arguments, dessen Speicherdauer aber nach Auswertung des Funktionsaufrufes endet.

```
int k = 3;
int& j = foo (k); // j ist Alias einer "Leiche"
std::cout << j << "\n"; // undefined behavior
```

# Temporäre Objekte

Was ist hier falsch?

```
int& foo (int i)
{
    return i;
}
```



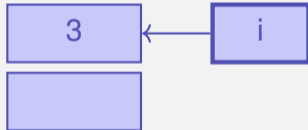
```
int k = 3;
int& j = foo (k); // j ist Alias einer "Leiche"
std::cout << j << "\n"; // undefined behavior
```

# Temporäre Objekte

Was ist hier falsch?

```
int& foo (int i)
{
    return i;
}
```

Wert des Aufrufarguments kommt  
auf den *Aufrufstapel*



```
int k = 3;
int& j = foo (k); // j ist Alias einer "Leiche"
std::cout << j << "\n"; // undefined behavior
```

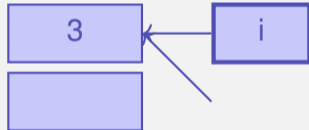


# Temporäre Objekte

Was ist hier falsch?

```
int& foo (int i)
{
    return i;
}
```

i wird als Referenz zurückgegeben



```
int k = 3;
int& j = foo (k); // j ist Alias einer "Leiche"
std::cout << j << "\n"; // undefined behavior
```

# Temporäre Objekte

Was ist hier falsch?

```
int& foo (int i)
{
    return i;
}
```

...und verschwindet vom Aufrufstapel



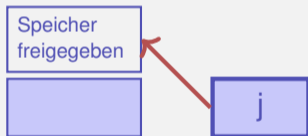
```
int k = 3;
int& j = foo (k); // j ist Alias einer "Leiche"
std::cout << j << "\n"; // undefined behavior
```

# Temporäre Objekte

Was ist hier falsch?

```
int& foo (int i)
{
    return i;
}
```

j wird Alias des freigegebenen Speichers



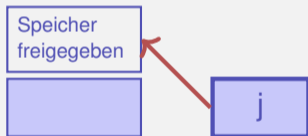
```
int k = 3;
int& j = foo (k); // j ist Alias einer "Leiche"
std::cout << j << "\n"; // undefined behavior
```

# Temporäre Objekte

Was ist hier falsch?

```
int& foo (int i)
{
    return i;
}
```

Wert von j wird ausgegeben



```
int k = 3;
int& j = foo (k); // j ist Alias einer "Leiche"
std::cout << j << "\n"; // undefined behavior
```

# Die Referenz-Richtlinie

## Referenz-Richtlinie

Wenn man eine Referenz erzeugt, muss das Objekt, auf das sie verweist, mindestens so lange „leben“ wie die Referenz selbst.

# Const-Referenzen

- haben Typ `const T &`
- Typ kann verstanden werden als „`(const T) &`“
- können auch mit R-Werten initialisiert werden (Compiler erzeugt temporäres Objekt ausreichender Lebensdauer)

# Const-Referenzen

- haben Typ `const T &`
- Typ kann verstanden werden als „`(const T) &`“
- können auch mit R-Werten initialisiert werden (Compiler erzeugt temporäres Objekt ausreichender Lebensdauer)

```
const T& r = lvalue;
```

r wird mit der Adresse von *lvalue* initialisiert (effizient)

# Const-Referenzen

- haben Typ `const T &`
- Typ kann verstanden werden als „`(const T) &`“
- können auch mit R-Werten initialisiert werden (Compiler erzeugt temporäres Objekt ausreichender Lebensdauer)

```
const T& r = rvalue;
```

`r` wird mit der Adresse eines temporären Objektes vom Wert des `rvalue` initialisiert (pragmatisch)



# Wann `const T&` ?

## Regel

Argumenttyp `const T&` (pass by *read-only* reference) wird aus Effizienzgründen anstatt `T` (pass by value) benutzt, wenn der Typ `T` grossen Speicherbedarf hat. Für fundamentale Typen (`int`, `double`,...) lohnt es sich aber nicht.

# Wann `const T&` ?

## Regel

Argumenttyp `const T&` (pass by *read-only* reference) wird aus Effizienzgründen anstatt `T` (pass by value) benutzt, wenn der Typ `T` grossen Speicherbedarf hat. Für fundamentale Typen (`int`, `double`,...) lohnt es sich aber nicht.

