Einführung in die Programmierung mit C++

Referenzen und Zeiger

Uwe Naumann

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

RWTH Aachen

Inhalt

Referenzen

- Schreib-/Lesezugriff
- Ausschließlicher Lesezugriff

Zeiger

- Zeigerarithmetik
- [Konstante] Zeiger [auf Konstanten]
- Zeigerketten

Inhalt

Referenzen
- Schreib-/Lesezugriff
- Ausschließlicher Lesezugriff

Zeiger
- Zeigerarithmetik
- [Konstante] Zeiger [auf Konstanten]
- Zeigerketten

Referenzen

Typisierte Aliasnamen für Allozierten Speicher


```
1  #include<iostream>
2
3  int main() {
4    using namespace std;
5    int i=42, &ir=i;
6    cout << i << " " << &i << endl
7         << ir << " " << &ir << endl;
8    ir=24;
9    cout << i << " " << ir << endl;
10   return 0;
11 }
```

generiert z.B. Ausgabe

```
42 0x7fff9179003c
42 0x7fff9179003c
24 24
```

- Die Verwendung von & zur Deklaration von Referenzen sollte zu keiner Verwechslung mit der Semantik des Adressoperators führen.
- Referenzen werden vollständig durch den Compiler aufgelöst. Daher müssen sie bei ihrer Deklaration initialisiert werden.
- Referenzen referenzieren während ihrer Gültigkeit immer denselben typisierten Speicherbereich.

Konstante Referenzen (für nichtkonstante Variablen) erlauben keine Schreibzugriffe; z.B. resultiert die Übersetzung von

```
1  #include<iostream>
2
3  int main() {
4    using namespace std;
5    int i=42; const int &ir=i;
6    cout << i << " " << &i << endl << ir << " " << &ir << endl;
7    ir=24;
8    cout << i << " " << ir << endl;
9    return 0;
10 }
```

in einem Fehler:

```
In function ‘int main()’:
error: assignment of read-only reference ‘ir’
    7 |   ir=24;
      |   ~~~^~
```

Inhalt

Referenzen
- Schreib-/Lesezugriff
- Ausschließlicher Lesezugriff

Zeiger
- Zeigerarithmetik
- [Konstante] Zeiger [auf Konstanten]
- Zeigerketten

Zeiger

Addressvariablen


```
1  #include<iostream>
2
3  int main() {
4    using namespace std;
5    int i=42;
6    int *ip=&i;
7    cout << i << endl
8         << &i << endl
9         << ip << endl
10        << &ip << endl
11        << *ip << endl;
12   return 0;
13 }
```

erzeugt z.B. Ausgabe

```
42
0x7fff7631785c
0x7fff7631785c
0x7fff76317860
42
```

- Typisierter Zeiger ip referenziert (speichert die Basisadresse) Variable i
- Zugriff auf Wert der referenzierten Variable mittels Dereferenzierungsoperator *
- Größe des referenzierten Speichers (in Bytes) is dem Compiler über den Typ des Zeigers bekannt (siehe auch **sizeof**)
- Auf einer 2^k-Bit Computerarchitektur beträgt die Größe von Zeigervariablen 2^{k-3} Byte (8 Byte auf 64-Bit Systemen).
- Nicht initialisierte Zeiger sollten gleich nullptr gesetzt werden. Deren (unerlaubte) Dereferenzierung führt zu einem Speicherzugriffsfehler.

Mit Zeigern kann man rechnen, sie inkrementieren bzw. dekrementieren. Dabei ist die kleinste Einheit gleich der Anzahl **sizeof**(T) der durch den jeweils referenzierten Datentyp (T) belegten Bytes.

Beispiel:

```
1  #include<iostream>
2
3  using T=double;
4
5  int main() {
6    T x=1;
7    T y=2;
8    T* p=&x;
9    std::cout << *++p << std::endl;
10   return 0;
11 }
```

Was wird ausgegeben?

→ gdb

Der referenzierter Inhalt ist nicht veränderbar. Übersetzungsversuch für

```
1  int main() {
2    int i=42;
3
4    const int * cip1=&i;
5    *cip1=43; cip1++; // Fehler; OK
6
7    int const * cip2=&i;
8    *cip2=43; cip2++; // Fehler; OK
9
10   return 0;
11 }
```

resultiert in folgenden Fehlermeldungen:

```
...:5:8: error: assignment of read-only location ‘* cip1’
   *cip1=43; cip1++;
        ^
...:8:8: error: assignment of read-only location ‘* cip2’
   *cip2=43; cip2++;
        ^
```

Die im Zeiger gespeicherte Addresse ist nicht veränderbar. Übersetzungsversuch für

```
1  int main() {
2    int i=42;
3
4    int *const icp=&i;
5    *icp=43; icp++; // OK; Fehler
6
7    const int *const cicp1=&i;
8    *cicp1=43; cicp1++; // Fehler; Fehler
9
10   int const *const cicp2=&i;
11   *cicp2=43; cicp2++; // Fehler; Fehler
12
13   const int const *const cicp3=&i; // Fehler
14
15   return 0;
16 }
17 ...
```

... resultiert in folgenden Fehlermeldungen:

```
...:5:12: error: increment of read-only variable ‘icp’
    5 |   *icp=43; icp++;
      |            ^~~
...:8:9: error: assignment of read-only location ‘*(const int*)cicp1’
    8 |   *cicp1=43; cicp1++;
      |   ~~~~~~~~~
...:8:14: error: increment of read-only variable ‘cicp1’
    8 |   *cicp1=43; cicp1++;
      |              ^~~~~
...:11:9: error: assignment of read-only location ‘*(const int*)cicp2’
   11 |   *cicp2=43; cicp2++;
      |   ~~~~~~~~~
...:11:14: error: increment of read-only variable ‘cicp2’
   11 |   *cicp2=43; cicp2++;
      |              ^~~~~
...:13:13: error: duplicate ‘const’
   13 |   const int const *const cicp3=&i;
      |             ^~~~~
```

AUFWACHEN!

