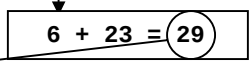
Lösung Klausur 4, infT — max. 23 Pkte.

1. 10 Pkte. (Auf dem Notebook zu entwickeln)
Gegeben sind folgende Tabellen eines immerwährenden Kalenders, um den Wochentag eines Datums im Bereich der Jahre 1901 bis 2040 zu ermitteln:

Jahre

					Monate											
					J	F	M	A	M	J	J	A	S	O	N	D
1901	1929	1957	1985	2013	2	5	5	1	3	6	1	4	0	2	5	0
1902	1930	1958	1986	2014	3	6	6	2	4	0	2	5	1	3	6	1
1903	1931	1959	1987	2015	4	0	0	3	5	1	3	6	2	4	0	2
1904	1932	1960	1988	2016	5	1	2	5	0	3	5	1	4	6	2	4
1905	1933	1961	1989	2017	0	3	3	6	1	4	6	2	5	0	3	5
1906	1934	1962	1990	2018	1	4	4	0	2	5	0	3	6	1	4	6
1907	1935	1963	1991	2019	2	5	5	1	3	6	1	4	0	2	5	0
1908	1936	1964	1992	2020	3	6	0	3	5	1	3	6	2	4	0	2
1909	1937	1965	1993	2021	5	1	1	4	6	2	4	0	3	5	1	3
1910	1938	1966	1994	2022	6	2	2	5	0	3	5	1	4	6	2	4
1911	1939	1967	1995	2023	0	3	3	6	1	4	6	2	5	0	3	5
1912	1940	1968	1996	2024	1	4	5	1	3	6	1	4	0	2	5	0
1913	1941	1969	1997	2025	3	6	6	2	4	0	2	5	1	3	6	1
1914	1942	1970	1998	2026	4	0	0	3	5	1	3	6	2	4	0	2
1915	1943	1971	1999	2027	5	1	1	4	6	2	4	0	3	5	1	3
1916	1944	1972	2000	2028	6	2	3	6	1	4	6	2	5	0	3	5
1917	1945	1973	2001	2029	1	4	4	0	2	5	0	3	6	1	4	6
1918	1946	1974	2002	2030	2	5	5	1	3	6	1	4	0	2	5	0
1919	1947	1975	2003	2031	3	6	6	2	4	0	2	5	1	3	6	1
1920	1948	1976	2004	2032	4	0	1	4	6	2	4	0	3	5	1	3
1921	1949	1977	2005	2033	6	2	2	5	0	3	5	1	4	6	2	4
1922	1950	1978	2006	2034	0	3	3	6	1	4	6	2	5	0	3	5
1923	1951	1979	2007	2035	1	4	4	0	2	5	0	3	6	1	4	6
1924	1952	1980	2008	2036	2	5	6	2	4	0	2	5	1	3	6	1
1925	1953	1981	2009	2037	4	0	0	3	5	1	3	6	2	4	0	2
1926	1954	1982	2010	2038	5	1	1	4	6	2	4	0	3	5	1	3
1927	1955	1983	2011	2039	6	2	2	5	0	3	5	1	4	6	2	4
1928	1956	1984	2012	2040	0	3	4	0	2	5	0	3	6	1	4	6



Wochentage

So	1	8	15	22	29	36
Mo	2	9	16	23	30	37
Di	3	10	17	24	31	
Mi	4	11	18	25	32	
Do	5	12	19	26	33	
Fr	6	13	20	27	34	
Sa	7	14	21	28	35	

Frage: Auf welchen Wochentag fällt der 23. April 2000?
Antwort: Auf Sonntag.
Lösung:
Suche in der Jahrestabelle die Jahrzahl 2000. Gehe auf der gleichen Zeile in die Monatstabelle zur Spalte April. Zähle zur gefundenen Zahl 6 die Zahl des Monatstages hinzu, ergibt 29. Gehe in die Wochentage-Tabelle und suche 29. Gehe auf der gleichen Zeile nach links und finde Sonntag.
Aufgabe:
Ergänzen Sie die gegebene Datei *EternalCalendar.cpp*, dass die Wochentage gefunden werden können.
Hinweis:
Am Inhalt der Arrays und an der Reihenfolge der Array-Elemente dürfen keine Änderungen vorgenommen werden.