Beispiele folgen später in der Vorlesung


# Was genau ist konstant?

Betrachte L-Wert vom Typ `const T`

- Fall 1:  $T$  ist kein Referenztyp

Dann ist der L-Wert eine **Konstante**.

```
const int n = 5;  
int& i = n;  
i = 6;
```




# Was genau ist konstant?

Betrachte L-Wert vom Typ `const T`

- Fall 1:  $T$  ist kein Referenztyp

Dann ist der L-Wert eine **Konstante**.

```
const int n = 5;  
int& i = n; // error: const-qualification is discarded  
i = 6;
```



Der Schummelversuch wird vom Compiler erkannt

# Was genau ist konstant?

Betrachte L-Wert vom Typ `const T`

- Fall 2:  $T$  ist Referenztyp

Dann ist der L-Wert ein Lese-Alias, durch den der Wert dahinter nicht verändert werden darf.

# Was genau ist konstant?

Betrachte L-Wert vom Typ `const T`

## ■ Fall 2: `T` ist Referenztyp

Dann ist der L-Wert ein Lese-Alias, durch den der Wert dahinter nicht verändert werden darf.

```
int n = 5;
const int& i = n; // i: Lese-Alias von n
int& j = n;      // j: Lese-Schreib-Alias
i = 6;          // Fehler: i ist Lese-Alias
j = 6;          // ok: n bekommt Wert 6
```

# 12. Vektoren I

Vektoren, Sieb des Eratosthenes, Speicherlayout, Iteration

# Vektoren: Motivation

- Wir können jetzt über Zahlen iterieren

```
for (int i=0; i<n ; ++i) ...
```



# Vektoren: Motivation

- Wir können jetzt über Zahlen iterieren

```
for (int i=0; i<n ; ++i) ...
```

- .. aber noch nicht über Daten!

# Vektoren: Motivation

- Wir können jetzt über Zahlen iterieren

```
for (int i=0; i<n ; ++i) ...
```

- .. aber noch nicht über Daten!
- Vektoren speichern *gleichartige* Daten.

# Vektoren: erste Anwendung

Das Sieb des Erathostenes

- berechnet alle Primzahlen  $< n$

# Vektoren: erste Anwendung

## Das Sieb des Erathostenes

- berechnet alle Primzahlen  $< n$
- Methode: Ausstreichen der Nicht-Primzahlen

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

# Vektoren: erste Anwendung

## Das Sieb des Erathostenes

- berechnet alle Primzahlen  $< n$
- Methode: Ausstreichen der Nicht-Primzahlen

<b>2</b>	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
----------	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----

Streiche alle echten Vielfachen von 2 ...

# Vektoren: erste Anwendung

## Das Sieb des Erathostenes

- berechnet alle Primzahlen  $< n$
- Methode: Ausstreichen der Nicht-Primzahlen

<b>2</b>	3	<del>4</del>	5	<del>6</del>	7	<del>8</del>	9	<del>10</del>	11	<del>12</del>	13	<del>14</del>	15	<del>16</del>	17	<del>18</del>	19	<del>20</del>	21	<del>22</del>	23
----------	---	--------------	---	--------------	---	--------------	---	---------------	----	---------------	----	---------------	----	---------------	----	---------------	----	---------------	----	---------------	----

Streiche alle echten Vielfachen von 2 ...

# Vektoren: erste Anwendung

## Das Sieb des Erathostenes

- berechnet alle Primzahlen  $< n$
- Methode: Ausstreichen der Nicht-Primzahlen

2	3	<del>4</del>	5	<del>6</del>	7	<del>8</del>	9	<del>10</del>	11	<del>12</del>	13	<del>14</del>	15	<del>16</del>	17	<del>18</del>	19	<del>20</del>	21	<del>22</del>	23
---	---	--------------	---	--------------	---	--------------	---	---------------	----	---------------	----	---------------	----	---------------	----	---------------	----	---------------	----	---------------	----

... und gehe zur nächsten Zahl

# Vektoren: erste Anwendung

## Das Sieb des Erathostenes

- berechnet alle Primzahlen  $< n$
- Methode: Ausstreichen der Nicht-Primzahlen

2	3	<del>4</del>	5	<del>6</del>	7	<del>8</del>	9	<del>10</del>	11	<del>12</del>	13	<del>14</del>	15	<del>16</del>	17	<del>18</del>	19	<del>20</del>	21	<del>22</del>	23
---	---	--------------	---	--------------	---	--------------	---	---------------	----	---------------	----	---------------	----	---------------	----	---------------	----	---------------	----	---------------	----

Streiche alle echten Vielfachen von 3 ...



# Vektoren: erste Anwendung

## Das Sieb des Erathostenes

- berechnet alle Primzahlen  $< n$
- Methode: Ausstreichen der Nicht-Primzahlen

2	3	<del>4</del>	5	<del>6</del>	7	<del>8</del>	<del>9</del>	<del>10</del>	11	<del>12</del>	13	<del>14</del>	<del>15</del>	<del>16</del>	17	<del>18</del>	19	<del>20</del>	<del>21</del>	<del>22</del>	23
---	---	--------------	---	--------------	---	--------------	--------------	---------------	----	---------------	----	---------------	---------------	---------------	----	---------------	----	---------------	---------------	---------------	----

Streiche alle echten Vielfachen von 3 ...

