

Von C zu

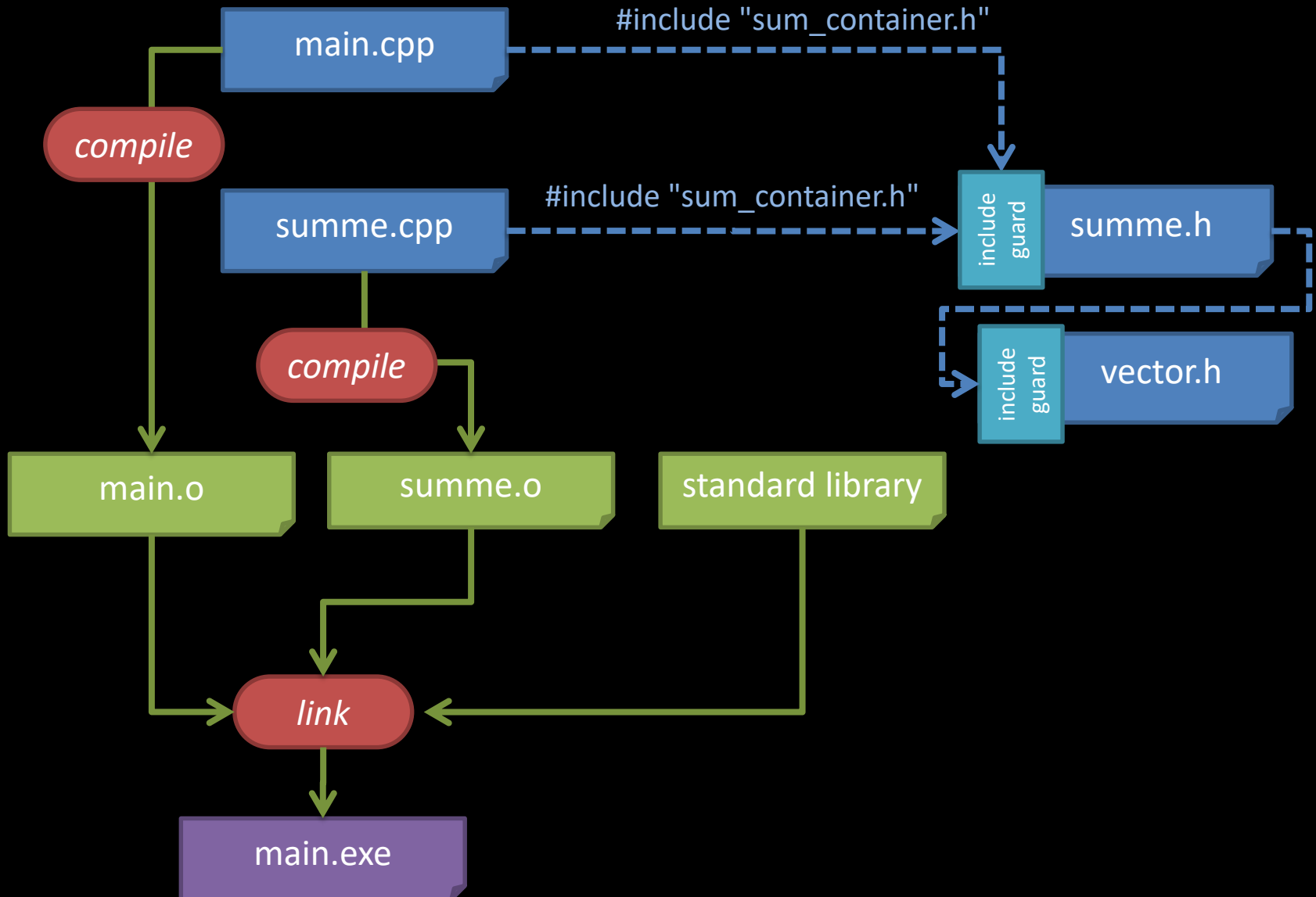


Typische Fehler und Verbesserungsmöglichkeiten

Teil 1:

Anmerkungen zur Aufgabe Summe








„Summe“ übersetzen (vereinfacht)



Die kleinen
Dinge beachten!



