§0 Konstanten und Referenzen

Leitideen: *C++ kennt neben Wertparametern auch Referenz- und konstante Referenzparameter, die statt mit einer Kopie des Arguments direkt auf diesem arbeiten.*

Die Implementierung von Referenzparametern kann mit Zeigern erfolgen, die als Wertparameter übergeben werden. Die Analyse dieses Mechanismus führt zum Referenzdatentyp. Dasselbe gilt auch für konstante Referenzen.

Bei der Vereinbarungssyntax von Referenzen wird das Lesen von innen nach außen beibehalten, die Ähnlichkeit zur Ausdruckssyntax jedoch aufgegeben.

§0 Konst. und Ref. – Themenübersicht

- Parameter von Funktionen
- Wertparameterübergabe (call by value)
- Referenzparameterübergabe (call by reference)
- Konstanten I,II
- Referenzen auf Variablen I,II
- Referenzen auf Konstanten
- Funktionswerte als Referenzen

Parameter von Funktionen

- Wertparameter (`T x`) [`T` Datentyp, `x` Parametername]
  - Beim Funktionsaufruf Kopie der Argumentwerte auf gleichnamige lokale Variablen
  - Vereinbarung dieser Variablennamen in Parameterliste, *nicht* im Funktionsblock
  - Immer Wertparameterübergabe, ausgenommen C-Vektoren und Funktionen, Referenzparameter und konstante Referenzparameter
- Referenzparameter (`T& x`)
  - Beim Funktionsaufruf Benutzung des Speicherplatzes der eingesetzten Variable (keine Kopie des Argumentwerts!)
  - Änderung der eingesetzten Variable zulässig (Hauptzweck!)
  - Konstanten und Ausdrücke (außer in Sonderfällen) als Argumente *unzulässig*
- Konstante Referenzparameter (`const T& x`)
  - Beim Funktionsaufruf Benutzung des Speicherplatzes der eingesetzten Variable, keine Kopie (Hauptzweck!)
  - Änderung der eingesetzten Variable *unzulässig*
  - Konstanten und Ausdrücke zulässig (Compiler legt Hilfsvariable an)

Wertparameterübergabe (call by value)

- Eingesetzte Argumente werden bei Wertparameterübergabe kopiert.
- Eine Änderung dieser Werte in der Funktion bewirkt *keine* Änderung evtl. eingesetzter Variablen in der *aufrufenden* Funktion.

```
void vertausche(int iv, int jv)
// iv=im; jv=jm; (Zuweisung)
{ int hv;
  hv=iv; iv=jv; jv=hv;
  return;
}

int main()
{ int im=2; jm=4;
  vertausche(im,jm); // im,jm unveraendert
  return 0;
}
```

Referenzparameterübergabe (call by reference)

- Direkter Lese- und Schreibzugriff auf eingesetzte Variable *statt* Kopie des Variablenwerts

```
void vertausche(int& iv, int& jv)
// iv ≡ im; jv ≡ jm; (gleiche Speicherplätze)
{ int hv;
  hv=iv;   // d.h. hv=im
  iv=jv;   // d.h. im=jm
  jv=hv;   // d.h. jm=hv
  return;
}

int main(void)
{ int im=2; jm=4;
  vertausche(im,jm); // im=4, jm=2
  return 0;
}
```

Anmerkung: Funktion `swap` in `<algorithm>` vorhanden!