# Vektoren: erste Anwendung

## Das Sieb des Erathostenes

- berechnet alle Primzahlen  $< n$
- Methode: Ausstreichen der Nicht-Primzahlen

2	3	<del>4</del>	5	<del>6</del>	7	<del>8</del>	<del>9</del>	<del>10</del>	11	<del>12</del>	13	<del>14</del>	<del>15</del>	<del>16</del>	17	<del>18</del>	19	<del>20</del>	<del>21</del>	<del>22</del>	23
---	---	--------------	---	--------------	---	--------------	--------------	---------------	----	---------------	----	---------------	---------------	---------------	----	---------------	----	---------------	---------------	---------------	----

... und gehe zur nächsten Zahl

# Vektoren: erste Anwendung

## Das Sieb des Erathostenes

- berechnet alle Primzahlen  $< n$
- Methode: Ausstreichen der Nicht-Primzahlen

2	3	<del>4</del>	5	<del>6</del>	7	<del>8</del>	<del>9</del>	<del>10</del>	11	<del>12</del>	13	<del>14</del>	<del>15</del>	<del>16</del>	17	<del>18</del>	19	<del>20</del>	<del>21</del>	<del>22</del>	23
---	---	--------------	---	--------------	---	--------------	--------------	---------------	----	---------------	----	---------------	---------------	---------------	----	---------------	----	---------------	---------------	---------------	----

Am Ende des Streichungsprozesses bleiben nur die Primzahlen übrig.

# Vektoren: erste Anwendung

## Das Sieb des Erathostenes

- berechnet alle Primzahlen  $< n$
- Methode: Ausstreichen der Nicht-Primzahlen

2	3	<del>4</del>	5	<del>6</del>	7	<del>8</del>	<del>9</del>	<del>10</del>	11	<del>12</del>	13	<del>14</del>	<del>15</del>	<del>16</del>	17	<del>18</del>	19	<del>20</del>	<del>21</del>	<del>22</del>	23
---	---	--------------	---	--------------	---	--------------	--------------	---------------	----	---------------	----	---------------	---------------	---------------	----	---------------	----	---------------	---------------	---------------	----

- Frage: wie streichen wir Zahlen aus ??

# Vektoren: erste Anwendung

## Das Sieb des Erathostenes

- berechnet alle Primzahlen  $< n$
- Methode: Ausstreichen der Nicht-Primzahlen

2	3	<del>4</del>	5	<del>6</del>	7	<del>8</del>	<del>9</del>	<del>10</del>	11	<del>12</del>	13	<del>14</del>	<del>15</del>	<del>16</del>	17	<del>18</del>	19	<del>20</del>	<del>21</del>	<del>22</del>	23
---	---	--------------	---	--------------	---	--------------	--------------	---------------	----	---------------	----	---------------	---------------	---------------	----	---------------	----	---------------	---------------	---------------	----

- Frage: wie streichen wir Zahlen aus ??
- Antwort: mit einem *Vektor*.

# Eratosthenes mit Vektoren: Initialisierung


...

```
#include <vector>
```

...

```
std::vector<bool> crossed_out (n, false);
```

Initialisierung mit  $n$  Elementen  
Initialwert `false`.



Elementtyp, in spitzen Klammern



# Eratosthenes mit Vektoren: Berechnung

```
for (unsigned int i = 2; i < crossed_out.size(); ++i)
    if (!crossed_out[i]) { // i is prime
        std::cout << i << " ";
        // cross out all proper multiples of i
        for (unsigned int m = 2*i; m < n; m += i)
            crossed_out[m] = true;
    }
```