Typische Fehler bei der Bearbeitung der Übung „Summe“

Name	Date modified	Type	Size
 bin	11.04.2015 20:31	File folder	
 obj	11.04.2015 20:31	File folder	
 main.cpp	10.04.2015 12:11	CPP File	1 KB
 summe.cbp	10.04.2015 12:04	CBP File	2 KB
 summe.cpp	10.04.2015 12:11	CPP File	1 KB
 summe.depend	10.04.2015 12:11	DEPEND File	1 KB
 SUMME.h	10.04.2015 12:11	H File	1 KB

```
#include <iostream>
#include <vector>
#include "summe.h"
#include <string>
```

```
using namespace std;
```

```
int main() {
```

Groß- / Kleinschreibung beachten!

Name	Date modified	Type	Size
bin	11.04.2015 20:31	File folder	
obj	11.04.2015 20:31	File folder	
main.cpp	10.04.2015 12:11	CPP File	1 KB
summe.cbp	10.04.2015 12:04	CBP File	2 KB
summe.cpp	10.04.2015 12:11	CPP File	1 KB
summe.depend	10.04.2015 12:11	DEPEND File	1 KB
SUMME.h	10.04.2015 12:11	H File	1 KB

```
#include <iostream>
#include <vector>
#include "summe.h"
#include <string>
```

```
using namespace std;
```

```
int main() {
```

Achtung, das funktioniert nur bei Windows
Immer Groß- und Kleinschreibung beachten!

Header Files



Verbesserungen: Zeile 1-4

```
1  #include <vector>
2  #include <iostream>
3  #ifndef __SUMME_H
4  #define __SUMME_H
5
6  using namespace std;
7
8  extern float berechneSumme(vector<float> v, int anzahl);
9
10 #endif // !__SUMME_H
11
```

#includes ...

immer nach Include-Guards schreiben!

Alle #include ... kommen nach den include-guards!

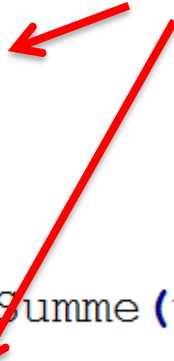
```
1  #include <vector>
2  #include <iostream>
3  #ifndef __SUMME_H
4  #define __SUMME_H
5
6  using namespace std;
7
8  extern float berechneSumme(vector<float> v, int anzahl);
9
10 #endif // !__SUMME_H
11
```


#includes ...

immer nach Include-Guards schreiben!

```
1  #include <vector>
2  #include <iostream>
3  #ifndef __SUMME_H
4  #define __SUMME_H
5
6  using namespace std;
7
8  extern float berechneSumme(vector<float> v, int anzahl);
9
10 #endif // !__SUMME_H
11
```

Alle #include ... kommen nach den include-guards!



#includes ...

immer nach Include-Guards schreiben!

```
#ifndef __SUMME_H  
#define __SUMME_H
```

```
#include <iostream>  
#include <vector>
```

Alle #include ... kommen nach den include-guards!

```
using namespace std;
```

```
extern float berechneSumme(vector<float> v, int anzahl);
```

```
#endif // !__SUMME_H
```

Typische Fehler bei der Bearbeitung der Übung „Summe“

```
#ifndef __SUMME_H
#define __SUMME_H

#include <iostream>
#include <vector>

using namespace std;

extern float berechneSumme(vector<float> v, int anzahl);

#endif // !__SUMME_H
```

nur inkludieren,
was man wirklich braucht!

Nur das inkludieren, was man wirklich braucht!

```
#ifndef __SUMME_H
#define __SUMME_H

#include <iostream>
#include <vector>

using namespace std;

extern float berechneSumme(vector<float> v, int anzahl);

#endif // !__SUMME_H
```

nur inkludieren,
was man wirklich braucht!