L:

```
/* eternalCalendar.cpp */
#include <iostream>
#include <string>

using namespace std;


class EternalCalendar {
  static const int months[];
public:
  static int getWeekday(int, int, int);
};


const int EternalCalendar::months[] = { /*
  J  F  M  A  M  J  J  A  S  O  N  D
  -------------------------------  */
  2, 5, 5, 1, 3, 6, 1, 4, 0, 2, 5, 0,  // 1901  1929  1957  1985  2013
  3, 6, 6, 2, 4, 0, 2, 5, 1, 3, 6, 1,  // 1902  1930
  4, 0, 0, 3, 5, 1, 3, 6, 2, 4, 0, 2,  // 1903  etc.
  5, 1, 2, 5, 0, 3, 5, 1, 4, 6, 2, 4,  // etc.
  0, 3, 3, 6, 1, 4, 6, 2, 5, 0, 3, 5,  //
  1, 4, 4, 0, 2, 5, 0, 3, 6, 1, 4, 6,  //
  2, 5, 5, 1, 3, 6, 1, 4, 0, 2, 5, 0,  //
  3, 6, 0, 3, 5, 1, 3, 6, 2, 4, 0, 2,  //   |
  5, 1, 1, 4, 6, 2, 4, 0, 3, 5, 1, 3,  //   |
  6, 2, 2, 5, 0, 3, 5, 1, 4, 6, 2, 4,  //   |
  0, 3, 3, 6, 1, 4, 6, 2, 5, 0, 3, 5,  //   |
  1, 4, 5, 1, 3, 6, 1, 4, 0, 2, 5, 0,  //   |
  3, 6, 6, 2, 4, 0, 2, 5, 1, 3, 6, 1,  //   |
  4, 0, 0, 3, 5, 1, 3, 6, 2, 4, 0, 2,  //   |
  5, 1, 1, 4, 6, 2, 4, 0, 3, 5, 1, 3,  //   |
  6, 2, 3, 6, 1, 4, 6, 2, 5, 0, 3, 5,  //   |
  1, 4, 4, 0, 2, 5, 0, 3, 6, 1, 4, 6,  //   |
  2, 5, 5, 1, 3, 6, 1, 4, 0, 2, 5, 0,  //   |
  3, 6, 6, 2, 4, 0, 2, 5, 1, 3, 6, 1,  //   |
  4, 0, 1, 4, 6, 2, 4, 0, 3, 5, 1, 3,  //   |
  6, 2, 2, 5, 0, 3, 5, 1, 4, 6, 2, 4,  //   |
  0, 3, 3, 6, 1, 4, 6, 2, 5, 0, 3, 5,  //   |
  1, 4, 4, 0, 2, 5, 0, 3, 6, 1, 4, 6,  //   v
  2, 5, 6, 2, 4, 0, 2, 5, 1, 3, 6, 1,  //
  4, 0, 0, 3, 5, 1, 3, 6, 2, 4, 0, 2,  //
  5, 1, 1, 4, 6, 2, 4, 0, 3, 5, 1, 3,  //
  6, 2, 2, 5, 0, 3, 5, 1, 4, 6, 2, 4,  // 1927
  0, 3, 4, 0, 2, 5, 0, 3, 6, 1, 4, 6,  // 1928  1956  1984  2012 2040
  };

inline int EternalCalendar::getWeekday(int day, int month, int year) {
  return (months[((year - 1901) % 28) * 12 + (month - 1)] + day - 1) % 7;
}


class GermanNames {
  static const string monthname[];
  static const string dayname[];
public:
  static const string& getMonthname(int i) {
    return monthname[i];
  }

  static const string& getDayname(int i) {
    return dayname[i];
  }
};
```

```
const string GermanNames::monthname[] = {
  "",
  "Januar",
  "Februar",
  "Maerz",
  "April",
  "Mai",
  "Juni",
  "Juli",
  "August",
  "September",
  "Oktober",
  "November",
  "Dezember"
};


const string GermanNames::dayname[] = {
  "Sonntag",
  "Montag",
  "Dienstag",
  "Mittwoch",
  "Donnerstag",
  "Freitag",
  "Samstag"
};



void printGermanWeekday(int day, int month, int year) {
  cout << "Der " << day << ". " << GermanNames::getMonthname(month)
       << ' ' << year << " ist ein "
       << GermanNames::getDayname(EternalCalendar::getWeekday(day, month, year))
       << endl;
}
```

1

```
int main() {
  printGermanWeekday(1, 1, 1901);
  printGermanWeekday(1, 1, 1928);
  printGermanWeekday(25, 2, 1955);
  printGermanWeekday(23, 4, 2000);
  printGermanWeekday(26, 6, 2004);
  printGermanWeekday(8, 7, 2004);

  return 0;
}
```

Code läuft korrekt: 1

2. 5 Pkte. (Antworten der Frage 2 aufs Blatt schreiben)

a) Gegeben ist der C++-Code der Klassen-Definition für ein **`Singleton`**. Notieren Sie die Implementation.

```
class Singleton {
  static Singleton *instance;
  Singleton();
public:
  static Singleton *getInstance();
};

Singleton * Singleton::instance = 0;

inline Singleton::Singleton() { }

inline Singleton * Singleton::getInstance() {
  if (instance == 0) {
    instance = new Singleton();   // lazy initialization
  }
  return instance;
}
```

1

b) Erklären Sie, weshalb die Implementation des folgenden Copy-Constructors zu einem rekursiven Aufruf führt.

```
class GoodForNothing {
  int i;
public:
  GoodForNothing(const GoodForNothing gfn) : i(gfn.i) { }
};
```

