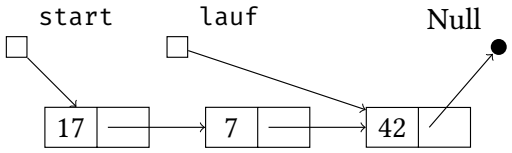
Verkettete Listen

list.cpp

```
#include <iostream>

using namespace std;

class Liste {
  public:
    double v;

  private:
    Liste* next;

  public:
    explicit Liste(double v_ = 0): v(v_), next(nullptr) {}

    static Liste* einlesen(string n) {
      Liste *start = nullptr, *lauf = start;

      cout << n << ": ";

      double x;
      while (cin >> x) {
        Liste* elem = new Liste{x};
        if (!start)
          start = lauf = elem;
        else {
          lauf->next = elem;
          lauf = elem;
        }
        if (cin.peek() == '\n') break;
      }

      return start;
    };

    friend ostream& operator<<(ostream& stream, const Liste& l) {
      for (const Liste *lauf = &l; lauf; lauf = lauf->next)
        stream << lauf->v << " ";
      return stream;
    }
};

int main() {
  Liste *liste = Liste::einlesen("l");
  if (!liste)
    return 2;

  cout << "l: " << *liste << endl;
  return 0;
}
```

```
l: 17 7 42 13
l: 17 7 42 13
```

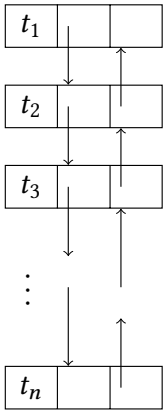
Speicherlayout verketteter Listen




```
Liste *start = nullptr,
  *lauf = start;

double x;
while (cin >> x) {
  Liste* elem = new Liste{x};
  if (!start)
    start = lauf = elem;
  else {
    lauf->next = elem;
    lauf = elem;
  }
  if (cin.peek() == '\n') break;
}
```

Verkettete Listen in STL (`list`)

- Datentyp `list<`T`>` für beliebiges T
- Doppelt verkettete Liste
- Einfügen/Löschen in konstanter Zeit
- Iteratoren bidirektional
- Kein wahlfreier Zugriff, insb. kein `operator[]`
- Verlängern/Verkürzen sowohl am Anfang wie auch am Ende:
  l`.push_front(`t`)`, l`.pop_front()`, l`.push_back(`t`)`, l`.pop_back()`
- Div. weitere Operationen als Methoden:
  `insert`, `erase`, `sort`, `unique`, `merge`, `splice`, `reverse`



Polynome mit verketteten Listen

Monom

```
class Monom {
  public:
    int pot;
    double v;

    Monom(double v_ = 0, int pot_=1): pot(pot_), v(v_) {}

    int grad() const { if (v != 0) return pot; else return -1; }

    friend bool operator<(const Monom& m1, const Monom& m2) {
      return m1.grad() > m2.grad();
    }
    friend bool operator==(const Monom& m1, const Monom& m2) {
      return m1.grad() == m2.grad();
    }

    friend ostream& operator<<(ostream& stream, const Monom& m)
    ↪  {
      return stream << showpos << m.v << "x^"
                    << noshowpos << m.pot;
    }
};
```

Polynome mit verketteten Listen

listpol.cpp

```
#include <iostream>
#include <list>
#include <utility>

using namespace std;
using namespace rel_ops;
```

Monom

```
class Polynom {
  private:
    list<Monom> ml;

  public:
    Polynom() {}
    Polynom(double v_, int pot_=1): ml({Monom{v_, pot_}}) {}

    Polynom& operator+=(Polynom q) {
      ml.merge(q.ml);
      if (ml.empty()) return *this;
      list<Monom>::iterator curr = ml.begin(),
        next = ++(ml.begin());
      for (; next != ml.end(); ++curr, ++next) {
        if (curr->pot == next->pot) {
          curr->v += next->v;
          *next = Monom{};
        }
      }
      ml.remove(Monom{});
      return *this;
    }

    friend ostream& operator<<(ostream& stream, const Polynom&
    ↪  p) {
      if (p.ml.empty()) return stream << "0";
      for (const auto& monom: p.ml)
        stream << monom;
      return stream;
    }
};

int main() {
  Polynom p, q;
  p += Polynom{4, 3}; p += Polynom{2, 2}; p += Polynom{1, 1};
  q += Polynom{1, 3}; q += Polynom{-2, 2}; q += Polynom{1, 0};

  cout << "p+q: " << (p+=q) << endl;

  return 0;
}
```

```
p+q: +5x^3+1x^1+1x^0
```

Paare (`pair`)

- Definiert in `<utility>`
- Datentyp `pair<`L`, `R`>` für beliebige L und R
- Objekte speichern jeweils einen Wert von Typ L und R
- Zugriff als Attribute `first`, `second`

Paare

```
pair<char, int> p{'A', 7};

cout << "("
     << p.first
     << ","
     << p.second
     << ")" << endl;
```

```
(A,7)
```

Generische Mengen in STL (`set`)