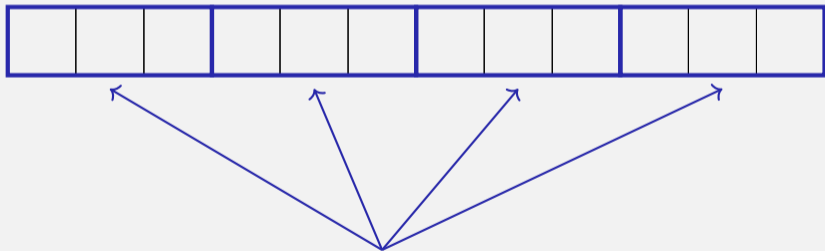
# Speicherlayout eines Vektors

- Ein Vector belegt einen *zusammenhängenden* Speicherbereich



# Speicherlayout eines Vektors

- Ein Vector belegt einen *zusammenhängenden* Speicherbereich

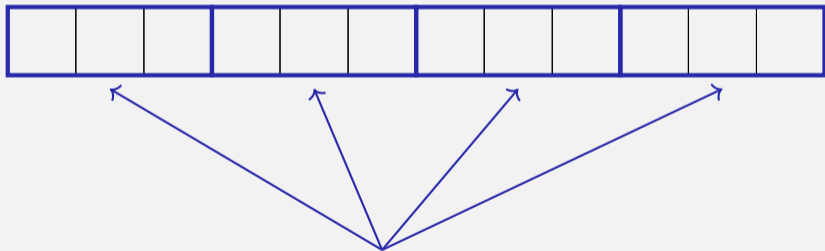


Speicherzellen für jeweils einen Wert vom Typ  $T$

# Speicherlayout eines Vektors

- Ein Vector belegt einen *zusammenhängenden* Speicherbereich

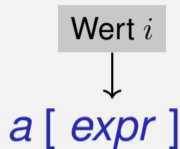
Beispiel: ein Vektor mit 4 Elementen



Speicherzellen für jeweils einen Wert vom Typ  $T$

# Wahlfreier Zugriff (Random Access)

Der L-Wert



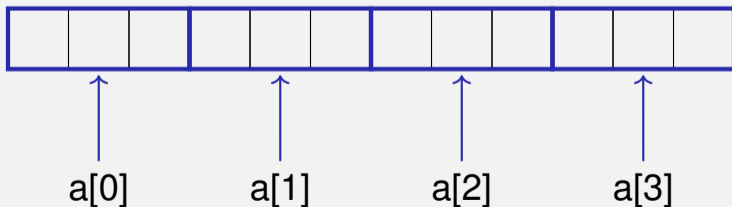
hat Typ  $T$  und bezieht sich auf das  $i$ -te Element des Vektors  $a$   
(Zählung ab 0!)

# Wahlfreier Zugriff (Random Access)

Der L-Wert



hat Typ  $T$  und bezieht sich auf das  $i$ -te Element des Vektors  $a$   
(Zählung ab 0!)



# Wahlfreier Zugriff (Random Access)

$a [ expr ]$

- Der Wert  $i$  von  $expr$  heisst *Index*
- $[]$ : Subskript-Operator
- $a[expr]$  Ist ein L-Wert

# Wahlfreier Zugriff (Random Access)

- Wahlfreier Zugriff ist sehr effizient:

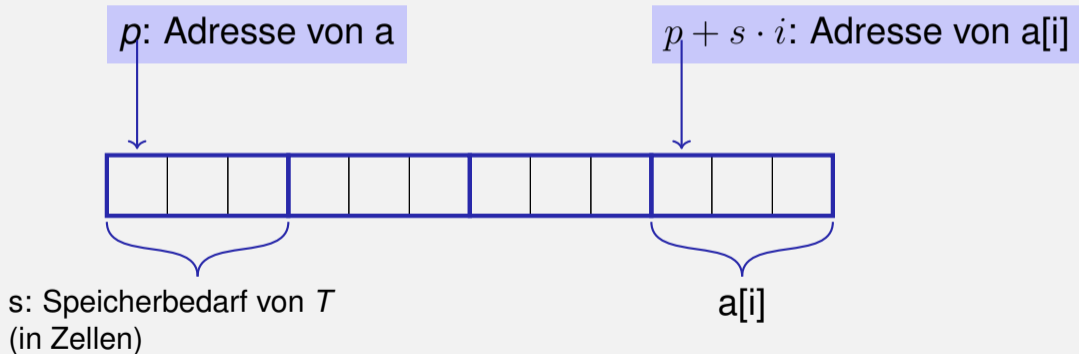
$p$ : Adresse von  $a$ , d.h. Adresse der ersten Speicherzelle



$s$ : Speicherbedarf von  $T$   
(in Zellen)

# Wahlfreier Zugriff (Random Access)

- Wahlfreier Zugriff ist sehr effizient:



# Vektor Initialisierung

- `std::vector<int> a (5);`

Die 5 Elemente von a werden mit null initialisiert



# Vektor Initialisierung

- `std::vector<int> a (5);`  
Die 5 Elemente von a werden mit null initialisiert
- `std::vector<int> a (5, 2);`  
Die 5 Elemente von a werden mit 2 initialisiert.
- `std::vector<int> a {4, 3, 5, 2, 1};`  
Der Vektor wird mit einer *Initialisierungsliste* initialisiert.
- `std::vector<int> a;`  
Ein leerer Vektor wird erstellt.

