A b s c h l u ß k l a u s u r

zur Vorlesung

„Objektorientierte Programmierung in C++“

(Abschluß 2. Semester) 1998

Bitte notieren Sie auf allen Lösungsblättern Ihren Namen und einmalig Ihre Einschreibnummer. Anhand der letzteren werden am 14.7.98 (bei der letzten Vorlesung) die Ergebnisse der Klausur bekanntgegeben. Alle Studenten, deren Klausurergebnisse bei der Rückgabe unzureichend sind, werden aufgefordert, sich für eine Prüfungszulassung zu zusätzlichen Konsultationen anzumelden (Dr. Ahrens, Tel. 20 181 238). Die Prüfungen finden voraussichtlich am 28./29./ 30.9.98 statt.
Mit der vollständigen Lösung der folgenden 8 (+1) Aufgaben können insgesamt 78(+6) Punkte erreicht werden. Streben Sie bei der Auswahl der von Ihnen gelösten Aufgaben 20 Punkte an (entspricht etwa 2 kompletten Lösungen), um die Zulassung zur Prüfung zu erhalten!

Viel Erfolg !!!

---

1. Key Concepts [12 Punkte]

Welche Ausgaben produziert das folgende korrekte C++ -Programm?

```
#include <iostream>
#define DASH cout<<"-"
using namespace std;

void ctor(int & r){++r;}
void dtor(int & r){r--;}

class A {
     static int As;
public:
     A(){ ctor(As); v(); }
     ~A(){ dtor(As); v(); }
     void v(){ vv();}
     virtual void vv(){ cout<<As; }
} a;

class B: public virtual A {
     static int Bs;
     A a;
public:
     B(){ ctor(Bs); v();}
     ~B(){ dtor(Bs); v(); }
     virtual void vv(){ cout<<Bs; }
};

class C: virtual public A, public B {
     static int Cs;
public:
     C(){ ctor(Cs); v(); }
     ~C(){ dtor(Cs); v(); }
     virtual void vv(){ cout<<Cs; }
     B b;
};

int A::As=0, B::Bs=0, C::Cs=0;

int main() {
     DASH;     A * p = new C;
     DASH;     p->vv();
     DASH;     B b;
     DASH;     p = &b;
     DASH;     p->vv();
     DASH;
}
```

Hinweis: Die Ausgabe dieses Programmes ist zugleich die *decryption phrase* für die Entschlüsselung der vorbereiteten Beispiellösungen unter

`ftp://ftp.informatik.hu-berlin.de/pub/local/vorlesung/c++/2.klausur.98/loesungen.asc`

Auspacken wie immer mittels:

```
pgp loesungen.asc
[Enter pass phrase: ..... ]
[Should 'loesungen' be renamed to 'loesungen.gz' (Y/n)? n ]
tar xvzf loesungen
```

---

2. *Ulam*s Vermutung [12 Punkte]

Die noch immer unbewiesene Ulamsche Vermutung (Stanislaw Marcin Ulam (1909-1984)) besagt, daß die durch das Rekursionsschema

$$U_n(1) = n$$

$$U_n(k+1) = \begin{cases} \frac{U_n(k)}{2} & \text{falls } U_n(k) \text{ gerade} \\ 3 \cdot U_n(k) + 1 & \text{falls } U_n(k) \text{ ungerade} \end{cases}$$

definierte Zahlenreihe für jedes n im Zyklus 1, 4, 2, 1,.... endet. Sei L(n) die Länge der Ulamreihe für n bis die Zahl 1 erstmals erreicht wird:

$$L(n) = \min\{k, U_n(k) = 1\}$$ (z.B. L(1)=1, L(2)=2, L(3)=8, ... L(7)=17, ...)

Implementieren Sie eine Klasse `Ulam` in einem Programm, mit dessen Hilfe einerseits die Vermutung für alle Zahlen bis n=10000 (durch Terminierung des Programmes) unter Beweis gestellt wird und welches zugleich $L_{max}(n) = \max\{L(k), 1 \le k \le n\}$ (die maximale Länge einer Ulamreihe bis n) und die entsprechende Zahl, für die dieses Maximum erreicht wird für n=10, n=100, n=1000 und n=10000 ermittelt.

```
class Ulam { ....
public:
          Ulam(long int n); ....
};
```

Achten Sie darauf, daß keine *memory leaks* auftreten, damit die Berechnung der Ulamreihen nicht wegen Speichermangels vorzeitig beendet wird!