Nur das inkludieren, was man wirklich braucht!

```
1 #ifndef __SUMME_H
2 #define __SUMME_H
3
4 #include <vector>
5
6 using namespace std;
7
8 extern float berechneSumme(vector<float> v, int anzahl);
9
10 #endif // !__SUMME_H
11
```

Tip: Forward Declarations anwenden, wenn möglich!

Typische Fehler bei der Bearbeitung der Übung „Summe“

```
1 #ifndef __SUMME_H
2 #define __SUMME_H
3
4 #include <vector>
5
6 using namespace std;
7
8 extern float berechneSumme(vector<float> v, int anzahl);
9
10 #endif // !__SUMME_H
11
```

kein using namespace ...
in den header-files

Kein using namespace in Header-Files !

```
1  #ifndef __SUMME_H
2  #define __SUMME_H
3
4  #include <vector>
5
6  using namespace std;
7
8  extern float berechneSumme(vector<float> v, int anzahl);
9
10 #endif // !__SUMME_H
11
```


kein using namespace ...
in den header-files,
sondern namespace:: Notation verwenden

Using namespace in Header-Files „verschmutzt“ die Files, die diesen Header inkludieren mit einem ungewollten using namespace!

Kein using namespace in Header-Files ! Stattdessen namespace:: verwenden!

```
1  □ #ifndef __SUMME_H
2    #define __SUMME_H
3
4    #include <vector>
5
6    extern float berechneSumme (std::vector<float> v, int anzahl);
7
8    #endif // !__SUMME_H
9
```

kein using namespace ...
in den header-files,
sondern namespace:: Notation verwenden



Verbesserungen: Zeile 6

```
1  □ #ifndef __SUMME_H
2    #define __SUMME_H
3
4    #include <vector>
5
6    extern float berechneSumme(std::vector<float> v, int anzahl);
7
8    #endif // !__SUMME_H
9
```

int anzahl : unnötig!

```
1 #ifndef __SUMME_H
2 #define __SUMME_H
3
4 #include <vector>
5
6 extern float berechneSumme(std::vector<float> v, int anzahl);
7
8 #endif // !__SUMME_H
9
```

int anzahl : unnötig!
std::vector kennt seine Größe

```
1  □ #ifndef __SUMME_H
2    #define __SUMME_H
3
4    #include <vector>
5
6    extern float berechneSumme(std::vector<float> v, int anzahl);
7
8    #endif // !__SUMME_H
9
```

std::vector kennt die Anzahl seiner Elemente
mit std::vector.size() kann sie abgefragt werden.
Also ist die Übergabe von anzahl unnötig!

weiter Zeile 6:

```
1  □ #ifndef __SUMME_H
2    #define __SUMME_H
3
4    #include <vector>
5
6    extern float berechneSumme(std::vector<float> v) ;
7
8    #endif // !__SUMME_H
9
```

Typischer C-Style bei der Bearbeitung der Übung „Summe“

```
1  #ifndef __SUMME_H
2  #define __SUMME_H
3
4  #include <vector>
5
6  extern float berechneSumme(std::vector<float> v) ;
7
8  #endif // !__SUMME_H
9
```

„extern“ ist für Funktionen optional
und in C++ eher unüblich.
Wir lassen es einfach weg!

hier Beispiel: check_extern zeigen!

The **extern** Storage Class

The **extern** storage class is used to give a reference of a global variable that is visible to ALL the program files.

When you use 'extern' the variable cannot be initialized as all it does is point the variable name at a storage location that has been previously defined.

When you have multiple files and you define a global variable or function, which will be used in other files also, then *extern* will be used in another file to give reference of defined variable or function. Just for understanding *extern* is used to declare a global variable or function in another file.

