2 Grundlegende Begriffe, Teil 2

2.4 Anweisungen

Deklarationsanweisung,
Ausdrucksanweisung,
Verbundanweisung,
leere Anweisung,
while-Anweisung,
do … while-Anweisung,
for-Anweisung,
bereichsbasierte for-Anweisung
if-Anweisungen,
break-Anweisung,
continue-Anweisung,
switch-Anweisung.

2.5 Lambda-Ausdruck

Motivierendes Beispiel
Einfache Lambda-Ausdrücke
Parameter
Umgebungen
Funktor

Beispiel für Deklarationsanweisung:

```
void f()  {
        int x;
        /* ... */
        while (x) {
                /* ... */
        }
        int y  =  x + 5;
        /* ... */
}
```

Beispiele für Ausdrucksanweisungen:

```
y  =  x;
203;
++x;

a < b ? min = a : min = b;
```

Syntax der Ausdrucksanweisung:

Ausdrucksanweisung:

Ausdruck ;

Bemerkung: **Ein Ausdruck muß beim Erreichen des Semikolons ausgewertet sein, sein Wert wird vernichtet. Seine Nebeneffekte bleiben bestehen.**

Form:

Verbundanweisung:

{ Folge von Anweisungen (evtl, leer) }

Beispiel geschachtelter Verbundanweisungen:

```
{  double  result  =  0.0;
   int        status  =  0;

      {  double result = 2.0;
            cout  <<  result  <<  "    "  << status;
      }
      if  (status == 0)  {
      }  else  {
      }
}
```

Bemerkung: **Generell ist das letzte Symbol einer Anweisung ein Semikolon. Nur eine Verbundanweisung wird nicht durch ein Semikolon abgeschlossen.**

leere Anweisung:

Bemerkung: **Eine leere Anweisung wird oft aus syntaktischen Gründen benötigt, z. B. als Schleifenkörper.**

Beispiel:

Drei leere Anweisungen:

```
100 + x  ;      ;      ;      ;  200 - y
```

while-Anweisung:

Form: **while (Bedingung) Anweisung**

```
//  Beispiel für eine while-Schleife :

#include<iostream>
using namespace std;

int main () {
   long long  x,  y;
   //  Nutzer garantiert korrekte Eingabe
   cout <<"Bitte zwei Zahlen fuer GGT:  ";
   cin >> x >> y;
   cout  << "GGT (" << x << ", " << y << ")  =  ";

   x = x == 0 ? y : x;
   y = y == 0 ? x : y;
   x = x < 0 ? -x : x;
   y = y < 0 ? -y : y;
   while (x != y)
      if (x > y)
         x -= y;
      else
         y -= x;
   cout << x << endl;
}


/*  Beispiellauf
Bitte zwei Zahlen fuer GGT:
      9876543219876543211    87654328765432
GGT (9876543219876543211, 87654328765432)  =  143
*/
```

do ... while-Anweisung
Form: do Anweisung while (Bedingung)

```
// Programm zum Berechnen einer Primzahl,
// die auf eine gegebene Zahl folgt.

#include <iostream>
#include <cmath>
using namespace std;

int main() {
   cout << "Berechnung der ersten Primzahl, \ndie >="
        " der eingegebenen Zahl ist.\n";

   long z;
   std::cin.exceptions (std::ios_base::iostate (-1));
   do {
      cout << "Bitte Zahl > 3:  ";
      cin >> z;
   } while (z <= 3);  //  Abfrage zu simpel

   if (z % 2 == 0)
      ++z;

   bool gefunden = false;

   do {
      long limit = 1 + long(sqrt(z*1.0));
      long rest {};
      long  teiler {1};
      do   {
         teiler += 2;
         rest    = z % teiler;
      } while (rest > 0 && teiler < limit);
```

```
        if (rest > 0 && teiler >= limit)
           gefunden = true;
         else z += 2;
   } while (!gefunden);
   cout << "Die nächste Primzahl ist " << z << endl;
}


/*  Einige Läufe

Berechnung der ersten Primzahl,
die >= der eingegebenen Zahl ist.
Bitte Zahl > 3:  19
Die nächste Primzahl ist 19

Berechnung der ersten Primzahl,
die >= der eingegebenen Zahl ist.
Bitte Zahl > 3:  60
Die nächste Primzahl ist 61

Berechnung der ersten Primzahl,
die >= der eingegebenen Zahl ist.
Bitte Zahl > 3:  2147385555
Die nächste Primzahl ist 2147385587

Berechnung der ersten Primzahl,
die >= der eingegebenen Zahl ist.
Bitte Zahl > 3:  9876543210

This application has requested the Runtime to terminate it
in an unusual way.  Please contact the application's
support team for more information.
*/
```