# Vektor Initialisierung

- `std::vector<int> a (5);`  
Die 5 Elemente von a werden mit null initialisiert
- `std::vector<int> a (5, 2);`  
Die 5 Elemente von a werden mit 2 initialisiert.
- `std::vector<int> a {4, 3, 5, 2, 1};`  
Der Vektor wird mit einer *Initialisierungsliste* initialisiert.
- `std::vector<int> a;`  
Ein leerer Vektor wird erstellt.

# Vektor Initialisierung

- `std::vector<int> a (5);`  
Die 5 Elemente von a werden mit null initialisiert
- `std::vector<int> a (5, 2);`  
Die 5 Elemente von a werden mit 2 initialisiert.
- `std::vector<int> a {4, 3, 5, 2, 1};`  
Der Vektor wird mit einer *Initialisierungsliste* initialisiert.
- `std::vector<int> a;`  
Ein leerer Vektor wird erstellt.

# Achtung

- Der Zugriff auf Elemente ausserhalb der gültigen Grenzen eines Vektors führt zu undefiniertem Verhalten.

```
std::vector arr (10);  
for (int i=0; i<=10; ++i)  
    arr[i] = 30;
```

# Achtung

- Der Zugriff auf Elemente ausserhalb der gültigen Grenzen eines Vektors führt zu undefiniertem Verhalten.


```
std::vector arr (10);  
for (int i=0; i<=10; ++i)  
    arr[i] = 30; // Laufzeit-Fehler: Zugriff auf arr[10]!
```

# Achtung

## Prüfung der Indexgrenzen

Bei Verwendung des Indexoperators auf einem Vektor ist es die alleinige *Verantwortung des Programmierers*, die Gültigkeit aller Elementzugriffe zu prüfen.

# Konsequenzen illegaler Index-Zugriffe



[Alle](#) [Videos](#) [Bilder](#) [News](#) [Shopping](#) [Mehr](#) [Einstellungen](#) [Tools](#)

Ungefähr 127'000 Ergebnisse (0.30 Sekunden)

**CWE - CWE-125: Out-of-bounds Read (3.0)**  
<https://cwe.mitre.org> › [CWE List](#) ▼ [Diese Seite übersetzen](#)  
However, this method only verifies that the given array index is less than the maximum length of the array but does not check for the minimum value (CWE-839). This will allow a negative value to be accepted as the input array index, which will result in a **out of bounds** read (CWE-125) and may allow access to sensitive ...

**CWE - CWE-787: Out-of-bounds Write (3.0)**  
<https://cwe.mitre.org> › [CWE List](#) ▼ [Diese Seite übersetzen](#)  
This typically occurs when the pointer or its index is incremented or decremented to a position beyond the bounds of the buffer or when pointer arithmetic results in a position outside of the valid memory location to name a few. This may result in corruption of sensitive information, a crash, or code execution among other ...

**c - How dangerous is it to access an array out of bounds? - Stack ...**  
<https://stackoverflow.com/.../how-dangerous-is-it-to-access-an-arr...> ▼ [Diese Seite übersetzen](#)  
As far as the ISO C standard (the official definition of the language) is concerned, accessing an array outside its bounds has "undefined behavior". The literal meaning of this is: behavior, upon use of a nonportable or erroneous program construct or of erroneous data, for which this International Standard imposes no ...

**Bypassing ASLR with CVE-2015-0071: An Out-of-Bounds Read ...**  
<https://blog.trendmicro.com/.../bypassing-aslr-with-cve-2015-007...> ▼ [Diese Seite übersetzen](#)  
Bypassing ASLR with CVE-2015-0071: An **Out-of-Bounds** Read Vulnerability. Posted on: March 13, 2015 at 8:10 am ... 0x00: pVtable: 0x08: lenath (the strina lenath): 0x0c: oStrina (pointer to the strina

# Konsequenzen illegaler Index-Zugriffe

The screenshot shows the Exploit Database search results for the query 'out-of-bounds'. The page lists various exploits with their dates, descriptions, and associated platforms. The search results are displayed in a table format with columns for date, status, description, platform, and author.

