

Einführung in die Programmiersprachen C und C++

Prof. Dr. Ulf Rehmann, Fakultät für Mathematik

Übungsblatt 10 (4 Seiten)

Beispiel für das Arbeiten mit einer Bibliothek

Die GNU-Multiple-Precision-Bibliothek GMP:

Diese Bibliothek für Langzahlen-Arithmetik ist sehr effizient in C geschrieben, die Funktionen sind allerdings ziemlich unhandlich, es lohnt sich eine Erschliessung durch einen C++-“Wrapper”.

Hierbei sollten Effizienz-Fragen ein wesentliche Rolle spielen.

Hier wird keine fertige Lösung angeboten, sondern es werden einige Hinweise und Werkzeuge gegeben (siehe GMP.h).

Aktuelle Version der Bibliothek auf Ftp-Servern, z. B. in prep.ai.mit.edu unter /gnu/. (Einloggen als User “anonymous”).

Das folgende gilt sinngemäß für die meiste GNU- und sonstige Public Domain Software.

Datei: `gmp-3.1.tar.gz` oder auch `gmp-3.1.tgz`. Auspacken mit `tar xzf gmp-3.1.tar.gz` liefert ein Verzeichnis `gmp-3.1` mit Quellcode und Dokumentation (Unterverzeichnis `doc`) sowie eine Datei `README` mit weiteren Hinweisen und eine Datei `INSTALL` mit Anleitung zur Installation.

Meist ist das Ausführen der folgenden Befehle erforderlich:

```
./configure      (Erstellen der “Makefiles”)
make             (Compilieren des Codes)
make install    (Installieren im System für den ständigen Gebrauch in Unix).
```

Der letzte Befehl muss in der Regel mit `root`-Berechtigung erfolgen.

In der Datei `INSTALL` findet sich der Hinweis auf die “Info”-Unterlagen. Im Kapitel über `integers` findet man Hinweise auf die definierten Typen und vorhandenen Funktionen:

Variable sind z. B. vor Benutzung zu initialisieren:

```
mpz_t pie;
mpz_init_set_str (pie, "3141592653589793238462643383279502884", 10);
...
mpz_sub (pie, ...);
...
mpz_clear (pie);
```

Hier sind die Initialisierung-Funktionen:

- Function: `void mpz_init_set (mpz_t ROP, mpz_t OP)`
- Function: `void mpz_init_set_ui (mpz_t ROP, unsigned long int OP)`
- Function: `void mpz_init_set_si (mpz_t ROP, signed long int OP)`
- Function: `void mpz_init_set_d (mpz_t ROP, double OP)`

Als C++-Funktion kann man überladen. Der Typ `mpz_t` ist eine “Referenz”-Version von `MP_INT`, siehe die

Datei `gmp.h`:

In `GMP.h`:

```
typedef unsigned long ulong;
typedef unsigned int  uint;
```

```
inline void mpz_set (MP_INT *i, long  a) { mpz_set_si(i,a); }
inline void mpz_set (MP_INT *i, int   a) { mpz_set_si(i,a); }
inline void mpz_set (MP_INT *i, ulong a) { mpz_set_ui(i,a); }
inline void mpz_set (MP_INT *i, uint  a) { mpz_set_ui(i,a); }
inline void mpz_set (MP_INT *i, double a) { mpz_set_d (i,a); }
inline void mpz_set (MP_INT *i, char  *s) { mpz_set_str(i,s,10);}
```

Diese sollten in die C++-Konstruktoren aufgenommen werden. Zur elementaren Arithmetik gibt es:

- Function: `void mpz_add (mpz_t ROP, mpz_t OP1, mpz_t OP2)`
- Function: `void mpz_add_ui (mpz_t ROP, mpz_t OP1, unsigned long int OP2)`
Set ROP to `OP1 + OP2`.
- Function: `void mpz_sub (mpz_t ROP, mpz_t OP1, mpz_t OP2)`
- Function: `void mpz_sub_ui (mpz_t ROP, mpz_t OP1, unsigned long int OP2)`
Set ROP to `OP1 - OP2`.
- Function: `void mpz_mul (mpz_t ROP, mpz_t OP1, mpz_t OP2)`
- Function: `void mpz_mul_si (mpz_t ROP, mpz_t OP1, long int OP2)`
- Function: `void mpz_mul_ui (mpz_t ROP, mpz_t OP1, unsigned long int OP2)`
Set ROP to `OP1 times OP2`.
- Function: `void mpz_addmul_ui (mpz_t ROP, mpz_t OP1, unsigned long int OP2)`
Add `OP1 times OP2` to ROP.
- Function: `void mpz_mul_2exp (mpz_t ROP, mpz_t OP1, unsigned long int OP2)`
Set ROP to `OP1 times 2 raised to OP2`. This operation can also be defined as a left shift, `OP2` steps.
- Function: `void mpz_neg (mpz_t ROP, mpz_t OP)`
Set ROP to `-OP`.
- Function: `void mpz_abs (mpz_t ROP, mpz_t OP)`
Set ROP to the absolute value of `OP`.

