Referenzen

- In c kennen wir gewöhnliche Variablen und Pointer.
- C++ führt zusätzlich Referenzen ein.
- Eine Referenz ist so etwas, wie ein alias auf eine Variable, ein zweiter Zugang zu der Variablen.
- Eine Referenz kann immer nur in Verbindung mit dem referenzierten Objekt (hier Variable oder Konstante) existieren.
- Eine Referenz hat im Gegensatz zum Pointer keine eigene Adresse.

Referenzen (Bsp. 1)

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
   int  myInt=123;
   int& other=myInt;

   cout << "myInt: "<< myInt << endl;
   cout << "other: "<< other << endl;

   other ++;
   cout << "myInt: "<< myInt << endl;
   cout << "other: "<< other << endl;

   myInt++;
   cout << "myInt: "<< myInt << endl;
   cout << "other: "<< other << endl;
  return 0;
}
```

Das ist eine Referenz

```
./a.out
myInt: 123
other: 123
myInt: 124
other: 124
myInt: 125
other: 125
```

Referenzen

- Eine Referenzvereinbarung wird durch ein & nach dem Typnamen einer Variablendefinition angezeigt: **`int& other`**
- Eine Referenz muss grundsätzlich Initialisiert sein.
- Eine Referenz ohne referenziertes Objekt kann es nicht geben. Deshalb muss es komplett heißen: `int& other=myInt;`
- Die Referenz other referenziert hier die Variable myInt.

Referenzen

- Wird das Programm ausgeführt zeigt sich, dass jede Wertänderung an myInt auch unter dem Namen other zu sehen ist und jede Wertänderung an other auch in der Variable myInt beobachtet werden kann.

```
./a.out
myInt: 123
other: 123
myInt: 124
other: 124
myInt: 125
other: 125
```

Referenzen

- Eine Referenz muss immer initialisiert sein
- Eine Referenz hat keine eigene Adresse, alle Operationen, werden an dem referenzierten Objekt ausgeführt.
- Bezogen auf das vorangegangene Beispiel würde eine sizeof-Operation `sizeof other` die Größe von `myInt` liefern und `&other` liefert die Adresse von `myInt.`
- An der Verwendung von & als Adressoperator hat sich nichts geändert.

Referenzen Beispiel Bsp. 2

Hier ist & das Adressymbol

```
./a.out
sizeof  myInt: 4
sizeof  other: 4
sizeof &myInt: 8
Adresse myInt: 0x7ffc8f2cb764
Adresse other: 0x7ffc8f2cb764
```

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
   int  myInt=123;
   int& other=myInt;

   cout << "sizeof  myInt: " << (sizeof myInt) << endl;
   cout << "sizeof  other: " << (sizeof other) << endl;
   cout << "sizeof &myInt: " << (sizeof &myInt)<< endl;

   cout << "Adresse myInt: " << (& myInt) << endl;
   cout << "Adresse other: " << (& other) << endl;
   return 0;
}
```

Referenzen

- Eine Referenz kann das Attribut const tragen.
  `const int& other=myInt;`
- Trägt eine Referenz das Attribut `const`, so kann das referenzierte Objekt (hier Variable) über die Referenz nicht verändert werden.
- Eine Variable kann durch eine const-Referenz referenziert werden, eine konstante Werte können nur über eine const-Referenz referenziert werden.

Referenzen

- Die Änderung der Referenzvereinbarung in Bsp.1 zu

```
const int& other=myInt;
```

ohne weitere Programmänderung führt zu einem Compilerfehler:

- `ref3.cpp:13:9: error: increment of read-only reference ‘other’`

```
other ++;
```

- Die Änderung des Wertes von myInt über other ist nicht mehr möglich.

Referenzen als Parameter

- Referenzen können auch als Funktionsparameter Verwendung finden.
- Die Bindung der Referenz an das referenzierte Objekt findet hier beim Funktionsaufruf statt.
- Werden Variablen als Aufrufparameter angegeben und die referenzierten Werte in der Funktion verändert, so ändern sich die originalen Werte beim Aufrufer. (Call by Reference)

Referenzen als Parameter

```
#include <iostream>
#include <iomanip>
using namespace std;

void swap(int& i, int& j) // tauscht i und j
{
  i^=j; j^=i; i^=j;
}

int main()
{
  int a=1, b=0;
  cout << "main 1: a="<< a << " b="<<b<<endl;
  swap(a,b);
  cout << "main 2: a="<< a << " b="<<b<<endl;
  return 0;
}
```

```
./a.out
main 1: a=1 b=0
main 2: a=0 b=1
```