Konstanten

- Deklarationsangabe `const`: Konstante (schreibgeschützt)
  Bsp.: `const int n  // Integerkonstante`
- Bedeutung bei C-Vektoren: Komponenten konstant
  Bsp.: `const char s[5] // Datentyp von "Text"`
- Bedeutung bei Zeigern: positionsabhängig
  Gespeicherte Adresse oder Zielobjekt schreibgeschützt oder beides
  Bsp.:
  ```
  const char *p  // Zeiger auf eine Konst.
  char * const q // Konstanter Zeiger
  ```
- Deklarationen (im Englischen) von innen nach außen lesbar
  Bsp.: p – pointer to char const
  q – const pointer to char

Konstanten - Fortsetzung

- Schreibschutz bei Konstanten als Unterstützung gedacht – lässt sich umgehen
- Initialisierung sogar syntaktisch notwendig – wird vom Compiler überprüft
- Bei Initialisierung: `const`-Attribute auf linker und rechter Seite irrelevant

  Ausnahme: Initialisierung von Zeiger auf Nichtkonstante durch Zeiger auf Konstante *verboten*
- Initialisierung $\neq$ Zuweisung
  Zuweisung an Konstante natürlich *unzulässig*
- Parameterübergabe und Rückgabe in Funktionen: Initialisierung, *keine* Zuweisung
- Eingebaute Datentypen: Initialisierung $\hat{=}$ Zuweisung
- In Klassen: Zuweisung durch überlad. Gleichheitsop., Initialisierung durch Konstruktoren

Referenzen auf Variablen

Begriff

Referenz auf Variable: weiterer Name für eine Variable
(Diese muss bei der Vereinbarung mit angegeben werden)

- *Bsp.:*
```
int  i=5;
int& j=i; // j anderer Name fuer i
          // j Referenz auf i
          // Wert von j: 5
i++;      // Wert von i,j: 6
j++;      // Wert von i,j: 7
```
- Zulässige Schreibweisen: `int& i`, `int &i`, `int & i`
  Üblich: Erste Schreibweise
- Lesen von innen nach außen: `&` steht für Referenz
  Bsp.:
```
char *p    ; // p Zeiger auf char
char*& cp=p; // Ref. auf Zeiger auf char
             // cp anderer Name fuer p
```

Referenzen auf Variablen - Fortsetzung

- Bisher: Vereinbarungssyntax ähnlich zu Ausdruckssyntax

  Bsp.:
  ```
  int *ip   // Zeiger auf int
  *ip       // int-Wert
  ```

  Referenzen: Vereinb.syntax $\neq$ Ausdruckssyntax

  Bsp.:
  ```
  int& j    // Referenz auf int
  j         // int-Wert
            // nicht mehr: &j int-Wert
  ```

- Realisierung von Referenzen auf Variablen mittels konstanter Zeiger denkbar:

  ```
  int& j = i;
  ```

  Umsetzung durch Compiler:

  ```
  int * const jp = &i
  ```
  Anlegen eines konst. Zeigers

  ```
  j → (*jp)
  ```
  Überall ersetzen

- Referenzen auch als Rückgabewerte von Funktionen möglich *(später!)*

Referenzen auf Konstanten („konstante Referenzen“)

Begriff

Referenz auf Konstante:

1. weiterer Name für eine Variable, deren Wert über die Referenz *nicht* geändert werden darf
2. weiterer Name für eine Konstante,
   ggf. wird eine Temporärvariable erzeugt

Der erste Name muss bei der Definition angegeben werden.

- *Bsp.*

```
int i=5;
const int& j=i; // j ander. Name f. i
const int& k=4; // k ander. Name f. 4
i++;            // i==6, j==6(!)
j++;            // verboten
k++;            // verboten
```

- Realisierung mittels konstanter Zeiger auf Konstanten denkbar.

Referenzen - Funktionswerte

Zweck

- Vermeidung unnötiger Kopien

  Voriges Beispiel:

```
typedef complex<double> Complex;
istream& operator>>(istream& stream, Complex& z)
```

  stream soll Ergebnis sein
  (keine Kopie, sondern *derselbe* Strom)

- Funktionsergebnis soll Schreibzugriff ermöglichen

  Beispiel:

```
vector<double> a(5); // a[i]==0 (i=0,..,4)
a.front() = 2;       // a[0]==2;
```

  Vereinbarung in `<vector>`: `T& front() { ... }`

Caveat:

- Bei Ergebnistyp Referenz: Rückgabe einer lokalen Variablen unzulässig