

**Einführung in die Programmiersprachen C und C++**

Dr. Franz Gähler, Fakultät für Mathematik

Übungsblatt 10 (8 Seiten)

Hier einige Beispiele zur Arbeit mit bereits definierten Klassen:

Die String-Klasse der C++ Standardbibliothek ist hier beschrieben:

<http://www.cplusplus.com/reference/string/>  
[http://www.cplusplus.com/reference/string/basic\\_string/](http://www.cplusplus.com/reference/string/basic_string/)

Hier ein Beispiel-Programm mit einigen Anwendungen aus der string Klasse mit einem Vergleich der Member-Funktion `compare` mit der C-Bibliotheksfunktion `strcoll`.

```
// strings.cc
using namespace std;
#include <iostream>
#include <locale>
#include <algorithm> // wegen reverse
#include <string.h>
int main() {
    string s(10, ' ');
    // Create a string of ten blanks.
    string t = "Äh";
    const char* A = "this is a test";
    setlocale(LC_ALL, "de_DE.utf8");
    s[0] = 'B';
    s[1] = 'c';
    s += A;
    cout << "s = " << (s + '\n');
    cout << "As a null-terminated sequence: " << s.c_str() << "\n";
    cout << "The sixteenth character is " << s[15] << "\n\n";
    cout << "Empty ? : " << s.empty() << "\n";
    cout << "Position von '\t' : " << s.find('\t') << "\n\n";

    cout << "compare : ";
    if (s.compare(t) < 0) // s < t ? Durch locale nicht beeinflusst
        cout << s << " < " << t << "\n";
    else
        cout << s << " >= " << t << "\n";

    cout << "strcoll : ";
    if (strcoll(s.c_str(), t.c_str()) < 0)
        cout << s << " < " << t << "\n";
    else
        cout << s << " >= " << t << "\n";

    cout << "Laenge von s : " << s.length() << "\n";
    cout << "Capacity von s : " << s.capacity() << "\n";
    cout << "strlen(s) : " << strlen(s.c_str()) << "\n";
    reverse(s.begin(), s.end());
    cout << "\nReverse : \n";
    s.push_back('\n');
    s.push_back('\n');
    cout << s;
}
```

*Aufgabe 10.1: Erproben Sie weitere Beispiele von Klassenfunktionen zur string Klasse*Die **GNU-Langzahlenbibliothek gmp** findet man hier:<http://gmplib.org/manual/>

Es ist leicht, z.B. mit beliebig langen Integers zu rechnen:

```
// lang.fak.cc
// Fakultaetsberechnung mit Langzahlen:
// Compilation mit: g++ lang.fak.cc -lgmpxx -lgmp
using namespace std;
#include <gmpxx.h>
#include <iostream>
#include <iomanip>

typedef mpz_class Int;

template <class T>
T fak (T i) {
    if (i <= 1) return 1;
    T r = 2;
    while (i > 2) r *= i--;
    return r;
}

// Rekursives Template cf. lang_fac_rec.cc
// Interessant hier die Eingabe i = 10000.

int main() {
    unsigned int i; Int k;
    while(1) {
        cout << "Eingabe : ";
        cin >> i;
        cout << "Fuer long long : ";
        cout << i << " != " << fak( (unsigned long long) i) << "\n";
        cout << "Fuer long double : ";
        cout << i << " != " << fak( (long double) i) << "\n";
        cout << "Fuer Int : ";
        cout << i << " != " << fak( (Int) i ) << "\n";
    }
}
```

*Aufgabe 10.2: Schreiben Sie Funktionen, die gg und kg fuer beliebig groÙe ganze Zahlen ermitteln.*

**Eine Klasse beliebig großer rationaler Zahlen:**

Als Beispiel die Teilsummen der harmonischen Reihe:

Es werden die Funktionen für `long double` und für `Rat` (beliebig große rationale Zahlen) verglichen.Für `long double` hängt die berechnete Summe von der Summationsreihenfolge ab!! (Nicht natürlich für `Rat`.)Die Ergebnisse für `Rat` wird zum Vergleich in (unbeschränkte) Fließdezimalzahlen umgewandelt.

```
// harmonisch.cc
// Demonstration von beliebig grossen rationale und Fliesskommazahlen
// Compilation mit: g++ harmonisch.cc -lgmpxx -lgmp
using namespace std;
#include <gmpxx.h>
#include <float.h>
#include <iostream>
#include <iomanip>

typedef mpq_class Rat;
typedef mpf_class Float;

Rat Harmonic(unsigned int n) {
    Rat r = 0;
    for(int i=1; i <=n; i++) {
        r += Rat(1,i);
    }
    return r;
}

Rat Harmonic1(unsigned int n) {
    Rat r = 0;
    for(int i=n; i >= 1; i--) {
        r += Rat(1,i);
    }
    return r;
}

// Die Summe ist fuer long double abhaengig von der Summationsreihenfolge:

long double harmonic(unsigned int n) {
    long double r = 0;
    for(int i = 1; i <= n; i++) {
        r += 1/(long double)i;
    }
    return r;
}

long double harmonic1(unsigned int n) {
    long double r = 0;
    for(int i = n; i >= 1; i--) {
        r += 1/(long double)i;
    }
    return r;
}
```

```
int main() {
    Rat h,k;
    unsigned int n;
    while(1) {
        cout << "Eingabe : ";
        cin >> n;
        cout << setprecision(23);
        h = Harmonic( n );
        cout << "Harmonic (" << n << ") = " << (Float)h;
        if (n<100) cout << " = " << h << "\n";
        else cout << " = " << "<Bruch>" << "\n";
        k = Harmonic1( n );
        cout << "Harmonic1(" << n << ") = " << (Float)k;
        if (n<100) cout << " = " << k << "\n";
        else cout << " = " << "<Bruch>" << "\n";
        cout << "harmonic (" << n << ") = " << harmonic( n ) << "\n";
        cout << "harmonic1(" << n << ") = " << harmonic1(n) << "\n";
    }
}

/*
Eingabe : 1000000
Harmonic (1000000) = 14.3927267228657236314
Harmonic1(1000000) = 14.3927267228657236314
harmonic (1000000) = 14.392726722865723355295
harmonic1(1000000) = 14.392726722865723646728
*/
```

*Aufgabe 10.3: Formulieren Sie die beiden Programme (aufsteigende, absteigende Summation) zur Berechnung der harmonischen Reihe als Template.*

Hier ein Programm, das einige Funktionen der GMP zur Untersuchung von Zahlen auf Primalität verwendet.  
Nach Übersetzung mit `g++ isprime.cc -o isprime -l gmpxx -l gmp` bewirkt z.B. ein Aufruf  
`./isprime 872453`

die Untersuchung der Zahl 872453 auf Primalität und die Ausgabe der nächstgrößeren Primzahl:

```
./isprime 872453
872453 ist prim
Naechste Primzahl: 872471
...
./isprime 123456791
123456791 ist wahrscheinlich prim
Naechste Primzahl: 123456803
```

Die Aussagen über Primalität sind meist "probabilistisch". Zahlen werden als "wahrscheinlich prim" bezeichnet, d.h., sie haben Primalitätstests bestanden, die nur sehr wenige Nicht-Primzahlen bestehen.

```
// isprime.cc
// g++ isprime.cc -o isprime -lgmpxx -lgmp
using namespace std;
```

```
#include <gmpxx.h>
#include <iostream>

typedef mpz::class Int;

Int nextprime (Int );
int prob_prime(Int, char);

int main(int argc, char *argv[]) {
    if (argc != 2) {
        cout << "Usage: " << argv[0] << " <Zahl>\n";
        return 1;
    }
    Int n = Int(argv[1]);
    cout << n;
    prob_prime(n, ' ');
    cout << "Nächste Primzahl: " << nextprime(n) << "\n";
}

string st[] = {
    " ist zusammengesetzt\n",
    " ist wahrscheinlich prim\n",
    " ist prim\n"
};

int prob_prime(Int n, char s) {
    int c;
    c = mpz_probab_prime_p (n.get_mpz_t(), 5);
    if ( s != 's') // nicht "schweigsam":x
        cout << st[c];
    return c;
}

Int nextprime (Int x) {
    Int n;
    mpz_nextprime( n.get_mpz_t(), x.get_mpz_t() );
    return n;
}
```

*Aufgabe 10.4:* Versuchen Sie den Nachweis, ob 123456791 wirklich prim ist. (Hinweis: Das Programm `primzahlen.c` von Blatt 2 könnte helfen. Können Sie das Programm `isprime.c` generell verbessern?

Hier ein Programm, das die Kombination einiger bisher betrachter Klassen und Programme verwendet. Das Programm `btree.cc` wird so erweitert, dass die in einem Baum gesammelten Daten (Paare vom Typ (`string, int`) von Wörtern und einer Häufigkeits-Angabe anschließend nach der Häufigkeit sortiert werden. Für das Sortieren wird die Implementation von `quicksort` von Blatt 6 verwendet.

Die Paare werden zu einer sortierbaren Klasse zusammengebunden:

```
class Pair {
public:
    string s; int n;
    friend bool operator<(Pair& x, Pair& y) {
        if (x.n == y.n) return (x.s < y.s);
        return (x.n < y.n);
    }
}
```

```
friend bool operator>(Pair& x, Pair& y) {
    if (x.n == y.n) return (x.s > y.s);
    return (x.n > y.n);
}
```