for-Schleife
Form: for (Init-Anweisung Bedingung; Ausdruck)
Anweisung

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

int main() {
  int  n = -1;

  while  (n < 0) {
      cout << "Bitte Zahl >= 0 :  ";
      //  Nutzer garantiert korrekte Eingabe einer Zahl
      cin >> n;
  }
  unsigned long long fak = 1;

  for (int i = 2;  i <= n;  ++i)
    fak *= i;
  cout << n << "!   =   " << fak << endl;
}

/*  Einige Läufe:
Bitte Zahl >= 0 :  -6
Bitte Zahl >= 0 :  1
1!   =   1

Bitte Zahl >= 0 :  19
19!   =   121645100408832000
Bitte Zahl >= 0 :  20
20!   =   2432902008176640000
Bitte Zahl >= 0 :  21
21!   =   14197454024290336768   // fehlerhaft
*/
```

Bereichsbasierte for-Schleife

Form: for (Deklarator : Bereichs-Initialisierer) Anweisung

Beispiel:

```
#include <iostream>
#include <array>
#include <vector>
using namespace std;

int main () {
    cout << "Die ersten Kubikzahlen" << endl;
    for (int i : {1, 2, 3, 4, 5, 6, 7, 8, 9})
        cout << i*i*i << "   ";
     cout << endl << endl;

    array <int, 7> a = {11, 22, 33, 44, 55, 66, 77};
    cout << "Ausgabe des Arrays a" << endl;
    for (auto y : a) {
       cout << y << "   ";
    }
    cout << endl << endl;

    for (auto& y : a) {
        y = 2*y + 9;
    }

    cout << "Ausgabe des geaenderten Arrays a" << endl;
    for (auto y : a) {
       cout << y << "    ";
    }
    cout <<endl <<  endl;
```

```
    // Nutzung von vector
    vector<double>  v;
    for (int i = 0;  i < 6  ; ++i) {
        v.push_back (i + 0.147);
    }

    cout << "Ausgabe des  Vektors v"  << endl;
    for (const auto &j : v ) {
        cout << j << "   ";
    }
    cout << endl;
}//main


/*
Die ersten Kubikzahlen
1  8  27  64  125  216  343  512  729

Ausgabe des Arrays a
11  22  33  44  55  66  77

Ausgabe des geaenderten Arrays a
31   53   75   97   119   141   163

Ausgabe des Vektors v
0.147  1.147  2.147  3.147  4.147  5.147
*/
```

if-Anweisung

Man kennt zwei Formen der if-Anweisung.

- **(i) if (Bedingung) Anweisung**
- **(ii) if (Bedingung) Anweisung else Anweisung**

Die Bedingung muß einen Wahrheitswert – wahr oder falsch – liefern.

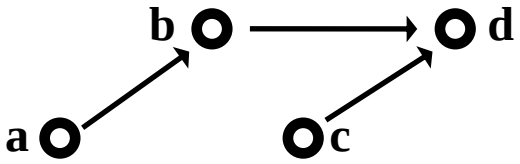
Beispiel:

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

array <int, 6> ai {22, 55, -61, -30, -61, -61};

int main () {
   int minv = ai [0];
   int vorhanden {1};
   int i {};
   while (++i < ai.size()) {
      if (minv < ai [i]) {
         // leer
      } else if (minv == ai [i]) {
         ++vorhanden;
      } else {
         vorhanden = 1;
         minv = ai [i];
      }
   }
```

