Überladen von Operatoren

Vergleichsoperatoren

Bei Vergleichsoperatoren genügt es die Operatoren `<` und `==` zu überladen. Die restlichen Operatoren werden dann durch in der STL vorhandene Funktionstemplates erzeugt, wenn die Headerdatei `<utility>` eingefügt und die Namespace-Directive `using namespace rel_ops` verwendet wird.

Bsp.: Datumsvergleich basierend auf den ursprünglich für C entwickelten Datumsfunktionen

```
#include <iostream>
#include <iomanip>
#include <ctime>
#include <utility>

using namespace std;
using namespace rel_ops;

bool operator<(tm tm1, tm tm2)
{
  time_t t1=mktime(&tm1),t2=mktime(&tm2); // Unix: Sekunden seit 1.1.1970
  return t1<t2;
}

bool operator==(tm tm1, tm tm2)
{
  time_t t1=mktime(&tm1),t2=mktime(&tm2);
  return t1==t2;
}

ostream& operator<<(ostream& stream, const tm& tm)
{
  char tmpstring[] = "31.12.1999";
  strftime(tmpstring,sizeof tmpstring,"%d.%m.%Y",&tm);
  stream << tmpstring;
  return stream;
}

istream& operator>>(istream& stream, tm& tm)
{
  char c1,c2;
  int d,m,Y;
  stream >> d >> c1 >> m >> c2 >> Y;
  // Fehlerbehandlung rudimentaer
  if (d<1 || d>31 || m<1 || m>12 || Y<1970
          || c1!='.' || c2!='.') {
    stream.setstate(ios::failbit);
    return stream;
  }
  tm.tm_mday=d; tm.tm_mon=m-1; tm.tm_year=Y-1900;
  tm.tm_hour=0; tm.tm_min=0; tm.tm_sec=0; tm.tm_isdst=-1;
```

```
  if (mktime(&tm)==-1)
    stream.setstate(ios::failbit);
  return stream;
}

int main()
{
  tm tm1,tm2;
  cout << "Datum (dd.mm.yyyy): "; cin >> tm1;
  cout << "Datum (dd.mm.yyyy): "; cin >> tm2;
  cout << boolalpha;
  cout << tm1 << "  < " << tm2 << " : " << (tm1<tm2)  << endl;
  cout << tm1 << "  > " << tm2 << " : " << (tm1>tm2)  << endl;
  cout << tm1 << "  = " << tm2 << " : " << (tm1==tm2) << endl;
}
```

Inkrement- und Dekrementoperatoren

Das Überladen des Präfixoperators `++` erfolgt wie bei anderen Operatoren, d.h. `++`x wird in `operator++(`x`)` umgesetzt. Beim Postfixoperator `++` kommt hingegen die Operatorfunktion `operator(`x`,`k`)` mit einem zusätzlichen Parameter vom Typ `int` zum Einsatz, der beim Aufruf automatisch 0 gesetzt wird.

Beispiel: Folgedatum

```
        :
tm& operator++(tm& tm)          // Praefixoperator
{
  tm.tm_mday+=1;
  tm.tm_isdst=-1;
  mktime(&tm);
  return tm;
}

tm operator++(tm& tm1, int k)   // Postfixoperator
{
  tm tmold=tm1;
  tm1.tm_mday+=1;
  tm1.tm_isdst=-1;
  mktime(&tm1);
  return tmold;
}
       :
int main()
{
  tm tm;
  cout << "Datum (dd.mm.yyyy): "; cin >> tm;
  cout << "Datum:      " << tm++ << endl;
  cout << "Folgedatum: " << tm   << endl;
}
```

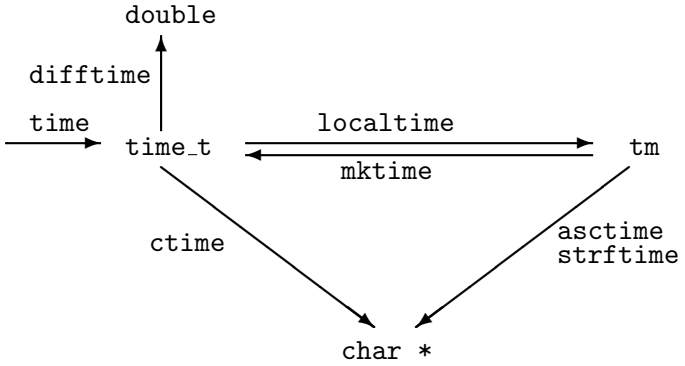
Ergänzung: Datumsfunktionen (<ctime>)