Was wird auf einem 64-Bit Computer ausgegeben?

```
1  #include<iostream>
2
3  int main() {
4      using namespace std;
5      int i;
6      cout << sizeof(&i)-sizeof(i) << sizeof(short) << endl;
7      return 0;
8  }
```

Wach?

```
1  #include<iostream>
2
3  int main() {
4      using namespace std;
5      int i;
6      cout << sizeof(&i)-sizeof(i) << sizeof(short) << endl;
7      return 0;
8  }
```

42 (*What else ...?*)

Aliasnamen für Zeiger können deklariert werden ...

```
1  #include<iostream>
2
3  int main() {
4    int i=42, *ip=&i, *&ipr=ip;
5    std::cout << i << " " << *ip << " " << *ipr << std::endl;
6    return 0;
7  }
```

(Ausgabe: 42 42 42) ... Zeiger auf Referenzen jedoch nicht. Übersetzung von

```
1  int main() {
2    int i=42, *ip=&i, &*ipr;
3  }
```

resultiert in Fehlermeldung:

```
...:2:23: error: cannot declare pointer to ‘int&’
    2 |   int i=42, *ip=&i, &*ipr;
      |                      ^~~
```

AUFWACHEN!

Was wird ausgegeben?

```
1  #include<iostream>
2
3  int main() {
4     using namespace std;
5     float f=1.2, g=2.1, *fp=&f, &fr=*fp; fr=g;
6     cout << f << endl;
7     return 0;
8  }
```

Wach?

```
1  #include<iostream>
2
3  int main() {
4     using namespace std;
5     float f=1.2, g=2.1, *fp=&f, &fr=*fp; fr=g;
6     cout << f << endl;
7     return 0;
8  }
```

2.1

WACH BLEIBEN!

Kann folgendes Programm erfolgreich übersetzt werden?

```
1  #include<iostream>
2
3  int main() {
4     using namespace std;
5     double d=0.123456789;
6     const double& dr=d;
7     d=2.3; dr=3.2;
8     cout << "d=" << d << endl
9          << "dr=" << dr << endl;
10    return 0;
11 }
```

Noch Wach?

Kann folgendes Programm erfolgreich übersetzt werden?

```
1  #include<iostream>
2
3  int main() {
4    using namespace std;
5    double d=0.123456789;
6    const double& dr=d;
7    d=2.3; dr=3.2;
8    cout << "d=" << d << endl
9         << "dr=" << dr << endl;
10   return 0;
11 }
```

Nein.

```
...:7:12: error: assignment of read-only reference ‘dr’
    7 |   d=2.3; dr=3.2;
      |          ~~~~~~
```

WEITER WACH BLEIBEN!

... und das?

```
1  #include<iostream>
2  using namespace std;
3
4  int main() {
5    double d=0.123456789;
6    const double& dr=d;
7    d=2.3; // dr=3.2;
8    cout << "d=" << d << endl
9         << "dr=" << dr << endl;
10   return 0;
11 }
```

Immernoch Wach?

```
1  #include<iostream>
2  using namespace std;
3
4  int main() {
5    double d=0.123456789;
6    const double& dr=d;
7    d=2.3; // dr=3.2;
8    cout << "d=" << d << endl
9         << "dr=" << dr << endl;
10   return 0;
11 }
```

Ja.

```
1  #include<iostream>
2
3  int main() {
4    using namespace std;
5    int i=1; int* ip=&i; int** ipp=&ip;
6    int*** ippp=&ipp;
7    cout << &i << " " << i << endl;
8    cout << ip << " "
9      << &ip << " " << *ip << endl;
10   cout << ipp << " "
11     << &ipp << " " << **ipp << endl;
12   cout << ippp << " "
13     << &ippp << " " << ***ippp << endl;
14   return 0;
15 }
```

generiert z.B. Ausgabe

```
0x7ffc0534360c 1
0x7ffc0534360c 0x7ffc05343610 1
0x7ffc05343610 0x7ffc05343618 1
0x7ffc05343618 0x7ffc05343620 1
```

- Es können Zeiger auf Zeiger auf ... auf Zeiger auf Typen deklariert werden.
- Entsprechende Referenzen können deklariert werden.
- Kombination mit **const** resultiert in einer Vielzahl von (mehr oder weniger sinnvollen) Varianten.

```
1  #include<iostream>
2
3  int main() {
4    using namespace std;
5    int i=1; int* ip=&i;
6    int** ipp=&ip; int*** ippp=&ipp;
7    int**& ippr=*ippp; int***& ipppr=ippp;
8    cout << &i << " " << i << endl;
9    cout << ip << " " << &ip << " " << *ip << endl;
10   cout << ippr << " " << &ippr << " " << **ippr << endl;
11   cout << ipppr << " " << &ipppr << " " << ***ipppr << endl;
12   return 0;
13 }
```

generiert die strukturell zur vorherigen Folie äquivalente Ausgabe

```
0x7fff462eca9c 1
0x7fff462eca9c 0x7fff462ecaa0 1
0x7fff462ecaa0 0x7fff462ecaa8 1
0x7fff462ecaa8 0x7fff462ecab0 1
```

Zusammenfassung

Referenzen

- Schreib-/Lesezugriff
- Ausschließlicher Lesezugriff

Zeiger

- Zeigerarithmetik
- [Konstante] Zeiger [auf Konstanten]
- Zeigerketten