Referenzen als Parameter

- An der Ausgabe sieht man, dass die Werte der Variablen a und b vertauscht worden sind.
- Im nachfolgenden Beispiel sehen wir eine Funktion swp1 mit gewöhnlichen Parametern, wie wir sie von c kennen (call by value) und eine Funktion swp2 mit Referenzparametern (call by reference).
- Im rosa Kasten der Ausgabe ist zu erkennen das im ersten Fall die originalen Werte nicht getauscht wurden im zweiten wurden sie vertauscht.
- **Achtung: Dem Aufruf der Funktion ist nicht anzusehen, ob es sich um call by value oder call by reference handelt.**

```
#include <iostream>
#include <iomanip>
using namespace std;

void swp1(int i, int j)
{
  i^=j; j^=i; i^=j;
}

void swp2(int& i, int& j)
{
  i^=j; j^=i; i^=j;
}

int main()
{
  int a=1, b=0;
  swp1(a,b);
  cout << "main 1: a="<< a << " b="<<b<<endl;
  a=1; b=0;
  swp2(a,b);
  cout << "main 2: a="<< a << " b="<<b<<endl;
  return 0;
}
```

```
./a.out
main 1: a=1 b=0
main 2: a=0 b=1
```

Const Referenzen als Parameter

- Sollen Werte berechneter Ausdrücke an Referenzparameter übergeben werden, so müssen diese als const gekennzeichnet sein.
- Damit ist die Änderung der referenzierten Werte dann nicht mehr möglich.
- Const ist einweiteres Kriterium zur Unterscheidung überladener Funktionen. Das folgende Beispiel enthält zwei solcher überladenener Funktionen `mul`, die sich nur durch const unterscheiden.

const Referenzen als Parameter

```
. . .
int mul(int& f1, int& f2)
{
  cout << "mul without const" << endl;
  return f1*f2;
}

int mul(const int& f1, const int& f2)
{
  cout << "mul with const" << endl;
  return f1*f2;
}

int main()
{
  int a=3, b=5;
  cout<<mul(a,b)  <<endl;
  cout<<mul(a+1,b)<<endl;
  return 0;
}
```

```
./a.out
mul without const
15
mul with const
20
```

Es erfolgt der Aufruf der ersten Funktion

Es erfolgt der Aufruf der zweiten Funktion

Referenzen als Returnwert

- Referenzen sind auch als Returnwert sehr interessant, dieser als Wert (Value) aber auch als Lvalue, also als Ziel einer Zuweisung eingesetzt werden kann.
- Eine Referenz als Returnwert darf niemals eine lokale Variable referenzieren, weil diese nach Verlassen der Funktion nicht mehr existiert.

Referenzen als Returnwert

```
#include <iostream>
#include <iomanip>
using namespace std;

int myNum=0;     // eine Variable

int & num()
{
  return myNum; // Rueckgabe einer Refernz auf myNum
}

int main()
{
  int i;
  i = num();     // Bewertung als Wert der Variablen
  cout << "i     : " << i << endl;
  num()=5;       // Bewertung als Lvalue !!!
  cout << "myNum: "<< myNum << endl;
  return 0;
}
```

```
 ./a.out
i     : 0
myNum: 5
```

Referenzen als Returnwert

- Die Funktion num gibt eine Referenz aud fie Variable myNum zurück.
- Diese Refrenz kann nun als Value in Ausdrücken oder als LValue links vom Zuweisungssymbol Verwendung finden.
- Die beiden fett markierten Zeilen demonstrieren dies.

Referenzen als Returnwert

```
. . . includegedöns . . .
const int n=10;

int& idx(int * vI, int i, int n)
{
  static int dummy;
  if (i<n)  return vI[i];
  cerr << "Indexerror at index"<< i << endl;
  return dummy;
}

int main()
{
  int vI[n]={0};
  int vJ[n]={0};
  for (int i=0; i<n; i++) idx(vI,i,n)=idx(vJ,i,n)=i;
  for (int i=0; i<n; i++)
    cout<<i<<' '<<idx(vI,i,n)<<' '<<idx(vJ,i,n)<< endl;
  idx(vI,99,n)=1234567890;
  return 0;
}
```

```
./a.out
0 0 0
1 1 1
2 2 2
3 3 3
4 4 4
5 5 5
6 6 6
7 7 7
8 8 8
9 9 9
Indexerror at index99
```

Referenzen als Returnwert

- Im Beispiel gibt die Funktion idx eine Referenz auf das i-te Element des übergebenen Arrays aus. Diese Referenz wird im Beispiel als Value und als Lvalue verwendet.
- Die Funktion führt eine Indexüberprüfung durch. ImFalle eines Indexerrors wird eine Fehlernachricht auf die Standardfehlerausgabe ausgegeben und die Referenz auf die Dummy-Variable zurückgegeben.
- Die Dummy-Variable ist static definiert, so dass sie auf alle existiert. Bei Verwendung der Referenz als LValue wird der zugewiesene Wert dann auf Dummy geschrieben. (testen Sie das mit dem Debugger – kdbg o.ä.)