`time_t` ist ein implementierungsdefinierter Datentyp für die Kalenderzeit (*g++-9.4 unter Ubuntu Linux 20.04 amd64:* `long`)

Kalenderzeitfunktionen

Die kalendarische Zeiteinteilung ist über den Datentyp `tm` zugänglich, der die folgenden öffentlichen Komponenten besitzt:

`int tm_sec`	Sekunden (0-61)	`int tm_wday`	Tage seit Sonntag (0-6)
`int tm_min`	Minuten (0-59)	`int tm_yday`	Tage seit 1. Januar (0-365)
`int tm_hour`	Stunden (0-23)	`int tm_isdst`	> 0: Sommerzeit
`int tm_mday`	Tag im Monat (1-31)		$= 0$: keine Sommerzeit
`int tm_mon`	Monate seit Januar (0-11)		< 0: unbekannt
`int tm_year`	Jahre seit 1900		



Funktion	*Wirkung*
`time_t time(time_t *tp)`	Liefert akt. Zeit; falls `tp`$\neq$`0` wird das Ergebnis zusätzlich in `*tp` abgelegt.
`tm *localtime(const time_t *tp)`	Berechnet den der Zeit `*tp` entsprechenden Ortszeitrecord und liefert einen Zeiger darauf.
`time_t mktime(tm *tmptr)`	Liefert die dem Ortszeitrecord `*tmptr` entsprechende Zeit. (Die Komp. `tm_wday` und `tm_yday` werden ignoriert; die anderen Komp. brauchen nicht im eingeschränkten Bereich zu liegen. Falls `tm_isdst`< 0 wird zu bestimmen versucht, ob Sommerzeit gilt; andernfalls wird das aus dem Zahlenwert geschlossen. Bei Fehlern Rückgabewert -1.)
`double difftime(time_t t2, time_t t1)`	Liefert die Zeitdifferenz zwischen `t2` und `t1` in Sekunden.
`char *asctime(const tm *tmptr)`	Liefert zum Zeitrecord `*tmptr` eine Zeichenkette der Gestalt `Sun Sep 16 01:03:52 1973\n`
`char *ctime(const time_t *tp)`	Entspricht `asctime(localtime(tp))`

Funktion	*Wirkung*
`size_t strftime(char *s, size_t maxsize, const char *format, const tm *tmptr)`	Erzeugt zum Zeitrecord `*tmptr` eine gemäß `format` formatierte Zeichenkette, von der höchstens `maxsize` Zeichen in `s` abgelegt werden. `%a` Wochentag (abgekürzt) `%b` Monat (abgekürzt) `%c` Zeit- und Datumsdarstellung `%d` Tag im Monat (`01-31`) `%H` Stunde (`00-23`) `%m` Monat (`01-12`) `%M` Minute (`00-59`) `%S` Sekunde (`00-61`) `%x` Datumsdarstellung `%X` Zeitdarstellung `%y` Jahr ohne Jahrhundert (`00-99`) `%Y` Jahr mit Jahrhundert Liefert Anzahl der geschriebenen Zeichen.

Bsp.: n Tage nach aktuellem Datum

```
#include <iostream>
#include <ctime>
#include <vector>
#include <sstream>

using namespace std;

int main(int argc, char *argv[])
{
  vector<string> wtag = {"So","Mo","Di","Mi","Do","Fr","Sa"};
  char datum[] = "31.12.1999";
  tm *tmptr;
  int n;
  time_t t;
  istringstream ein;

  switch (argc) {
     case 1: n = 0; break;
     case 2: ein.str(argv[1]); ein >> n; break;
    default: cout << "usage: heute [+/-n]" << endl; return 1;
  }

  time(&t); tmptr = localtime(&t);
  tmptr->tm_isdst = -1; tmptr->tm_mday += n; t = mktime(tmptr);
  if (t!=-1) {
    strftime(datum,sizeof datum,"%d.%m.%Y",tmptr);
    cout << wtag[tmptr->tm_wday] << " " << datum << endl;
  } else {
    cout << "Fehler aufgetreten." << endl;
  }
  return 0;
}
```