3. Kaffee und Milch [10 Punkte]

Schreiben Sie ein C++ -Programm, welches folgenden Vorgang nachbildet:

Aus einer Tasse voll Milch (100ml) wird ein Löffel voll (5ml) entnommen und in eine Tasse voll Kaffee (ebenfalls 100ml) gebracht. Nach dem Umrühren (homogene Durchmischung vorausgesetzt) wird erneut ein Löffel des Kaffee/Milch-Gemisches in die Milchtasse zurückgebracht. (die Tassen laufen nicht über, wenn man einen Löffel Flüssigkeit zufügt!)

und danach die Frage beantwortet: *Ist mehr Milch im Kaffee oder mehr Kaffee in der Milch?*

Verwenden Sie dabei das folgende Programmgerüst:

```
class Gefäss {
     double pKaffee;    // Anteil Kaffee in %
     double pMilch;     // Anteil Milch in %
     double menge;      // Inhalt
public: ....
};

class Tasse : public Gefäss {....};
class Löffel : public Gefäss {....};
class Tasse_voll_Milch : public Tasse {....};
class Tasse_voll_Kaffee : public Tasse {....};

int main() {
     Gefäss& K = *new Glass_voll_Kaffee;
     Gefäss& M = *new Glass_voll_Milch;
     Gefäss& L = *new Löffel;

     L=M-L;    // ein Löffelchen hin
     K=K+L;    // ein Löffelchen her und gut umrühren

     L=K-L;    // ein Löffelchen hin
     M=M+L;    // ein Löffelchen her und gut umrühren

     K.stat();      // wieviel Milch & Kaffee in % ?
     M.stat();      // wieviel Milch & Kaffee in % ?
}
```

4. Polynome [8 Punkte]

Implementieren Sie eine Klasse `Polynom` zur Darstellung von Polynomen n-ter Ordnung mit folgender Schnittstelle:

```
class Polynom { ....
public:
     Polynom(int order, double c[]);  // P(x)=c0+c1*x+c2*x^2+...+cn*x^n
     // c enthalte genügent (order+1) Koeffizienten; c[order]!=0
     ~Polynom();
     double eval(double x0);          // berechnet P(x0)
     Polynom derive();                // berechnet P'(x)
     friend ostream& operator<<(ostream& o, const Polynom& p);
     // Ausgabe in der Form:
     //        6+5*x+4*x^2+2*x^4+x^5
     //        (Koeffizienten und Potenzen 0 und 1 weglassen)

     friend Polynom operator+(const Polynom& p1, const Polynom& p2);
};
```

5. Stoppuhr [8 Punkte]

Implementieren Sie eine Klasse `Timer`, die folgende Schnittstelle zur Zeitmessung in C++-Programmen bereitstellt:

```
// Headerfile Timer
#ifndef __TIMER__
#define __TIMER__

class Timer {
     long micros;
     enum {stopped, running} state;
     static void err(char*);
public:
     Timer():state(stopped){}
     void start();
     void stop();
     double millisec();      // letzte Zeit zwischen start und stop in ms
     double sec();           // letzte Zeit zwischen start und stop in s
};

#endif
```

Fehlerhafte Benutzungen (`start` an einem laufenden Timer, `stop` an einem nicht laufenden Timer, Ablesen laufender Timer) sollen Fehlerausschriften erzeugen. Verwenden Sie Ihre Timerklasse anschließend, um in einem separaten Programm den Zeitverbrauch (in ms) von jeweils 10000000 Multiplikationen von `int`s und `double`s zu ermitteln. Schreiben Sie ein `makefile` welches ihr Testprogramm in Abhängigkeit der Files `Timer` und `Timer.cc` übersetzt.
Hinweis: Die ANSI C - Funktion `clock_t clock(void);` aus `time.h` liefert (unter Unix) die jeweils seit ihrem letzten Aufruf vergangene Zeit in Microsekunden als `long` (`clock_t`).

6. Schach für Anfänger [10 Punkte]

Implementieren Sie in einem Namensraum `Chess` die Klassen `Rook` (Turm), `Bishop` (Läufer) `Knight` (Springer) und `Queen` (Dame), die die grundlegenden Zugmöglichkeiten dieser Figuren repräsentieren. Ob Felder belegt sind oder andere Figuren im Weg stehen, soll dabei keine Rolle spielen. Gehen Sie aus von:

```
class Figure {
public:
     Figure(Pos p):where(p){}
     virtual ~Figure(){}
     virtual bool canMoveTo (Pos pos) const = 0;
     // kann die Figur sich nach p bewegen ?
     //   - ist pos auf dem Brett ?
```

```
     //   - steht die Figur nicht schon auf pos ?
     //   - ist pos mit den Mitteln dieser Figur erreichbar ?
     void move(Pos pos);          // falls möglich, bewege nach pos
protected:
     bool onBoard(Pos) const;     // gültiges Feld ?
     bool wouldMove(Pos) const;  // würde sich was bewegen ?
     Pos where;
};
```

Hinweis: Turm , Läufer und Springer sind Figuren. Eine Dame ist eine Figur, die ziehen kann **w**ie ein Turm oder **w**ie ein Läufer, aber **w**eder Turm noch Läufer ist !

---

7. Scrabble [8 Punkte]

Schreiben Sie ein C++ -**P**rogramm, **w**elches aus einer als **P**rogrammparameter übergebenen Zeichenkette alle zeichen**w**eisen **P**ermutationen berechnet und ausgibt. Versuchen Sie dabei nach Möglichkeit ohne Kopien der Ausgangszeichenkette auszukommen!