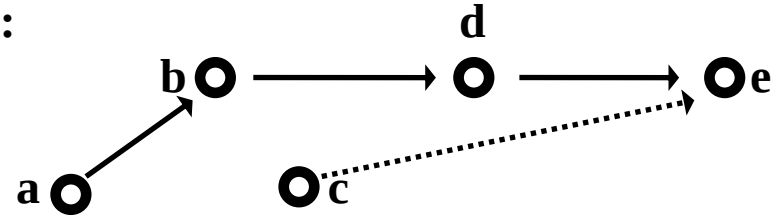
```
cout << "Minimalwert ist " << minv
        << " und erscheint " << vorhanden
        << "-mal in ai.";
}
/* Ausgabe:
Minimalwert ist -61 und erscheint 3-mal in ai.
*/
```

Ein zweites Beispiel: Sortieren von 5 Zahlen a, b, c, d, e durch 7 Vergleiche, dies ist die Minimalzahl.

Bild: Seien $a \le b$, $c \le d$ und $b \le d$:



Einfügen von e durch 2 Vergleiche in die Kette $a \le b \le d$.
Damit:



Durch zwei weitere Vergleiche wird c in die Kette $a \le b \le d \le e$ eingefügt.

Programmfragment zum Sortieren von 5 Zahlen durch 7 Vergleiche:

```
if  (a > b) {
      int h = a;  a = b;  b = h;
}
assert  (a <= b);
```

```
if  (c > d) {
      int h = c;  c = d;  d = h;
}
assert  (c <= d);

if  (b > d) {
      int h = b;  b = d;  d = h;
          h = a;  a = c;  c = h;
}
assert (a <= b && b <= d && c <= d);

if  (b > e)
      if (a > e) {
            int  h = e;  e = d;
                 d = b;  b = a;  a = h;
      } else {
            int h = e;  e = d;  d = b;  b = h;
      }
else if (d > e) {
      int h = e;  e = d;  d = h;
}
assert (a <= b && b <= d && d <= e && c <= e);

if  (c > b) {
      if  (c > d) {
            int h = c;  c = d;  d = h;
      }
} else if (c > a) {
      int  h = b;  b = c;  c = h;
} else {
      int  h = c;  c = b;  b = a;  a = h;
}
assert (a <= b && b <= c && c <= d && d <= e);
```

Beispiel zu break:

Die Ausführung einer Break-Anweisung führt zum Verlassen der nächstumschließenden Schleifen- oder Verteiler-Anweisung.

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

int main () {
   for (int i = 0;  i < 5;  ++i) {
      int j = i;
      while (true) {  //  Endlosschleife
         ++j;
         cout << "j = " << j << "   ";
         if (j > 2*i) break;
      }
      cout << endl;
   }
}//main


/*
Ausgabe:

j = 1
j = 2  j = 3
j = 3  j = 4  j = 5
j = 4  j = 5  j = 6  j = 7
j = 5  j = 6  j = 7  j = 8  j = 9
*/
```

```
//  Beispiel zu continue:
//  Zählen von Nichtziffern


#include <iostream>

int const size   {5};
char* zk [size]  =  {"abc", "a1b2c3", "alpha0",
                         "beta1", "9876543"};

int main () {
  int   bu_zahl  {};
  for (int i = 0; i < size;  ++i)
    for (char* cp = zk [i]; *cp != '\0';  ++cp) {
      if (*cp >= '0' && *cp <= '9')
        continue;
      ++bu_zahl;   // Wird ab und zu übersprungen
    }
  std::cout << "Zahl der Nichtziffern = "
              << bu_zahl << std::endl;
}


/*  Ausgabe:

Zahl der Nichtziffern = 15

*/
```

```
//  Verteiler-Anweisung  switch (param)
#include<iostream>
using namespace std;

int main () {
  int a;
   char c;
   cout << "Wandlung roemischer Zahlzeichen" << endl;
   cout << "Zeichen = ";
   cin >> c;    cout << c;
   switch (c) {
      case 'I'      : a = 1;        break;
      case 'V'      : a = 5;        break;
      case 'X'      : a = 10;       break;
      case 'L'      : a = 50;       break;
      case 'C'      : a = 100;      break;
      case 'D'      : a = 500;      break;
      case 'M'      : a = 1000;     break;
      default       : a = 0;
   }
   if  (a > 0)    cout << " entspricht " << a << endl;
   else           cout <<" ist kein römisches Zahlzeichen!"
                       << endl;
}//main

/*  Einige Läufe:
Wandlung roemischer Zahlzeichen
Zeichen = D
D entspricht 500
Wandlung roemischer Zahlzeichen
Zeichen = B
B ist kein römisches Zahlzeichen!
*/
```

Ein Programmiertrick von Tom Duff,
"Duff's Device"

Ziel: Reduzierung des Schleifenoverheads

Original:

```
send(to, from, count)
        register short *to, *from;
        register count;
        {
                register n = (count+7)/8;
                switch (count%8){
                case 0:         do{     *to = *from++;
                case 7:                 *to = *from++;
                case 6:                 *to = *from++;
                case 5:                 *to = *from++;
                case 4:                 *to = *from++;
                case 3:                 *to = *from++;
                case 2:                 *to = *from++;
                case 1:                 *to = *from++;
                                } while (--n>0);
                }
        }
```

```
//  Eine Anwendung
//  Berechnung einer Summe

#include <iostream>

int main () {
    int  anzahl  =  10;
    int summe      {};
    int i              {1};

    int n = (anzahl + 3) / 4;

    switch (anzahl % 4) {
      case 0:       do { summe += i++;
      case 3:            summe += i++;
      case 2:            summe += i++;
      case 1:            summe += i++;
                    } while(--n > 0);
    }

     std::cout << summe << std::endl;
}//main

// Ergebnis = 55
```

Motivierendes Beispiel aus Standard für Lambda-Ausdruck

```
#include <algorithm>
#include <cmath>
#include <array>
#include <iostream>

void abssort (float* x, unsigned N) {
   std::sort (x,  x + N,
       [] (float a, float b) {
          return  std::abs(a) < std::abs(b);
       });
}

std::array <float, 7>  a {2.1, -3.3, 4.5, -2.4, -5.3, 7.3, -1.6};

int main () {
   std:: cout << "unsortiert :  ";
   for (auto i : a)
      std::cout << i << "   ";
   std::cout << std::endl;
   std::cout << "sortiert :    ";
   abssort ((float*)&a, 7);
   for (auto i : a)
      std::cout << i << "   ";
   std::cout << std::endl;
}//main

/*
unsortiert :   2.1  -3.3  4.5  -2.4  -5.3  7.3  -1.6
sortiert :      -1.6  2.1  -2.4  -3.3  4.5  -5.3  7.3
*/
```

Einfache Lambda Funktionen

Lambda-Ausdrücke werden als einfache Funktionen für die Algorithmen der Standardbibliothek benötigt. Lambda-Ausdrücke beschreiben namenlose Funktionen.

Form: ["capture list"] (Parameter) weiter {Ausdruck ;}

Bemerkung: Die "capture list", (Parameter) und weiter müssen nicht vorhanden sein. Eingeleitet wird eine Lambda-Funktion mittels einer Liste (evtl. leer) in eckigen Klammern.

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

int main() {
   auto fun1 = [] {cout << "Eine erste Lambda-Funktion."
                  << endl;};
   fun1 ();                          // Lambda Funktion ausführen

   auto fun2 = []() -> void {cout << "Eine zweite Lambda-"
                "Funktion." << endl;};
   fun2 ();
}//main


/*
Eine erste Lambda-Funktion.
Eine zweite Lambda-Funktion.
*/
```