The extern modifier is most commonly used when there are two or more files sharing the same global variables or functions as explained below.

weiter Zeile 6:

```
1  #ifndef __SUMME_H
2  #define __SUMME_H
3
4  #include <vector>
5
6  float berechneSumme(std::vector<float> v);
7
8  #endif // !__SUMME_H
9
```

Kopien sind u.U. sehr „teuer“

```
1 #ifndef __SUMME_H
2 #define __SUMME_H
3
4 #include <vector>
5
6 float berechneSumme (std::vector<float> v) ;
7
8 #endif // !__SUMME_H
9
```

Wenn wir vector so übergeben wird eine Kopie erzeugt und übergeben - das kann „teuer“ werden!

Besser wir übergeben eine const-reference !

Call by value macht Kopie

Call by reference übergibt Bezug auf Original

```
1 #ifndef __SUMME_H
2 #define __SUMME_H
3
4 #include <vector>
5
6 float berechneSumme(const std::vector<float>& v);
7
8 #endif // !__SUMME_H
9
```

Besser wir übergeben eine const-reference!
const, da in vector keine Elemente
hinzufügen oder löschen wollen

Reference &, da wir keine Kopie brauchen,
wir wollen die Element nur lesen!

weiter Zeile 6:

```
1  □ #ifndef __SUMME_H
2    #define __SUMME_H
3
4    #include <vector>
5
6    float berechneSumme(const std::vector<float>& v) ;
7
8    #endif // !__SUMME_H
9
```

Naming: Wähle treffende Bezeichner!

```
1 #ifndef __SUMME_H
2 #define __SUMME_H
3
4 #include <vector>
5
6 float berechneSumme(const std::vector<float>& v);
7
8 #endif // !__SUMME_H
9
```

Der Benutzer der Funktionsschnittstelle sollte schnell erkennen können, wie die Funktion zu benutzen ist!

Naming: Wähle treffende Bezeichner!

```
1 #ifndef __SUMME_H
2 #define __SUMME_H
3
4 #include <vector>
5
6 float berechneSumme(const std::vector<float>& summanden);
7
8 #endif // !__SUMME_H
9
```

Wir ersetzen v durch summanden.



Schon ganz gut jetzt...
... wir machen hier später weiter.

```
1  #ifndef __SUMME_H
2  #define __SUMME_H
3
4  #include <vector>
5
6  float berechneSumme(const std::vector<float>& summanden) ;
7
8  #endif // !__SUMME_H
9
```



Let's have a look
to your cpp files!

Zeile 1 bis 3:

```
1  #include <vector>
2  #include "summe.h"
3  #include <iostream>
4
5  int berechneSumme(vector<int> &daten, int &N)
6  {
7      int summe = 0;
8      for (int k=0; k <= N; k++)
9      {
10         summe += daten[k];
11     }
12     return summe;
13 }
```

Regel 1:

Immer eigenes Header-File zuerst inkludieren!

```
1  #include <vector>
2  #include "summe.h"
3  #include <iostream>
4
5  int berechneSumme(vector<int> &daten, int &N)
6  {
7      int summe = 0;
8      for (int k=0; k <= N; k++)
9      {
10         summe += daten[k];
11     }
12     return summe;
13 }
```

Using namespace in Header-Files „verschmutzt“ die Files, die diesen Header inkludieren mit einem ungewollten using namespace!

Regel 2:

Nur inkludieren, was man wirklich braucht!

```
1  #include "summe.h"
2  #include <vector>
3  #include <iostream>
4
5  int berechneSumme(vector<int> &daten, int &N)
6  {
7      int summe = 0;
8      for (int k=0; k <= N; k++)
9      {
10         summe += daten[k];
11     }
12     return summe;
13 }
```

Das führt zu sogenannten self-contained headers, alles was der Header braucht wird auch inkludiert und nur das.