Date	Status	Description	Platform	Author
2018-03-23	🟢	Android Bluetooth - BNEP BNEP_SETUP_CONNECTION_REQUEST_MSG <b>Out-of-Bounds</b> Read	Android	QuarksLab
2018-03-06	🟢	Chrome V8 JIT - Empty BytecodeJumpTable <b>Out-of-Bounds</b> Read	Multiple	Google...
2018-02-15	🟢	Pdfium - <b>Out-of-Bounds</b> Read with Shading Pattern Backed by Pattern Colorspace	Multiple	Google...
2018-01-17	🟢	Microsoft Edge Chakra - 'AsmJSByteCodeGenerators:EmitCall' <b>Out-of-Bounds</b> Read	Windows	Google...
2018-01-17	🟢	Microsoft Edge Chakra JIT - <b>Out-of-Bounds</b> Write	Windows	Google...
2018-01-11	🟢	Microsoft Edge Chakra - 'AppendLeftOverItemsFromEndSegment' <b>Out-of-Bounds</b> Read	Windows	Google...
2018-01-09	🟢	Microsoft Edge Chakra - 'asm.js' <b>Out-of-Bounds</b> Read	Windows	Google...
2017-12-19	🟢	Microsoft Windows - 'jscrip!RegExpFncObj::LastParen' <b>Out-of-Bounds</b> Read	Windows	Google...
2017-11-22	🟢	WebKit - 'WebCore::SVGPatternElement::collectPatternAttributes' <b>Out-of-Bounds</b> Read	Multiple	Google...
2017-11-22	🟢	WebKit - 'WebCore::SimpleLineLayout::RunResolver::runForPoint' <b>Out-of-Bounds</b> Read	Multiple	Google...
2017-11-22	🟢	WebKit - 'WebCore::RenderText::localCaretRect' <b>Out-of-Bounds</b> Read	Multiple	Google...
2017-09-25	🟢	Apple iOS 10.2 - Broadcom <b>Out-of-Bounds</b> Write when Handling 802.11k Neighbor Report...	iOS	Google...
2017-09-25	🟢	Adobe Flash - <b>Out-of-Bounds</b> Read in applyToRange	Multiple	Google...
2017-09-25	🟢	Adobe Flash - <b>Out-of-Bounds</b> Write in MP4 Edge Processing	Multiple	Google...
2017-09-25	🟢	Adobe Flash - <b>Out-of-Bounds</b> Memory Read in MP4 Parsing	Multiple	Google...
2017-09-19	🟢	Microsoft Edge 38.14393.1066.0 - 'COptionsCollectionCacheItem::GetAt' <b>Out-of-Bounds</b> Read	Windows	Google...
2017-09-18	🟢	Microsoft Windows Kernel - 'win32k.sys'.TTF Font Processing <b>Out-of-Bounds</b> Read with...	Windows	Google...
2017-09-18	🟢	Microsoft Windows Kernel - 'win32k.sys'.TTF Font Processing <b>Out-of-Bounds</b> Reads/Writes...	Windows	Google...
2017-09-06	🟢	Jungo DriverWizard WinDriver < 12.4.0 - Kernel <b>Out-of-Bounds</b> Write Privilege Escalation	Windows	mr_me
2017-08-17	🟢	Microsoft Edge - <b>Out-of-Bounds</b> Access when Fetching Source	Windows	Google...
2017-08-17	🟢	Adobe Flash - Invoke Accesses Trait <b>Out-of-Bounds</b>	Windows	Google...
2017-08-16	🟢	Microsoft Edge 38.14393.1066.0 - 'CinputDateTImeScrollerElement::_SelectValueInternal'...	Windows	Google...
2017-07-06	🟢	LibTIFF - 'TIFFGetField (tifffsplit)' <b>Out-of-Bounds</b> Read	Linux	zhangtan

Search filters: out-of-bounds | Highlight All | Match Case | Whole Words | 116 of 116 matches



# Vektoren bieten Komfort

```
std::vector<int> v (10);  
v.at(5) = 3; // with bound check  
v.push_back(8); // 8 is appended  
std::vector<int> w = v; // w is initialized with v  
int sz = v.size(); // sz = 11
```

# 13. Zeichen und Texte I

Zeichen und Texte, ASCII, UTF-8, Caesar-Code

# Zeichen und Texte

- Texte haben wir schon gesehen:

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

# Zeichen und Texte

- Texte haben wir schon gesehen:

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

String-Literal

# 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?

# 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: `std::string`  $\approx$  Vektor von `char` Elementen

# Der Typ `char` („character“)

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

# 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'



# 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`

# Der Typ `char` („character“)

ist formal ein ganzzahliger Typ

- Werte konvertierbar nach `int` / `unsigned int`
- Werte belegen meistens 8 Bit

# Der Typ `char` („character“)

ist formal ein ganzzahliger Typ

- Werte konvertierbar nach `int` / `unsigned int`
- Werte belegen meistens 8 Bit

# Der Typ `char` („character“)

ist formal ein ganzzahliger Typ

- Werte konvertierbar nach `int` / `unsigned int`
- Werte belegen meistens 8 Bit

Wertebereich:

$\{-128, \dots, 127\}$  oder  $\{0, \dots, 255\}$

# Der ASCII-Code

- definiert konkrete Konversionsregeln  
`char`  $\longrightarrow$  `int` / `unsigned int`

