Einführung in die Programmierung mit C++

Felder der Standardbibliothek

Uwe Naumann

Informatik 12:
Software and Tools for Computational Engineering (STCE)

RWTH Aachen

Inhalt

Die Größe statischer Felder muss zur Übersetzungszeit des Programms feststehen, z.B.

- **int** i[42];
- **const** size_t n=42; ... **int** i[n];
- **#define** N 42; ... **int** i[N];
- **int** i[N]; und `g++ -DN=42 ...`

Die Größe dynamischer Felder kann zur Laufzeit des Programms definiert werden, z.B. **int** i[n]; und

- Einlesen von `n` (z.B. aus Datei)
- Berechnen von `n`
- dynamische Änderung der Größe (ggf. Umkopieren)

Die Verwendung statischer Felder ist restriktiver aber oft auch effizienter.

Inhalt

Dynamische Felder der Standardbibliothek

Allokation. Initialisierung. Elementzugriff, Speicherlayout


```
1  #include<iostream>
2  #include<vector>
3
4  int main() {
5    size_t n; std::cin >> n;
6    std::vector<int> v(n,42);
7    for (size_t i=0;i<v.size();i++)
8      std::cout << &v[i] << ": "
9            << v[i] << std::endl;
10   return 0;
11 }
```

Nach Eingabe von 3 wird z.B.

```
0x55d8670a52c0: 42
0x55d8670a52c4: 42
0x55d8670a52c8: 42
```

ausgegeben.

- Die Standardbibliothek stellt einen typgenerischen dynamischen Vektortypen zur Verfügung.
- Bei Allokation ist homogene Initialisierung mit einem Wert möglich.
- Der Zugriff auf einzelne Elemente erfolgt mittels v[i], wobei keine Überprüfung der Zulässigkeit geschieht.
- Die Vektorgröße vom Typ size_t wird mittels size() bestimmt.
- Der durch Vektoren belegte Speicherplatz wird automatisch freigegeben.
- Vektoren sollten 1D-Feldern vorgezogen werden.

Dynamische Felder der Standardbibliothek

Dynamisches Wachstum. Sicherer Elementzugriff


```
1  #include<iostream>
2  #include<vector>
3
4  int main() {
5    using namespace std;
6    size_t n; cin >> n;
7    vector<int> v;
8    cout << &v << endl;
9    for (size_t i=0;i<n;i++) {
10     v.push_back(i);
11     cout << &v.at(i) << ": "
12          << v.at(i) << endl;
13   }
14   return 0;
15 }
```

- Vektoren der Größe 0 können alloziert werden.
- Sie können dynamisch befüllt werden.
- Sicherer Elementzugriff geschieht mittels at() (Extrakosten).
- Unzulässige Elementzugriffe generieren eine Ausnahme.

→ Live: unzulässiger Elementzugriff

Sinnvolle Behandlung der generierten Ausnahme ermöglicht Weiterführung des Programms trotz Fehlers, z.B. generiert

```
1  #include<iostream>
2  #include<vector>
3
4  int main() {
5    std::vector<int> v;
6    try {
7      ++v.at(42);
8    } catch (std::exception& e) { // caught "by reference" (see later)
9      std::cerr << e.what() << std::endl;
10   }
11   std::cout << "done." << std::endl;
12   return 0;
13 }
```

die Ausgabe

```
vector::_M_range_check: __n (which is 42) >= this->size() (which is 0)
done.
```

Dynamische Felder der Standardbibliothek

Iterator


```
1  #include<iostream>
2  #include<vector>
3
4  int main() {
5    size_t n; std::cin >> n;
6    std::vector<int> v(n);
7    for (std::vector<int>::iterator i=v.begin();
8          i!=v.end();i++) {
9      *i=n--;
10     std::cout << *i << std::endl;
11   }
12   return 0;
13 }
```

erzeugt bei Eingabe von 2 die Ausgabe

2
1

- Für den Zugriff auf Elemente von Containern (z.B. std::vector) stellt die Standardbibliothek Iteratoren zur Verfügung.
- Schreib-/Lesezugriff
- spezielle Iteratorwerte markieren Anfang (begin()) und Ende (end(); Position "nach dem letzten Element") des Vektors
- "Dereferenzierung" mittels *

Die Deklaration von Iteratoren kann man oft dem Compiler überlassen.

```
1  #include<iostream>
2  #include<vector>
3
4  int main() {
5    size_t n; std::cin >> n;
6    std::vector<int> v(n);
7    for (auto i=v.begin();i!=v.end();i++) {
8      *i=n--; std::cout << *i << std::endl;
9    }
10   return 0;
11 }
```

Spezielle Iteratoren für exklusiven Lesezugriff werden bereitgestellt, z.B. wird bei Eingabe von 2 durch

```
1  #include<iostream>
2  #include<vector>
3
4  int main() {
5    size_t n; std::cin >> n; std::vector<int> v(n);
6    for (std::vector<int>::iterator i=v.begin(); // read/write (also: auto)
7          i!=v.end(); i++)
8      *i=--n;
9    for (std::vector<int>::const_iterator i=v.cbegin(); // read-only (also: auto)
10         i!=v.cend();i++)
11     std::cout << *i << std::endl;
12   return 0;
13 }
```

```
1
0
```

ausgegeben.