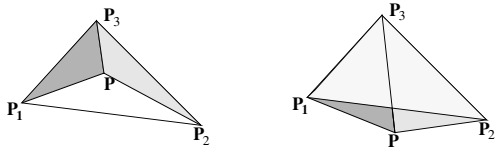
```
tempus ahrens 101 ( 2.klausur.98/7 ) > ./7 ABC
ABC
ACB
BAC
BCA
CBA
CAB
```

Hinweis: Das **P**roblem ist rekursiv gut lösbar; allerdings muß man darauf achten, daß bei der Ausgabe jedesmal der komplette (ggf. nur teil**w**eise) modifizierte String ausgegeben **w**ird und daß jedes Vertauschen von Zeichen rückgängig gemacht **w**ird, bevor die nächste Tauschaktion stattfindet. Ansonsten überlagern sich die einzelnen Tauschaktionen (vermutlich fehlerhaft).

---

8. Fehlertext [10 Punkte]

Finden Sie möglichst viele Fehler in dem folgenden vermeintlichen C++-**P**rogramm, **w**elches von einigen **P**unkten **P** entscheiden soll, ob sie in einem vorgegebenen Dreieck ($\mathbf{P_1}$, $\mathbf{P_2}$, $\mathbf{P_3}$) liegen. Die dabei verfolgte Grundidee ist die folgende: Falls **P** im Dreieck liegt, ist die Summe der Flächen der Dreiecke (**P**, $\mathbf{P_2}$, $\mathbf{P_3}$), ($\mathbf{P_1}$, **P**, $\mathbf{P_3}$) und ($\mathbf{P_1}$, $\mathbf{P_2}$, **P**) gleich der Fläche des Ausgangsdreiecks, ansonsten ist die Summe größer.



Für die Fläche eines Dreiecks mit den Eckpunkten $\mathbf{P_i}=(\mathbf{x_i}, y_i)$ gilt übrigens:

$$A = \left| \frac{1}{2} \cdot \begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix} \right|$$ (Betrag der **H**älfte der Determinate)

```
/*  1 */    #include <iostream>
/*  2 */
/*  3 */    class Point {
/*  4 */         double x, y;
/*  5 */    public:
/*  6 */         Point(double px, double py) : x(px); y(py)
/*  7 */    };
/*  8 */
/*  9 */    class Segment {
/* 10 */         Point* P = new Point[2];
/* 11 */    public:
/* 12 */         Segment (Point p1, Point p2) {
/* 13 */              P[1]=p1;
/* 14 */              P[2]=p2;
/* 15 */         }
/* 16 */         double operator double (void) {
/* 17 */              double x=P[0].x-P[1].x, y=P[0].y-P[1].y;
/* 18 */              retrun sqrt (((x*x + y*y)));
/* 19 */         }
/* 20 */    }
/* 21 */
/* 22 */    class triangle {
/* 23 */         Point P[3];
/* 24 */    public:
/* 25 */         Triangle (Point A, Point B, Point C);
/* 26 */         operator double() {
/* 27 */              return abs ( 0.5 *
/* 28 */                           (P[1].x-P[0].x)*(P[2].y-P[0].y)
/* 29 */                          -(P[1].y-P[0].y)*(P[2].x-P[0].x)
/* 30 */                          );
/* 31 */         }
/* 32 */         double area()
/* 33 */              return double(*this);
/* 34 */         }
/* 35 */         bool contains (const Point&) const;
/* 36 */    };
/* 37 */
/* 38 */    Triangle:Triangle (Point p1, Point p2, Point p3) {
/* 39 */         P[0]=p1; P[1]=p2; P[2]=p3;
/* 40 */    }
/* 41 */
/* 42 */    bool Triangle::contains (const Point p) {
/* 43 */         return    Triangle(P[0], P[1], p) +
/* 44 */                   Triangle(p, P[1], P[2]) +
/* 45 */                   Triangle(P[O], p, P[2]) == area ;
/* 46 */    }
/* 47 */
/* 48 */    int main() { /* ein 3eck und 4 Testpunkte
/* 49 */         t = new Triangle(Point(0,0), Point (2, 0), Point(0,2));
/* 50 */
/* 51 */         cout<<t.contains (new Point (1,1) <<endl;
/* 52 */         cout<<t.contains (new Point (2,2))<<endl;
/* 53 */         cout<<t.contains (new Point (.5, .5))<<endl;
/* 54 */         cout<<t.contains (new Point (1,0000001, 1))<endl;
/* 55 */    }
```

9. Zugabe: Templates ad infinitum ? [6 Punkte]

Was sind die Ausgaben des folgenden (korrekten) C++ -**P**rogramms:

```
#include <iostream>
template <int I> class B {
     public: B() {
          B<I>>1>();
          std::cout << I%2;
     }
};
template <> class B<0> {};
int main(){ B<99>(); return 0; }
```