Zeile 3-11 : Wo ist der Bug?

```
1  #include "summe.h"
2
3  int berechneSumme(vector<int> &daten, int &N)
4  {
5      int summe = 0;
6      for (int k=0; k <= N; k++)
7      {
8          summe += daten[k];
9      }
10     return summe;
11 }
```



Und was macht der Bug bei der Ausführung?

k <= N iteriert zu weit!

```
1  #include "summe.h"
2
3  int berechneSumme(vector<int> &daten, int &N)
4  {
5      int summe = 0;
6      for (int k=0; k <= N; k++)
7      {
8          summe += daten[k];
9      }
10     return summe;
11 }
```

Und was macht der Bug bei der Ausführung?

k <= N iteriert zu weit!

```
1  #include "summe.h"
2
3  int berechneSumme(vector<int> &daten, int &N)
4  {
5      int summe = 0;
6      for (int k=0; k <= N; k++)
7      {
8          summe += daten[k];
9      }
10     return summe;
11 }
```

Und was macht der Bug bei der Ausführung?

Starten wir das Programm,

erhalten wir **manchmal** das erwartete Ergebnis, **meistens nicht!**

Zeile 3 : was fällt auf?

```
1  #include "summe.h"
2
3  int berechneSumme(vector<int> &daten, int &N)
4  {
5      int summe = 0;
6      for (int k=0; k < N; k++)
7      {
8          summe += daten[k];
9      }
10     return summe;
11 }
```

Zeile 3 : was fällt auf?

wir sollten float-Werte addieren!

```
1  #include "summe.h"
2
3  int berechneSumme (vector<int> &daten, int &N)
4  {
5      int summe = 0;
6      for (int k=0; k < N; k++)
7      {
8          summe += daten[k];
9      }
10     return summe;
11 }
```

Zeile 3 : was fällt auf?

int& N:

N kann in Funktion verändert werden, hier ist
Kopie besser, const reference bringt nichts!

```
1  #include "summe.h"
2
3  int berechneSumme(vector<int> &daten, int &N)
4  {
5      int summe = 0;
6      for (int k=0; k < N; k++)
7      {
8          summe += daten[k];
9      }
10     return summe;
11 }
```

Umgeschrieben auf unser Header-file,
was kann noch verbessert werden?

```
1  #include "summe.h"
2
3  float berechneSumme( const vector<float>& summanden ) {
4      float summe = 0;
5      for( int k = 0; k < summanden.size(); k++ ) {
6          summe += summanden[k];
7      }
8      return summe;
9  }
```


Umgeschrieben auf unser Header-file,
was kann noch verbessert werden?

```
1  #include "summe.h"
2
3  float berechneSumme( const vector<float>& summanden ) {
4      float summe = 0;
5      for( int k = 0; k < summanden.size(); k++ ) {
6          summe += summanden[k];
7      }
8      return summe;
9  }
```

Umgeschrieben auf unser Header-file, was kann noch verbessert werden?

```
1  #include "summe.h"
2
3  float berechneSumme( const vector<float>& summanden ) {
4      float summe = 0;
5      for( int k = 0; k < summanden.size(); k++ ) {
6          summe += summanden[k];
7      }
8      return summe;
9  }
```

das ist eigentlich ein double!

[]-Operator von vector
wirft keine out-of-range
exception. vector::at(i) schon!

Pre-Increment ist schneller
als Post-Increment
(kein Zwischenwert nötig)

Akzeptabler Code

```
1  #include "summe.h"
2
3  float berechneSumme( const vector<float>& summanden ) {
4      float summe = 0.0f;
5      for( int k = 0; k < summanden.size(); ++k ) {
6          summe += summanden.at( k );
7      }
8      return summe;
9  }
```

```
int main()
```

Zeilen 1 – 4

```
1  #include <iostream>
2  #include <vector>
3  #include "summe.h"
4  using namespace std;
5
6
7  int main()
8  {
9      int N = 0;
10     int n;
11     string Name;
12     cout << "Wie ist dein Name?" <<endl;
13     cin >> Name;
14     vector<int> daten;
15     cout << "Hallo " << Name <<"! Wie viele Werte sollen addiert werden?" << endl;
16     cin >> N;
```

Inkludereihenfolge!

```
1  #include <iostream>
2  #include <vector>
3  #include "summe.h"
4  using namespace std;
```

Am Besten zuerst die eigenen Dateien inkludieren! Danach die Standardlibraries.