Nutzung von Lambda-Ausdrücken für Algorithmen der Standardbibliothek

```
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;

int main () {
	vector<int> vi  {3, -6, 8, -9, -2, 4, 21, -1, 18, -4};
	// Ein Lambda für jedes Vector-Element
	for_each (vi.begin(),  vi.end(),  [](int i) {
				cout << i << " ";});
	cout << endl;
	vector<int> orig = vi;

	// Verdopple jedes positive Element
	transform (vi.begin(),  vi.end(),  vi.begin(),
			[](int i) {return i > 0 ? 2*i : i;});
	for (auto i : vi){cout << i << " ";}   cout << endl;

	vi = orig;
	// Explizite Angabe des Rückgabetyps ist notwendig
	transform(vi.begin(), vi.end(), vi.begin(),
	    [](int i) -> int {if (i > 0) return 3*i; else return i;});
	for (auto i : vi) {cout << i << " ";}    cout << endl;
}//main

/*
3 -6 8 -9 -2 4 21 -1 18 -4
6 -6 16 -9 -2 8 42 -1 36 -4
9 -6 24 -9 -2 12 63 -1 54 -4
*/
```

Einsatz von Parametern

```
include <iostream>
using namespace std;

int main () {
   // Einfacher Lambda-Ausdruck
   [] {cout << "Einfacher Ausdruck!" << endl;} ();
   //  Nutzung in if-Anweisung
   if ([] (int i, int j) {return 2*i == j;} (11, 22))
      cout << "Bedingung erfuellt." << endl;
   else  cout << " Bedingung nicht erfuellt." << endl;

   //  Rückgabetyp muß angeben werden
   cout  << "Rueckgabe = " <<
      [] (int x, int y) -> int {
         if  (x > 5)
            return  x + y;
         else if (y < 2)
            return  x - y;
         else  return  x * y;
      }  (4, 3) << endl;
   // Rückgabetyp wird vom Compiler erkannt.
   cout << "Rueckgabe = " <<
      [] (double x, double y) {return x + y;} (3.5, 2.7)
      << endl;
}//main
/*
Einfacher Ausdruck!
Bedingung erfuellt.
Rueckgabe = 12
Rueckgabe = 6.2
*/
```

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


function<int (int, int)> f1 () {
    return [] (int x, int y) {
        return x*y;};
};

int main () {
    //  Addition zweier Ganzzahlen
    auto f2 = [] (int x, int y) {return x + y;};
    cout << "2 + 3 = " << f2 (2, 3) << endl;

    //  Funktionsobjekt
    function<int(int, int)> f3 = [](int x, int y) {return x + y;};
    cout << "3 + 4 = " << f3 (3, 4) << endl;

    auto lf = f1 ();
    cout << lf (6, 7) << endl;
}//main


/*
2 + 3 = 5
3 + 4 = 7
42
*/
```

```
//  Nutzung von Elementen der Umgebung


#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;


int main () {
   vector<int> vi {1, 2, 3, 4, 5, 6, 7, 8};

   int sum = 0;
   for_each (vi.begin(),  vi.end(),
          [&sum] (int elem) {
             sum += elem;
          });
   double mw =
   static_cast<double>(sum)/static_cast<double>(vi.size());
   cout << "Mittelwert  =  " << mw << endl;
}


/*
Mittelwert  =  4.5
*/
```

```
// Fünf verschiedene Arten auf die Umgebung von
// Lambdafuntionen zuzugreifen.