Diese Funktionen sollten durch Überladen von C++-Operatoren erschlossen werden:

```
inline void mpz_add (MP_INT *z, MP_INT *x, ulong a){ mpz_add_ui(z,x,a); }
inline void mpz_add (MP_INT *z, MP_INT *x, long  a){
    a>=0 ? mpz_add_ui (z,x,a) : mpz_sub_ui (z,x,-a);
}
inline void mpz_add (MP_INT *z, MP_INT *x, uint a){ mpz_add_ui (z,x,a); }
inline void mpz_add (MP_INT *z, MP_INT *x, int  a){ mpz_add (z,x,(long) a); }
...

// Addition
friend Int operator+(const Int& x, const Int& y) {
    Int z; mpz_add (z.p->i, x.p->i, y.p->i); return z;
}
template<class T> friend Int operator+(const Int& x, T y) {
    Int z; mpz_add (z.p->i, x.p->i, y); return z;
};
template<class T> friend Int operator+( T y, const Int& x) { return x+y; }
```

Die Langzahlen selbst sollten mittels Referenz-Zähler verwaltet werden, um Kopieraufwand zu vermeiden. Jedes Datenobjekt erhaelt eine Zähler-Komponente n:

```
class Int {
    struct Irep {
        MP_INT *i;
        int n;
        Irep() { n = 1; mpz_init(i = new MP_INT); } //Konstruktor
        ~Irep() { mpz_clear(i); delete i; }          //Destruktor
    };
    Irep *p;
public:
```

Dieser Zähler führt Buch über die Anzahl der Referenzen, die auf dies Objekt zeigen. Auf diese Weise wird z. B. bei einer Zuweisung `Int x,y; ... y = x;` nicht das ganze Datenobjekt kopiert, sondern `y` zeigt lediglich auf das Datenobjekt von `x`, es muss also der Zähler des Datenobjektes, auf den `y` vor der Zuweisung zeigte, dekrementiert und der Zähler des Datenobjektes, auf den `x` zeigt, inkrementiert werden. Falls bei der Dekrementierung 0 erreicht wird, kann das Datenobjekt gelöscht werden, so dass der Bereich für andere Zwecke frei wird. Ferner muss der Zeiger von `x` nach `y` kopiert werden. Dies drei Operationen sind schneller als die evtl Kopie von einigen -zig Byte für die Zahl selbst. Hier der Zuweisungscode aus `GMP.h`:

```
Int operator=(const Int& x) {
    x.p->n++; if (--p->n == 0) { delete p; }
    p = x.p; return *this;
}
```

Ähnlich sind andere Operatoren zu behandeln.

Hier ein Anwendungsprogramm mit Timing: Die Laufzeit selbstdefinierter Funktionen (Fakultätsberechnug) wird mit der der in `gmp` eingebauten Funktion "factorial" verglichen, um das Overhead durch Konstruktoren und Operatoren zu beurteilen:

```
// gmp.cc
// erfordert Gnu Multiple Precision Library (GMP)
//
// zu uebersetzen mit g++ gmp.cc -I/vol/gnu/include -lgmp
```

```
#include <stdio.h>
#include <iostream.h>
#include "Timer.h"
#include "GMP.h"
```

```
typedef long unsigned int ulong;
```

```
Int fakultaet(ulong n) {
    return n <= 1 ? 1 : n * fakultaet(n-1);
}

Int fak1(ulong n) {
    Int res = 1;
    while ( n>1 ) { res = res * n--; }
    return res;
}

Int fak2(ulong n) {
```

```
Int res = 1;
    while ( n>1 ) { res *= n--; }
    return res;
}

Int ggt(Int a, Int b) {
    a = abs(a), b = abs(b);
    return b==0 ? a : ggt(b,a%b);
}

Int timing( Int f(ulong), ulong n, char *s) {
    Int x; int t;
    timer(0); x = f(n); t = timer(1);
    printf("Zeit (%s) : %d msec\n",s,t);
    return x;
}

// Definition eines Array, Einträge als int oder als C-Strings moeglich:
Int a[] = {7,11,-47,139,-2531,"549797184491917","-1111111111111111111111111111111111"};

main() {
    long unsigned int n;
    int i;
    Int ii = 1;
    cout << ii << "... \n";
    cout << "ggt(" << a[5] << ", " << a[6] << ") = " << ggt(a[5],a[6]) << "\n";

    for (i = 0; i < sizeof(a)/sizeof(Int) ; i++) {
        ii *= a[i];
        cout << i << " : " << ii << "\n";
    }
    cout << "Prod = " << ii << "\n";
    for (i = 0; i < sizeof(a)/sizeof(Int) ; i++) {
        ii = ii/a[i];
        cout << i << " : " << ii << "\n";
    }
    cout << "Prod = " << ii << "\n";
    //-----
    Int aa[4];
    while(1) {
        cout << " : "; cin >> n;
        if (n < 0) break;
        aa[0] = timing(factorial,n,"factorial");
        aa[1] = timing(fakultaet,n,"fakultaet");
        aa[2] = timing(fak1, n,"fak1 ");
        aa[3] = timing(fak2, n,"fak2 ");
        if ( ! (aa[0] == aa[1] && aa[1] == aa[2] && aa[2] == aa[3] ) ) {
            printf("ERROR");
        }
        cout << aa[0] << "\n" ;
        cout << aa[0].base(16) << "\n" ;
    }
}
```