Using von den #include absetzen.

```
7  int main()
8  {
9      int N = 0;
10     int n;
11     string Name;
12     cout << "Wie ist dein Name?" <<endl;
13     cin >> Name;
14     vector<int> daten;
15     cout << "Hallo " << Name <<"! Wie viele Werte sollen addiert werden?" << endl;
16     cin >> N;
```

Zeilen 9 und 10

```
7  int main()
8  {
9      int N = 0;
10     int n;
11     string Name;
12     cout << "Wie ist dein Name?" << endl;
13     cin >> Name;
14     vector<int> daten;
15     cout << "Hallo " << Name << "! Wie viele Werte sollen addiert werden?" << endl;
16     cin >> N;
17     for (unsigned int i=0; i <N; i++)
18     {
19         cout << "Bitte geben sie den " << i+1 << ". Wert ein!" << endl;
20         cin >> n;
21         daten.push_back(n);
22         daten[i];
23         //cout << "WERT_TEST_" << i+1 << ": " << daten[i] << endl;
24     }
```

Tun Sie das nicht!

Seien Sie kreativ in der Wahl ihrer Variablen.

Niemals Bezeichner nur durch Groß- / Kleinschreibung unterscheiden!
Variablen am besten gleich initialisieren!
Variablen in C++ dort deklarieren und definieren, wo sie gebraucht werden!

Strukturieren!

```
7  int main()
8  {
9      Name eingeben
10     string Name;
11     cout << "Wie ist dein Name?" << endl;
12     cin >> Name;
13
14     Werte eingeben
15     int anzahl = 0;
16     cout << "Hallo " << Name << "! Wie viele Werte sollen addiert werden?" << endl;
17     cin >> anzahl;
18     // hier Fehlerabfrage durchführen
19
20     int input = 0;
21     vector<int> daten;
22     for (unsigned int i = 0; i < anzahl; ++i)
23     {
24         cout << "Bitte geben sie den " << i+1 << ". Wert ein!" << endl;
25         cin >> input;
26         daten.push_back(input);
27         daten[i];
28         //cout << "WERT_TEST_" << i+1 << ": " << daten[i] << endl;
29     }
```


Zeilen 18 - 27

```
7  int main()
8  {   Name eingeben
9      string Name;
10     cout << "Wie ist dein Name?" << endl;
11     cin >> Name;
12     Werte eingeben
13     int anzahl = 0;
14     cout << "Hallo " << Name << "! Wie viele Werte sollen addiert werden?" << endl;
15     cin >> anzahl;
16     // hier Fehlerabfrage durchführen
17
18     int input = 0;
19     vector<int> daten;
20     for (unsigned int i = 0; i < anzahl; ++i)
21     {
22         cout << "Bitte geben sie den " << i+1 << ". Wert ein!" << endl;
23         cin >> input;
24         daten.push_back(input);
25         daten[i];
26         //cout << "WERT_TEST_" << i+1 << ": " << daten[i] << endl;
27     }
```

Zeilen 18 - 27

```
7  int main()
8  {   Name eingeben
9      string Name;
10     cout << "Wie ist dein Name?" << endl;
11     cin >> Name;
12     Werte eingeben
13     int anzahl = 0;
14     cout << "Hallo " << Name << "! Wie viele Werte sollen addiert werden?" << endl;
15     cin >> anzahl;
16     // hier Fehlerabfrage durchführen
17
18     int input = 0;
19     vector<int> daten;
20     for (unsigned int i = 0; i < anzahl; ++i)
21     {
22         cout << "Bitte geben sie den " << i+1 << ". Wert ein!" << endl;
23         cin >> input;
24         daten.push_back(input);
25         daten[i];
26         //cout << "WERT_TEST_" << i+1 << ": " << daten[i] << endl;
27     }
```

Keine „sinnlosen“ Zeilen!