Spezielle Iteratoren für den Zugriff auf die Elemente in umgekehrter Reihenfolge werden bereitgestellt, z.B. wird bei Eingabe von 2 durch

```
1  #include<iostream>
2  #include<vector>
3
4  int main() {
5    size_t n; std::cin >> n;
6    std::vector<int> v(n);
7    for (std::vector<int>::reverse_iterator i=v.rbegin();i!=v.rend();i++)
8      *i=n--;
9    for (auto i=v.cbegin();i!=v.cend();i++)
10     std::cout << *i << std::endl;
11   return 0;
12 }
```

1
2

ausgegeben.

Spezielle Iteratoren für den exklusiven Lesezugriff auf die Elemente in umgekehrter Reihenfolge werden bereitgestellt, z.B. wird bei Eingabe von 2 durch

```
1  #include<iostream>
2  #include<vector>
3
4  int main() {
5    size_t n; std::cin >> n; std::vector<int> v(n);
6    for (auto i=v.rbegin();i!=v.rend();*i++=n--);
7    for (std::vector<int>::const_reverse_iterator i=v.crbegin();i!=v.crend();i++)
8      std::cout << *i << std::endl;
9    return 0;
10 }
```

```
2
1
```

ausgegeben. Die Verwendung von **auto** spart Schreibarbeit. Beachte: Interessante for-Schleifensyntax!

AUFWACHEN!

Wird folgender Code erfolgreich übersetzt?

```
1  #include<iostream>
2  #include<vector>
3
4  int main() {
5    using T=float;
6    const size_t n=3; std::vector<T> v(n);
7    std::vector<T>::iterator vi;
8    for (vi=v.begin();vi!=v.end();*(vi++)=42);
9    std::vector<T>::const_iterator vci;
10   for (vci=v.begin();vci!=v.end();vci++) std::cout << *vci << std::endl;
11   for (vi=v.cbegin();vi!=v.cend();vi++) std::cout << *vi << std::endl;
12   return 0;
13 }
```

Wach?

```
1  #include<iostream>
2  #include<vector>
3
4  int main() {
5    using T=float;
6    const size_t n=3; std::vector<T> v(n);
7    std::vector<T>::iterator vi;
8    for (vi=v.begin();vi!=v.end();*(vi++)=42);
9    std::vector<T>::const_iterator vci;
10   for (vci=v.begin();vci!=v.end();vci++) std::cout << *vci << std::endl;
11   for (vi=v.cbegin();vi!=v.cend();vi++) std::cout << *vi << std::endl;
12   return 0;
13 }
```

```
error: no match for ‘operator=’ ...
   11 |   for (vi=v.cbegin();vi!=v.cend();vi++) std::cout << *vi << std::endl;
      |                      ^
...
```

WACH BLEIBEN!

und der?

```
1  include<iostream>
2  #include<vector>
3
4  int main() {
5     using T=float;
6     const size_t n=3; std::vector<T> v(n);
7     std::vector<T>::iterator vi;
8     for (vi=v.begin();vi!=v.end();*(vi++)=42);
9     std::vector<T>::const_iterator vci;
10    for (vci=v.begin();vci!=v.end();vci++) std::cout << *vci << std::endl;
11    for (auto vi=v.cbegin();vi!=v.cend();vi++) std::cout << *vi << std::endl;
12    return 0;
13 }
```

Noch Wach?

```
1  include<iostream>
2  #include<vector>
3
4  int main() {
5    using T=float;
6    const size_t n=3; std::vector<T> v(n);
7    std::vector<T>::iterator vi;
8    for (vi=v.begin();vi!=v.end();*(vi++)=42);
9    std::vector<T>::const_iterator vci;
10   for (vci=v.begin();vci!=v.end();vci++) std::cout << *vci << std::endl;
11   for (auto vi=v.cbegin();vi!=v.cend();vi++) std::cout << *vi << std::endl;
12   return 0;
13 }
```

Ja!

Dynamische Felder der Standardbibliothek

Matrizen

Matrizen (und Tensoren) können rekursiv als Vektoren von Vektoren (von Vektoren ...) implementiert werden.

```
1  #include<iostream>
2  #include<vector>
3
4  int main() {
5    size_t m,n,mn=0; std::cin >> m >> n;
6    std::vector<std::vector<int>> v(m,std::vector<int>(n));
7    for (auto i=v.begin();i!=v.end();i++)
8      for (auto j=i->begin();j!=i->end();*j++=mn++);
9    for (size_t i=0;i<m;i++) {
10     for (size_t j=0;j<n;j++)
11       std::cout << v[i].at(j) << " ";
12     std::cout << std::endl;
13   }
14   return 0;
15 }
```

(Sicherer) Elementzugriff, Verwendung von Iteratoren sowie **auto** folgt aus dem bereits Gelernten. → Optional: Speicherlayout mit gdb

AUFWACHEN!

