BERGISCHE UNIVERSITÄT
GESAMTHOCHSCHULE WUPPERTAL
GAUSS-STRASSE 20
42097 WUPPERTAL
(Korrespondenzanschrift)
42119 WUPPERTAL
(Lieferanschrift)
TELEX 8 592 262 bughw
TELEFAX (0202) 439-2901
TELEFON (0202) 439-1

Fachbereich 7
MATHEMATIK

Prof. Dr. Hans-Jürgen Buhl
Praktische Informatik / Numerik

e-mail: Juergen.Buhl@math.uni-wuppertal.de

Informatik I

WS98/99 – Übungsblatt 6

Bearbeitungstermin: 50. Kalenderwoche

Aufgabe 1. *Methoden-Operatoren oder „friend-Operatoren"?*

Bringen Sie die beiden Versionen einer arithmetischen Klasse

```
//////////////////////////////////
// Datei:   comp0.cc
// Version: 1.0
// Zweck:   default constructor und Methoden-Operator
// Autor:   Hans-Juergen Buhl
// Datum:   17.09.1998
//////////////////////////////////

#include <iostream>

//using namespace std;

class comp {

  double re, im;

public:

  comp( const double a = 0, const double b = 0 ) :  re(a), im(b) {};

  comp operator+ (const comp) const;

  void print() const { cout << re << " + i * " << im; };

};

comp comp::operator+ (const comp c) const
{
  comp Erg(re + c.re, im + c.im);
  return Erg;
};
```



```
int main()
{
  comp x;
  comp x2(2.5, 1.5);

  x.print();
  cout << "    ";
  x2.print();
  cout << endl;

  x2 = x2 + 1;
  x2 = x2.operator+(1);
  x2.print();
  cout << endl;

  return 0;
}
```

und

```
//////////////////////////////////
// Datei:   comp1.cc
// Version: 1.0
// Zweck:   default constructor und friend-Operator
// Autor:   Hans-Juergen Buhl
// Datum:   17.09.1998
//////////////////////////////////

#include <iostream>

//using namespace std;

class comp {

  double re, im;

public:

  comp( const double a = 0, const double b = 0 ) :  re(a), im(b) {};

  friend comp operator+ (const comp, const comp);

  void print() const { cout << re << " + i * " << im; };

};

comp operator+ (const comp c1, const comp c2)
{
```

```
  comp Erg(c1.re + c2.re, c1.im + c2.im);
  return Erg;
};

int main()
{
  comp x;
  comp x2(2.5, 1.5);

  x.print();
  cout << "    ";
  x2.print();
  cout << endl;

  x2 = x2 + 1;
  x2 = x2.operator+(1);
  x2.print();
  cout << endl;

  x2 = 1 + x2;
  x2.print();
  cout << endl;

  return 0;
}
```

zum Ablauf. Diskutieren Sie Vor- und Nachteile. Ändern Sie die Methode `print()` so ab, daß mittels einer Fallunterscheidung Ausgaben der Form 1.0 + i * 0.2, 2.0, 3.1 - i * 0.9, i * 4.2, - i * 4.4 entstehen.

Aufgabe 2. *redundante Attribute*

Zur effektiven Impementierung benutzt man häufig redundante Attribute: Z.B. ist der Operator + mit Hilfe von Realteil und Imaginärteil einfach zu berechnen, während der Operator * mit Hilfe von Winkel und Länge einfacher berechnet werden kann. Ergänzen Sie das folgende Beispiel um die beiden friend-Operatoren + und * und testen Sie:

```
//////////////////////////////////
// Datei:   comp2.cc
// Version: 1.0
// Zweck:   redundante Attribute
// Autor:   Hans-Juergen Buhl
// Datum:   17.09.1998
//////////////////////////////////

#include <iostream>
#include        <math.h>

//using namespace std;
```



```
class comp {

  double re;
  double im;
                       // re == cos(Winkel) * Laenge
                       // im == sin(Winkel) * Laenge
  double Winkel;       // -Pi <= Winkel < +Pi,   {read only}
  double Laenge;       // >= 0.0                 {read only}

public:

  comp() : re(0.0), im(0.0), Winkel(0.0), Laenge(0.0) {};

  comp( const double r, const double i ) :  re(r), im(i),
                                            Winkel( atan2(im, re) ),
                                            Laenge( hypot(re, im) ) {};

  void print() const { cout << re << " + i * " << im; };

  void printPolar() const { cout << "Winkel= " << Winkel
                            << " Laenge= " << Laenge; };

};

int main()
{
  comp x;
  comp x2(-2.0, -0.000000001);

  double d(3.5);
  comp x3(d, 4);

  x.print();
  cout << "    ";
  x2.print();
  cout << "    ";
  x3.print();
  cout << endl;

  x.printPolar();
  cout << endl;
  x2.printPolar();
  cout << endl;
  x3.printPolar();
  cout << endl;

  return 0;
}
```

Aufgabe 3. *Design von Klassen*

Konzipieren Sie (in UML-Diagrammen) Klassen Adresse, Bestellnummer, Personal, Student, Seminarraum, Büro, Fachbereichsräume.

Aufgabe 4. *Meßung der CPU-Zeit eines Algorithmus*

Testen Sie das folgende Programm, um die CPU-Zeit einer Schleife zu messen:

```
//////////////////////////////////
// Datei:   TakeClock.cc
// Version: 1.0
// Zweck:   Zeitmessung von Algorithmenteilen,
//          Scientific Ausgabeformat von Gleitkommazahlen
// Autor:   Hans-Juergen Buhl
// Datum:   17.09.1998
//////////////////////////////////

#include <iostream>
#include <iomanip.h>
#include <math.h>

#include <time.h>
#include <unistd.h>

//using namespace std;

int main()
{
  clock_t StartTime(clock());

  // ...
  // sleep(2);
  double f(sin(1.4567));
  for (int i = 0; i<1000000; i++) f = sin(f);
  // ...

  clock_t Time(clock()-StartTime);

  cout <<  "Der Algorithmus dauerte "
       <<  setiosflags(ios::scientific)
       <<  setw(12) << setprecision(4)
       <<  static_cast<double>(Time)  / CLOCKS_PER_SEC
       <<  " CPU-Sekunden (Auflösung = "
       <<  resetiosflags(ios::scientific)
       <<  1.0 / CLK_TCK
       <<  " Sekunden)" << endl;

  return 0;
}
```


Vergleichen Sie die Ergebniszeit bei mindestens 5-maligem Start von `TakeClock`. Interpretieren Sie!

Entkommentieren Sie die Anweisung `sleep(2)`, übersetzen Sie erneut, und vergleichen Sie die Ergebnisse eines mehrmaligen Starts dieser Version untereinander und mit denen des Ablaufs der vorherigen Version.

Experimentieren Sie mit den aktuellen Parametern der Funktionsaufrufe `setw(12)`, `setprecision(4)`.