- Datentyp `set<`T`>` für beliebiges T
- Strikte Totalordnung auf T
- I.d.R. balancierter Binärbaum
- `insert`/`erase` in $O(\log(n))$
- Elemente nicht modifizierbar
- Iteratoren bidirektional
- Suchen in $O(\log(n))$ mit `find`, kein `operator[]`
- Mengenoperationen in `<algorithm>` einheitlich für alle Container-Datentypen

Sieb des Eratosthenes mit `set`

primes_set.cpp

```cpp
#include <set>
#include <iostream>

using namespace std;

const unsigned int n = 50;

int main() {
  set<unsigned int> nonprimes{0, 1};

  for (unsigned int k = 2; k*k < n; k++)
    if (!nonprimes.count(k))
      for (unsigned int i = k; i*k < n; i++)
        nonprimes.insert(i*k);

  for (unsigned int k = 0; k < n; k++)
    if (!nonprimes.count(k))
      cout << k << " ";
  cout << endl;

  return 0;
}
```

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
```

Mengen von natürlichen Zahlen in STL (`bitset`)

- Datentyp `bitset<`N`>` für beliebiges $N \in \mathbb{N}$
- I.d.R. Array von 64-bit Zahlen
- Menge von Werten aus $[0, N) \subset \mathbb{N}$
- Kein Einfügen/Löschen, stattdessen N viele Werte vom Typ `bool`
- Mitgliedschaft von Zahl $n < N$ setzen mit `set(`n`)` bzw. `reset(`n`)`
- Auslesen mit `test(`n`)`, `operator[]`, `all`, `any`, `none`, `count`
- Bitoperatoren `~`, `&`, `|`, `^`, `<<` und `>>`
- Ein-/Ausgabe als Zeichenkette von `0` und `1`

Sieb des Eratosthenes mit `bitset`

primes_bitset.cpp

```cpp
#include <bitset>
#include <iostream>

using namespace std;

const unsigned int n = 50;

int main() {
  bitset<n> primes;
  primes.set();
  primes.reset(0); primes.reset(1);

  for (unsigned int k = 2; k*k < n; k++)
    if (primes[k])
      for (unsigned int i = k; i*k < n; i++)
        primes.reset(i*k);

  for (unsigned int k = 0; k < n; k++)
    if (primes[k])
      cout << k << " ";
  cout << endl;

  return 0;
}
```

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
```

Endliche Abbildungen in STL (`map`)

- *Dictionary*, *Associative array*, *Symbol table*, *Zuordnungstabelle*
- Datentyp `map<`K`, `V`>` für beliebige K und V
- Strikte Totalordnung auf K
- I.d.R. balancierter Binärbaum
- `insert` von Paaren und `erase` in $O(\log(n))$
- Iteratoren bidirektional
- Suchen in $O(\log(n))$ mit `find`
- `operator[]` legt mit Standardkonstruktor Werte automatisch an ($O(\log(n))$)

Interaktive Erhebung von Übungspunkten mit `map`

input_exc.cpp

```cpp
#include <map>
#include <string>
#include <iostream>
#include <iomanip>
#include <fstream>

using namespace std;

int main(int argc, char* argv[]) {
  map<string, int> points;

  ifstream names(argv[1]);
  string s;
  while (names >> s) points[s];

  cout << "Teilnehmer  Punkte:" << endl;
  int n;
  while (cin >> s >> n) {
    if (points.find(s) != points.end())
      points[s] += n;
    else {
      char c;
      cout << "Neuer Teilnehmer (j/N)? "; cin >> c;
      if (c == 'j') points.insert(pair<string, int>{s, n});
    }
  }

  for (pair<string, int> entry: points)
    cout << setw(10) << entry.first
         << setw(3) << entry.second
         << endl;

  return 0;
}
```

teilnehmer.txt

```
Meyer
Mueller
Moser
Maier
```

```
Teilnehmer  Punkte:
Moser 3
Mueller 4
Maurer 2
Neuer Teilnehmer (j/N)? j
Moser 5
     Maier  0
    Maurer  2
     Meyer  0
     Moser  8
   Mueller  4
```

Worthäufigkeit mit `map`

freq.cpp

```cpp
#include <map>
#include <string>
#include <iostream>
#include <iomanip>
#include <fstream>
#include <vector>
#include <algorithm>

using namespace std;

bool ordnung(const pair<string, int>& a,
  const pair<string, int>& b) {
    return a.second > b.second;
}

int main(int argc, char* argv[]) {
  map<string, int> freq;
  int nwort = 0;

  ifstream ein(argv[1]);
  string wort;
  while (ein >> wort) {
    nwort++;
    freq[wort]++;
  }

  vector<pair<string, int>> vektor(freq.begin(), freq.end());
  stable_sort(vektor.begin(), vektor.end(), ordnung);

  int count = 0;
  for (auto entry: vektor) {
    cout << setw(10) << entry.first
         << setw(5) << entry.second
         << endl;
    if (++count >= 10) break;
  }

  cout << endl << "Wortzahl: " << setw(8) << nwort << endl;

  return 0;
}
```

```
         ;    7
         (    6
         )    6
         s    6
         b    5
         ,    4
        <<    4
         a    4
         c    4
         *    3

Wortzahl:       82
```