Was wird bei Eingabe von 2 2 2 ausgegeben?

```
1   #include<iostream>
2  #include<vector>
3
4  int main() {
5    size_t m,n,p; std::cin >> m >> n >> p;
6    using VT=std::vector<int>;
7    using MT=std::vector<VT>;
8    using TT=std::vector<MT>;
9    TT v(m,MT(n,VT(p)));
10   for (auto i=v.begin();i!=v.end();i++,m--)
11     for (auto j=i->rbegin();j!=i->rend();j++,n--)
12       for (auto k=j->begin();k!=j->end();*k++=p--);
13   for (auto i=v.cbegin();i!=v.cend();i++)
14     for (auto j=i->cbegin();j!=i->cend();j++)
15       for (auto k=j->cbegin();k!=j->cend();k++)
16         std::cout << *k << std::endl;
17   return 0;
18 }
```

Wach?

Was wird bei Eingabe von 2 2 2 ausgegeben?

```
1   #include<iostream>
2  #include<vector>
3
4  int main() {
5    size_t m,n,p; std::cin >> m >> n >> p;
6    using VT=std::vector<int>;
7    using MT=std::vector<VT>;
8    using TT=std::vector<MT>;
9    TT v(m,MT(n,VT(p)));
10   for (auto i=v.begin();i!=v.end();i++,m--)
11     for (auto j=i->rbegin();j!=i->rend();j++,n--)
12       for (auto k=j->begin();k!=j->end();*k++=p--);
13   for (auto i=v.cbegin();i!=v.cend();i++)
14     for (auto j=i->cbegin();j!=i->cend();j++)
15       for (auto k=j->cbegin();k!=j->cend();k++)
16         std::cout << *k << std::endl;
17   return 0;
18 }
```

0 -1 2 1 -4 -5 -2 -3

- Reservieren von Speicher für dynamische Wachstum mit Datenlokalität (v.reserve)
- Abfrage von
  - Größe des reservierten Speichers (v.capacity)
  - maximal möglicher Größe (v.max_size)
- Änderung der Größe (v.resize)
- Einfügen und Löschen von Elementen (ineffizient; besser → std::list)
- Löschen des Inhalts (v.clear)
- *Stack*-Modus (v.push_back / v.pop_back)

Siehe Literatur und Internet für komplette Übersicht und weiterführende Details.

Inhalt

```
1  #include<iostream>
2  #include<array>
3
4  int main() {
5    const size_t n=3;
6    std::array<int,n> v={1,2,3};
7    for (size_t i=0;i<v.size();i++)
8      std::cout << &v[i] << ": "
9                << v[i] << std::endl;
10   return 0;
11 }
```

- Die Standardbibliothek stellt einen typgenerischen statischen Vektortypen zur Verfügung, der nutzerdefinierten statischen Feldern vorgezogen werden sollte.
- Die für nutzerdefinierte statische Felder diskutierten Konzepte sind analog anwendbar (z.B. Spezifikation der Größe)
- Die Mehrzahl der für std::vector diskutierten Konzepte kann ebenfalls verwendet werden (z.B. Iteratoren).

Inhalt

AUFWACHEN!

Verwenden Sie möglichst viele der eingeführten Konzepte in einer Implementierung eines Matrix-Vektor-Produkts.

Das sollten sie eigentlich nie tun, da z.B. folgendes Programm sicher nicht dem Prinzip "*keep it simple*" entspricht ...

Wach?

```
1  #include<iostream>
2  #include<vector>
3  #include<array>
4
5  int main() {
6    size_t m,mn=0; const size_t n=3; std::cin >> m;
7    std::vector<std::vector<int>> A(m,std::vector<int>(n));
8    std::vector<int> Av; std::array<int,n> v;
9    for (auto i=A.begin();i!=A.end();i++) {
10     for (auto j=i->begin();j!=i->end();*j++=mn++)
11       std::cout << mn << " "; std::cout << std::endl;
12   }
13   for (auto i=v.begin();i!=v.end();*i++=mn--)
14     std::cout << mn << " "; std::cout << std::endl;
15   for (size_t i=0;i<m;i++) {
16     int Avi=0; for (size_t j=0;j<n;j++) Avi+=A[i][j]*v[j];
17     Av.push_back(Avi);
18   }
19   for (auto i=Av.cbegin();i!=Av.cend();i++)
20     std::cout << *i << " "; std::cout << std::endl;
21   return 0;
22 }
```

Zusammenfassung

Dynamische Felder

- Sicherer Elementzugriff
- Iteratoren
- Alternativer Elementzugriff
- Matrizen und Tensoren

Statische Felder

Matrix-Vektor Produkt