#include <iostream>
using namespace std;

void fcn1 () {
	size_t  v1 = 42;
	// Kopie der Variablen v1 wird nach f übertragen.
	auto f = [v1] {return v1;};
	v1 = 0;
	auto j = f (); // j = 42
	cout << "In fcn1  v1 = " << v1 << "    j = "
		<< j << endl;
}

void fcn2 () {
	size_t v1 = 42;
	// f enthält eine Referenz auf v1
	auto f = [&v1] { return v1; };
	v1 = 0;
	auto j = f (); // j = 0;
	cout << "In fcn2  v1 = " << v1 << "    j = "
		<< j << endl;
}

void fcn3 () {
	size_t v1 = 42;
	// Schlüsselwort mutable erlaubt Veränderung von
	// per Wert übergebenen Variablen.
	auto f = [v1] () mutable -> int {v1 += 10;  return v1;};
	v1 = 0;
```

```
    auto j = f ();   // j = 52
    cout << "In fcn3  v1 = " << v1 << "    j = "
         << j << endl;
}

void fcn4 () {
    size_t  v1 =  42;
    // Variable v1 wird per Referenz in f inkorporiert.
    // Wert von v1 änderbar durch f.
    auto f = [&v1] {return ++v1;};
    v1 = 0;
    auto j = f ();          // j = 1
    cout << "In fcn4  v1 = " << v1 << "    j = "
         << j << endl;
}

void fcn5 () {
    size_t v1 = 42;
    // p ist ein const pointer auf v1
    size_t* const p = &v1;
    // v1 wird inkrementiert
    auto f = [p] () {return ++*p;};
    auto j = f ();
    cout << "In fcn5  v1 = " << v1 << "   j = "
         << j << endl;
    v1 = 0;
    j = f ();
    cout << "In fcn5  v1 = " << v1 << "    j = "
         << j << endl;
}
```

```
int main() {
        fcn1();
        fcn2();
        fcn3();
        fcn4();
        fcn5();
}


/*   Ausgabe

In fcn1  v1 = 0      j = 42
In fcn2  v1 = 0      j = 0
In fcn3  v1 = 0      j = 52
In fcn4  v1 = 1      j = 1
In fcn5  v1 = 43     j = 43
In fcn5  v1 = 1      j = 1

*/
```

Syntax von Lambda-Ausdrücken:
["capture list"] (Parameter) weiter {Ausdruck ;}

Formen von "capture list":

[] **leere Liste**

[=] **Übernahme der gesamten Umgebung per Wert**

[&] **Übernahme der gesamten Umgebung per Referenz**

[this] **Zugriff auf alle Elemente der umschließenden Klasse**

[Namen] **Die Namen mit vorangestellten & werden per Referenz übernommen, die anderen per Wert.**

[=, Liste von Referenzen] **Übername der Umgebung per Wert, nur die aufgeführten Elemente werden perReferenz inkorporiert.**

[&, Liste von Bezeichnern] **Übername der Umgebung per Referenz, nur die aufgeführten Bezeichner werden per Wert inkorporiert.**

Beispiel zu Funktor:

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

template <typename T1, typename T2>
struct f2pow {
   T1 operator() (T1 base, T2 exp) const {
      return std::pow (base, exp);
   }
};

int main () {
   cout << "2**3 = " << f2pow<int, int>() (2, 3) << endl;
   cout << "5**4 = " << f2pow<int, int>() (5, 4) << endl;
   cout << "2.2**3 = " << f2pow<double, int>() (2.2, 3)
        << endl;
   cout << "5.5**4 = " << f2pow<double, int>() (5.5, 4)
        << endl;
}//main


/*  Ausgabe:
2**3 = 8
5**4 = 625
2.2**3 = 10.648
5.5**4 = 915.063
*/
```

Zweites Beispiel zu Funktor:

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

int main () {
   int        ivar {42};
   double dvar {3.14};
   class  lami {
   public:
      lami (int ivar, double dvar) :
            ivar (ivar), dvar (dvar) {}
       void operator() () {
          int i {9};
          cout << ivar << "   " << dvar << "   " <<  i << endl;
       };
   private:
      int       ivar;
      double dvar;
   };

   lami  x  (ivar, dvar);
   x ();
}//main


/*Ausgabe:
42    3.14    9
*/
```