# Der ASCII-Code

- definiert konkrete Konversionsregeln

`char`  $\longrightarrow$  `int` / `unsigned int`

- wird von fast allen Plattformen benutzt

Zeichen  $\longrightarrow$   $\{0, \dots, 127\}$

'A', 'B', ... , 'Z'  $\longrightarrow$  65, 66, ..., 90

'a', 'b', ... , 'z'  $\longrightarrow$  97, 98, ..., 122

'0', '1', ... , '9'  $\longrightarrow$  48, 49, ..., 57

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

`std::cout << c;`

`abcdefghijklmnopqrstuvwxy`

# Der ASCII-Code

- definiert konkrete Konversionsregeln  
`char`  $\longrightarrow$  `int` / `unsigned int`
- wird von fast allen Plattformen benutzt

Zeichen  $\longrightarrow$   $\{0, \dots, 127\}$

'A', 'B', ... , 'Z'  $\longrightarrow$  65, 66, ..., 90

'a', 'b', ... , 'z'  $\longrightarrow$  97, 98, ..., 122

'0', '1', ... , '9'  $\longrightarrow$  48, 49, ..., 57

- ```
for (char c = 'a'; c <= 'z'; ++c)
    std::cout << c;
```

abcdefghijklmnopqrstuvwxyz



# Erweiterung von ASCII: UTF-8

- Internationalisierung von Software  $\Rightarrow$  grosse Zeichensätze nötig.  
Heute üblich: Unicode, 100 Schriftsysteme, 110000 Zeichen.

# Erweiterung von ASCII: UTF-8

- Internationalisierung von Software  $\Rightarrow$  grosse Zeichensätze nötig.  
Heute üblich: Unicode, 100 Schriftsysteme, 110000 Zeichen.
- ASCII kann mit 7 Bits codiert werden. Ein achttes Bit ist verwendbar

# Erweiterung von ASCII: UTF-8

- Internationalisierung von Software  $\Rightarrow$  grosse Zeichensätze nötig. Heute üblich: Unicode, 100 Schriftsysteme, 110000 Zeichen.
- ASCII kann mit 7 Bits codiert werden. Ein achttes Bit ist verwendbar, um weitere 128 Zeichen zu codieren – das ist aber Geschichte

# Erweiterung von ASCII: UTF-8

- Internationalisierung von Software  $\Rightarrow$  grosse Zeichensätze nötig. Heute üblich: Unicode, 100 Schriftsysteme, 110000 Zeichen.
- ASCII kann mit 7 Bits codiert werden. Ein achttes Bit ist verwendbar, um das Vorkommen weiterer Bits festzulegen.

| Bits | Encoding                                              |
|------|-------------------------------------------------------|
| 7    | 0xxxxxxx                                              |
| 11   | 110xxxxx 10xxxxxx                                     |
| 16   | 1110xxxx 10xxxxxx 10xxxxxx                            |
| 21   | 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx                   |
| 26   | 111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx          |
| 31   | 1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx |

# Erweiterung von ASCII: UTF-8

- Internationalisierung von Software  $\Rightarrow$  grosse Zeichensätze nötig. Heute üblich: Unicode, 100 Schriftsysteme, 110000 Zeichen.
- ASCII kann mit 7 Bits codiert werden. Ein achttes Bit ist verwendbar, um das Vorkommen weiterer Bits festzulegen.





| Bits | Encoding                                              |
|------|-------------------------------------------------------|
| 7    | 0xxxxxxx                                              |
| 11   | 110xxxxx 10xxxxxx                                     |
| 16   | 1110xxxx 10xxxxxx 10xxxxxx                            |
| 21   | 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx                   |
| 26   | 111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx          |
| 31   | 1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx |

# Einige Zeichen in UTF-8





| Symbol | Codierung (jeweils 16 Bit) |
|--------|----------------------------|
|--------|----------------------------|

|   |                            |
|---|----------------------------|
| س | 11101111 10101111 10111001 |
|---|----------------------------|

# Einige Zeichen in UTF-8

| Symbol                                                                            | Codierung (jeweils 16 Bit) |
|-----------------------------------------------------------------------------------|----------------------------|
|  | 11101111 10101111 10111001 |
|  | 11100010 10011000 10100000 |
|  | 11100010 10011000 10000011 |
|  | 11100010 10011000 10011001 |