Nach dem Aufbau des Binärbaumes sollen sie zu einem durch `quicksort` sortierbaren Array zusammengefasst werden, dazu muss die Anzahl der Paare bekannt sein. Die lässt sich durch Mitzählen der `nodes` im Binärbaum in einer weiteren "Size." Komponente `s` von `struct node` erreichen. Dies geschieht über die neue Member-Funktion `void consist(node *n)`:

```
class tree {
    struct node{
        T e; int c; int s; node *l, *r;
        // Element, Zahler, Groesse, linker/rechter Teilbaum
        node() { c = s = 1; l = NULL; r = NULL; }
    } *n;
    friend int size (node *n) { return ( n ? n->s : 0 ); }
    void consist(node *n) {
        if (n) { n->s = 1 + size(n->l) + size(n->r); }
    }
}
```

Eine kleine Änderung in `node * insert()` ist noch nötig, außerdem eine `inorder`-Überladung zum Eintrag der Paare in einem Array:

```
friend Pair* inorder(Pair* p, tree t) {
    if (t.n) {
        p = inorder(p, t.n->l);
        p->s = t.n->e; p->n = t.n->c; p++;
        p = inorder(p, t.n->r);
    }
    return p;
}
```

Hier ist das gesamte Programm:

```
// btree_quicksort.cc : Binaerbaum + Quicksort, Version fuer de_DE.UTF-8

using namespace std;
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <wctype.h>
#include <iostream>
#include <locale.h>
#include "quicksort.h"

class Pair {
public:
    string s; int n;
    friend bool operator<(Pair& x, Pair& y) {
```

```

    if (x.n == y.n) return (strcoll( x.s.c_str(), y.s.c_str() ) < 0);
    return (x.n < y.n);
}
friend bool operator>(Pair& x, Pair& y) {
    if (x.n == y.n) return (strcoll( x.s.c_str(), y.s.c_str() ) > 0);
    return (x.n > y.n);
}
};

template<class T>
class tree {
    struct node{
        T e; int c; int s; node *l, *r;
        // Element, Zaehler, Groesse, linker/rechter Teilbaum
        node() { c = s = 1; l = NULL; r = NULL; }
        friend int size (node *n) { return ( n ? n->s : 0 ); }
    } *n;
    void consist(node *n) {
        if (n) { n->s = 1 + size(n->l) + size(n->r); }
    }
public:
    tree() { n = NULL; }           // Konstruktor : fuer Deklarationen
    tree(node *nn) { n = nn; }     // Konstruktor : fuer Initialisierungen
    friend int size(tree t) { return size(t.n); }
    node* insert(const T &e) { // Member-Funktion :
        if (n == NULL) {
            n = new node;          // liefert und initialisiert struct node n
            n->e = e;              // Datenelement
        }
        else if (e == n->e) { n->c++; } // e vorhanden, erhoehe Zaehler
        else if (strcoll( e.c.str(), n->e.c.str() ) < 0) // ordne lexikographisch
            n->l = tree(n->l).insert(e);
        else
            n->r = tree(n->r).insert(e);
        consist(n);
        return this->n;
    }
    friend ostream& inorder(ostream& os, tree t) {
        if (t.n) {
            inorder(os, t.n->l);
            os << t.n->e << '(' << t.n->c << ")\n" ;
            inorder(os, t.n->r);
        }
        return os;
    }
    friend Pair* inorder(Pair* p, tree t) {
        if (t.n) {
            p = inorder(p, tree(t.n->l));
            p->s = t.n->e; p->n = t.n->c; p++;
            p = inorder(p, tree(t.n->r));
        }
        return p;
    }
};

```

```

    }

};

int main() {
    char s[200]; string line;
    wchar_t w[200], wline[2000], *wlp, *wp;
    tree<string> tr;           // Deklariere Baum aus Strings
    setlocale(LC_ALL, "de_DE.UTF-8"); // cf. "man setlocale", "man locale"

    while (getline(cin,line)) {           // lies eine Zeile Text
        mbstowcs(wline,line.c_str(),2000); // konvertiere nach wide char
        wlp = wline;
        while (*wlp) {                  // bis zum Zeilenende ...
            while (*wlp && !iswalpha(*wlp)) wlp++; // finde Wortanfang
            if (!*wlp) continue;         // Zeile zu Ende
            wp = w;
            while (iswalpha(*wlp)) *wp++=*wlp++; // kopiere Wort
            *wp = L'\0';                // Wortende markieren
            wcstombs(s,w,200);          // Wort nach UTF-8 konvertieren
            tr.insert(s);               // in Baum einfuegen
        }
    }

    Pair *pp;
    cout << size(tr) << "\n";
    pp = new Pair[ size(tr) ]; // Anlage eines Arrays p von Pair

    cout << "Inorder arbeitet:\n";
    inorder(pp, tr); // Ausgabe der sortierten Woerter in Array

    cout << "Quicksort arbeitet:\n";
    quicksort(pp, size(tr));

    for (int i = 0 ; i < size(tr) ; i++) {
        cout << pp[i].n << " " << pp[i].s << "\n";
    }
}


```

Aufgabe 10.5: Modifizieren Sie das Programm btree\_qsort.cc so, dass die Liste der Wörter in der Reihenfolge absteigender Häufigkeiten ausgegeben wird.