Ein C++
Abenteurer

noch
angeseilt!



```
1  #ifndef SUMME_CONTAINER_H_INCLUDED
2  #define SUMME_CONTAINER_H_INCLUDED
3
4  #include <vector>
5
6  using element_type = float;
7  constexpr element_type ELEMENT_TYPE_ZERO = 0.0f;
8
9  using container_type = std::vector<element_type>;
10 using container_type_iterator = container_type::const_iterator;
11
12 element_type sum_oldstyle( const container_type& summands );
13
14 #endif // SUMME_CONTAINER_H_INCLUDED
15
```

```
1  #include "sum_container.h"
2
3  using namespace std;
4
5  element_type sum_oldstyle( const container_type& summands ) {
6      element_type result = ELEMENT_TYPE_ZERO;
7      for( container_type::size_type i = 0; i < summands.size(); ++i ) {
8          result += summands.at( i );
9      }
10     return result;
11 }
12
13
```

```

1  #ifndef SUMME_CONTAINER_H_INCLUDED
2  #define SUMME_CONTAINER_H_INCLUDED
3
4  #include <vector>
5  // #include <list>
6  // #include <array>
7
8  using element_type = float;
9  constexpr element_type ELEMENT_TYPE_ZERO = 0.0f;
10
11 using container_type = std::vector<element_type>;
12 using container_type_iterator = container_type::const_iterator;
13
14 element_type sum_oldstyle( const container_type& summands );
15 element_type sum_c11( const container_type& summands );
16 element_type sum_iterator( const container_type& summands );
17 element_type sum_algorithm( const container_type& summands );
18 element_type sum_func_obj( const container_type& summands );
19 element_type sum_lambda( const container_type& summands );
20
21 auto sum_fancy( const container_type& summands ) -> element_type;
22
23 template <typename ELEMENT_T,
24           template <typename ELEMENT_T,
25                   typename = std::allocator<ELEMENT_T> >
26           class CONTAINER_T = std::vector>
27 ELEMENT_T sum_template( const CONTAINER_T<ELEMENT_T>& summands )
28 {
29     ELEMENT_T result = 0;
30     typename CONTAINER_T<ELEMENT_T>::const_iterator it = summands.begin();
31     while( it != summands.end() ) {
32         result += *it;
33         ++it;
34     }
35     return result;
36 }
37
38 #endif // SUMME_CONTAINER_H_INCLUDED

```

```

119 int main() {
120     const container_type summands = { 1, 2, 3, 4, 5, 6, 7, 8 };
121     element_type result_expected = 36;
122     ostream& ostr = cout;
123     //ofstream ostr( "test.txt" );
124
125     test_sum_oldstyle( ostr, summands, result_expected );
126     test_sum_c11( ostr, summands, result_expected );
127     test_sum_iterator( ostr, summands, result_expected );
128     test_sum_algorithm( ostr, summands, result_expected );
129     test_sum_func_obj( ostr, summands, result_expected );
130     test_sum_lambda( ostr, summands, result_expected );
131     test_sum_template( ostr );
132
133     ostr << "Summe aus " << summands << " = " << sum_fancy( summands ) << endl;
134     return 0;
135 }

```

```

32 bool test_sum_oldstyle( ostream& ostr, const container_type& summands, element_type result_expected ) {
33     element_type test_result = sum_oldstyle( summands );
34     bool test_ok = (result_expected == test_result);
35     if( test_ok ) {
36         ostr << "Test of sum_oldstyle succeeded!" << endl;
37     }
38     else {
39         ostr << "Test of sum_oldstyle failed, expected: " << result_expected << " but was " << test_result << endl;
40     }
41     return test_ok;
42 }

```

May C++
be with you.