# Einige Zeichen in UTF-8

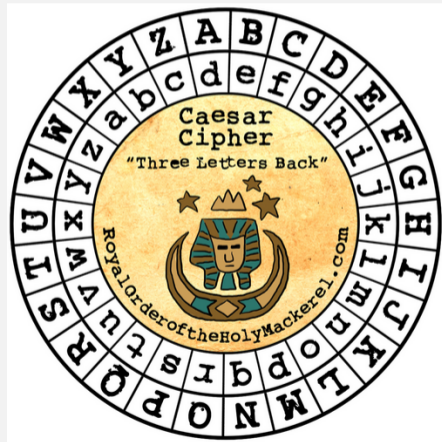
| Symbol                                                                            | Codierung (jeweils 16 Bit) |
|-----------------------------------------------------------------------------------|----------------------------|
|  | 11101111 10101111 10111001 |
|  | 11100010 10011000 10100000 |
|  | 11100010 10011000 10000011 |
|  | 11100010 10011000 10011001 |
| A                                                                                 | 01000001                   |



# Caesar-Code

Ersetze jedes druckbare Zeichen in einem Text durch seinen Vor-Vor-Vorgänger.

|     |       |   |     |       |
|-----|-------|---|-----|-------|
| ' ' | (32)  | → | ' ' | (124) |
| '!' | (33)  | → | '}' | (125) |
|     | ⋮     |   |     |       |
| 'D' | (68)  | → | 'A' | (65)  |
| 'E' | (69)  | → | 'B' | (66)  |
|     | ⋮     |   |     |       |
| ~   | (126) | → | '{' | (123) |



```
// pre: divisor > 0
// post: return the remainder of dividend / divisor
//       with 0 <= result < divisor
int mod(int dividend, int divisor);

// POST: if c is one of the 95 printable ASCII characters, c is
//       cyclically shifted s printable characters to the right
char shift(char c, int s) {
    if (c >= 32 && c <= 126) { // c printable
        c = 32 + mod(c - 32 + s, 95);
    }
    return c;
}
```

```
// pre: divisor > 0
// post: return the remainder of dividend / divisor
//       with 0 <= result < divisor
int mod(int dividend, int divisor);

// POST: if c is one of the 95 printable ASCII characters, c is
//       cyclically shifted s printable characters to the right
char shift(char c, int s) {
    if (c >= 32 && c <= 126) { // c printable
        c = 32 + mod(c - 32 + s,95)};
    }
    return c;
}
```

"- 32" transforms interval [32, 126] to [0, 94]

"32 +" transforms interval [0, 94] back to [32, 126]

mod(x,95) is the representative of  $x \pmod{95}$  in interval [0, 94]

```
// POST: Each character read from std::cin was shifted cyclically
//       by s characters and afterwards written to std::cout
void caesar(int s) {
    std::cin >> std::noskipws; // #include <ios>

    char next;
    while (std::cin >> next) {
        std::cout << shift(next, s);
    }
}
```

Leerzeichen und Zeilenumbrüche  
sollen *nicht* ignoriert werden

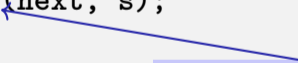
```
// POST: Each character read from std::cin was shifted cyclically
//       by s characters and afterwards written to std::cout
void caesar(int s) {
    std::cin >> std::noskipws; // #include <ios>

    char next;
    while (std::cin >> next) ← {
        std::cout << shift(next, s),
    }
}
```

Konversion nach bool: liefert *false* genau dann, wenn die Eingabe leer ist.

```
// POST: Each character read from std::cin was shifted cyclically
//       by s characters and afterwards written to std::cout
void caesar(int s) {
    std::cin >> std::noskipws; // #include <ios>

    char next;
    while (std::cin >> next) {
        std::cout << shift(next, s);
    }
}
```



Verschiebt nur druckbare Zeichen.

```
int main() {  
    int s;  
    std::cin >> s;  
  
    // Shift input by s  
    caesar(s);  
  
    return 0;  
}
```

Verschlüsseln: Verschiebung um  $n$  (hier: 3)

```
3.  
Hello World, my password is 1234.  
Khoor#Zruog/#p|#sdvvzrug#lv#45671
```

Entschlüsseln: Verschiebung um  $-n$  (hier: -3)

```
-3.  
Khoor#Zruog/#p|#sdvvzrug#lv#45671  
Hello World, my password is 1234.
```

# Caesar-Code: Generalisierung

```
void caesar(int s) {  
    std::cin >> std::noskipws;  
  
    char next;  
    while (std::cin >> next) {  
        std::cout << shift(next, s);  
    }  
}
```

- Momentan nur von `std::cin`  
nach `std::cout`



# Caesar-Code: Generalisierung

```
void caesar(int s) {  
    std::cin >> std::noskipws;  
  
    char next;  
    while (std::cin >> next) {  
        std::cout << shift(next, s);  
    }  
}
```

- Momentan nur von `std::cin` nach `std::cout`

- Besser: von beliebiger Zeichenquelle (Konsole, Datei, ...) zu beliebiger Zeichensenke (Konsole, ...)