1

L:
Der Parameter gfn ist nicht als Referenz angegeben. Beim Aufruf des Copy-Konstruktors findet daher ein Passing-by-value statt, was den Aufruf des Copy-Konstruktors zur Folge hat... also eine Rekursion. Dies kann verhindert werden wenn eine Referenz verwendet wird:

```
GoodForNothing(const GoodForNothing& gfn);
```

c) Gegeben sind folgende Klassen:

```
class A {    double a;
           public:
             double getA() { return a; }
             A(double a = 0) : a(a) { }    };
class B {    A *a;
           public:
             B(const A * const);
             // Copy Constructor
             ~B();                         };
```

Schreiben Sie die Implementation des **`B`** -Konstruktors, der das via Pointer übergebene Objekt als Kopie auf dem Heap anlegt. Es wird hier davon ausgegangen, dass kein Null-Pointer übergeben wird.

```
inline B::B(const A * const a) : a(new A(*a)) { }
```

1

Ergänzen Sie die Klasse **`B`** um den implementierten Copy-Konstruktor. Es wird davon ausgegangen, dass das Member nie einen Null-Pointer enthält.

```
inline B:: B(const B& b) : a(new A(*b.a)) { }
```

1

Implementieren Sie den Destruktor.

```
inline B::~B() { deleta a; }
```

1

3. 8 Pkte. (Auf dem Notebook zu entwickeln)
Im dreidimensionalen Raum ist jeder Punkt P durch die kartesischen Koordinaten x, y und z bestimmt, wobei es sich bei den Koordinaten um reelle Zahlen handle.

Definieren Sie die Klasse **`Point3D`** zur Darstellung eines Punktes. Die drei Koordinaten seien vom Typ **`double`**. Ausserdem sollen in der **`public`**-Schnittstelle folgende Funktionen deklariert/implementiert werden:

- Ein Konstruktor, der die Koordinaten initialisiert. Ohne Angabe von Initialwerten soll der Punkt im Ursprung angelegt werden.
- Ein unärer Operator **`operator-`**, der einen neuen Punkt Pn = -Pa mit (-x, -y, -z) liefert, wenn der alte Punkt Pa die Koordinaten (x, y, z) aufweist.
- Ein überladener Operator **`+=`** , der koordinatenweise die rechten Werte zu den linken Werten summiert.
- Ein überladener Operator **`-=`** in entsprechender Art (vgl. Operator **`+=`**).
- Eine Funktion **`toString()`**, die die Koordinaten eines Punktes in einen String der Form **`(x, y, z)`** bringt.
  Hinweis:
  Definieren Sie einen Stream der Klasse **`stringstream`** (in der Header-Datei **`sstream`** vorhanden). Mit dem Operator **`<<`** schreiben Sie die Zeichen und Koordinaten in den Stream. Die **`stringstream`**-Funktion **`str()`** wandelt den Stream in einen String.
- Überladen Sie für die Klasse **`Point3D`** auch den Operator **`<<`** der mit **`cout << p;`** einen Punkt an der Konsole anzeigt. Nutzen Sie die Funktion **`toString()`**.
- Überladen Sie die binären Operatoren **`+`** und **`-`**. Verwenden Sie dazu die Operatoren **`+=`** bzw. `-=`.

Hinweise:
Teilen Sie den Code in Header- und Implementations-Dateien auf, soweit erforderlich. Wo sinnvoll, formulieren Sie die Funktionen **`inline`**, nehmen eine Aufteilung in **`private`** und **`public`** vor und nutzen die Möglichkeiten von **`const`**ness.
Für Testzwecke erhalten Sie eine Datei *Point3DTest.cpp*.

```
/* Point3D.h */
#ifndef POINT3D_H
#define POINT3D_H

#include <sstream>
#include <iostream>

using namespace std;


class Point3D {
  double x, y, z;
public:
  Point3D(double x = 0, double y = 0, double z = 0) : x(x), y(y), z(z) { }

  Point3D operator-() const {
    return Point3D(-x, -y, -z);
  }

  Point3D& operator+=(const Point3D& p) {
    x += p.x;
    y += p.y;
    z += p.z;
    return *this;
  }

  Point3D& operator-=(const Point3D& p) {
    x -= p.x;
    y -= p.y;
    z -= p.z;
    return *this;
  }

  Point3D operator+(const Point3D& left, const Point3D& right) {
    Point3D p(left);
    return (p += right);
  }

  Point3D operator-(const Point3D& left, const Point3D& right) {
    Point3D p(left);
    return (p -= right);
  }

  string toString() const {
    stringstream ss;
    ss << '(' << x << ", " << y << ", " << z << ')';
    return ss.str();
  }
};


inline ostream& operator<<(ostream& os, const Point3D& p) {
  return os << p.toString();
}

#endif
```

Kein *Point3D.cpp*, da alles **`inline`** programmiert ist