Aufgabe 1: Geburtstag und Alter (5 Punkte)

Gegeben sei folgender `struct` für Kalenderdaten:

```
struct datum {
      int tag, monat, jahr;
};
```

Ferner existiere eine Funktion `heute`, die als Rückgabewert das heutige Datum als `datum`-struct hat. Der Funktionskopf ist `datum heute()` ;
Schreiben Sie folgende beiden Funktionen:

(a) Eine Funktion `geburtstag`, die als Argument das Geburtsdatum einer Person in Form eines `datum` erhält und unter Verwendung von `heute` ermittelt, ob die Person heute Geburtstag hat. Die Rückgabe soll als Logikwert erfolgen.
(b) Eine Funktion `alter`, ebenfalls mit einem Geburtsdatum-Argument, die auch unter Verwendung von `heute` das aktuelle Alter in Jahren ermittelt und zurückgibt. Entsprechend der üblichen Konvention ist das Lebensalter eine ganze Zahl, die die vollendeten Lebensjahre zählt.

Sie können stets voraussetzen, dass alle `datum` Parameter gültige Datumseinträge enthalten und dass das Geburtsdatum immer vor dem heutigen Datum liegt.

Aufgabe 2: Reelle symmetrische 2 x 2 Matrizen (9 Punkte)

Die Klasse `matrix` soll reelle symmetrische Matrizen beschreiben, also Matrizen der Form

$$\begin{pmatrix} a & b \\ b & c \end{pmatrix} \qquad \text{mit } a, b, c \in \mathbb{R}.$$

Die Variablen a, b und c sollen von außen unzugänglich sein. Als Schnittstellen sind folgende Konstruktoren, Methoden und freie Funktionen zu programmieren:

- Konstruktion einer Matrix mit den drei Parametern a, b, c
- Konstruktion einer Matrix ohne Angabe von Parametern
- Subtraktion zweier Matrizen
- Multiplikation einer Matrix mit einem Skalar, d.h. Multiplikation aller Parameter
- Berechnung der Determinanten (also $a \cdot c - b^2$)

Die Klasse kommt in folgendem Programm zum Einsatz:

```
#include <iostream>
using namespace std;

// Klasse matrix und freie Funktionen zu ergänzen

int main()
{
      matrix m1( 1.1, 2.2, 3.3) ; // Matrix mit Einträgen
      matrix m2( 0.1, 0.2, -0.7) ; // Matrix mit Einträgen
      matrix m3 ;
      m3 = m1 - m2 ; // Differenz zweier Matrizen
      m3 *= 2.5 ; // Multiplikation der Elemente mit einem Faktor
      cout << "Die Determinante von m3 ist " << det(m3) << endl ;
}
```

Schreiben Sie die benötigte Klasse `matrix` und alle benötigten Konstruktoren, Methoden und freie Funktionen, die in dem Hauptprogramm aufgerufen werden.

Aufgabe 3: Integration per Trapezregel (5 Punkte)

Die Trapezregel zur näherungsweisen Integralberechnung einer Funktion $f(x)$ mit $n+1$ Stützstellen $x_i, i = 0, ..., n$ lautet:

$$I = \frac{1}{2} \sum_{i=0}^{n-1} (f(x_{i+1}) + f(x_i))(x_{i+1} - x_i) \tag{1}$$

Schreiben Sie eine Funktion `integral`. Der Funktion wird beim Aufruf der Parameter n, ein Feld mit $n+1$ Komponenten mit den Stützstellen x_i und ein Zeiger auf eine Funktion `double f(double)` übergeben. Der obige Ausdruck ist damit zu berechnen und zurückzugeben.

Aufgabe 4: Auflösung einer Rekursion (4 Punkte)

Gegeben sei die folgende rekursive Funktion:

```
Type2 f (Type1 x)
{
      if ( test(x)) return q(x) ;
      return f(p(x)) ;
}
```

Dabei sind `Type1 p (Type1)`, `Type2 q (Type1)` sowie `bool test (Type1)` beliebige Funktionen, die hier für die Rechenoperationen mit dem Argument x stehen. Schreiben Sie, durch Verwendung einer Schleife, eine nicht-rekursive Formulierung der obigen Funktion f.

Aufgabe 5: Baumstruktur (7 Punkte)

```
struct element {
       double val;
       element* right;
       element* left;
};
```

Mit obiger Struktur seien Daten in einer unsortierten Baumstruktur organisiert, d.h. jeder Knoten kann zwei weitere Astknoten haben. Schreiben Sie die folgenden drei Funktionen:

- `getvalue`. Die Funktion soll als Argument einen Zeiger auf ein Baumelement erhalten und den Wert des Elements zurückgeben.
- `create`. Die Funktion soll als Argument einen `double`-Wert erhalten. Innerhalb der Funktion ist dynamisch ein neues Feld zu erzeugen, welches dann mit Argumentwert und Nullzeigern initialisiert wird, schließlich wird ein Zeiger auf das neue Element zurückgegeben.
- `minimum`. Diese Funktion soll als Argument einen Zeiger auf ein Element erhalten und im gesamten erreichbaren Baum das Minimum ermitteln und zurückgeben. (Der Baum ist nicht geordnet und muss deswegen komplett durchsucht werden.)

Sie können davon ausgehen, dass die Funktionen `getvalue` und `minimum` einen gültigen Zeiger auf ein Baumelement erhalten, dass am Ende des Baumes alle Astenden mit dem Nullzeiger markiert sind und dass nirgends geschlossene Pfade durch die Zeiger markiert sind.

1 Aufgabe 1

1.1 Aufgabe 1a[1]

```
1  bool geburtstag(datum* tag)
2  {
3      datum heutigertag = heute();
4          if ((tag->tag == heutigertag.tag) && (tag->monat == heutigertag.monat))
5              return true;
6          else return false;
7  }
```

1.2 Aufgabe 1b[2]

```
1  int alter(datum* geburtsdatum)
2  {
3      datum heutigertag = heute();
4      int alter = heutigertag.jahr - geburtsdatum->jahr;
5      if (geburtsdatum->monat > heutigertag.monat)
6          alter--;
7      else if ((geburtsdatum->monat >= heutigertag.monat) &&
8              (geburtsdatum->tag > heutigertag.tag))
9          alter--;
10     return alter;
11 }
```

[1] Der ein oder andere ist vielleicht darüber gestolpert, dass im Funktionskopf ein Pointer auf ein Element bergeben wird; in dem Fall wird die Funktion aus der main()-Funktion heraus mit `geburtstag(&tag)` aufgerufen. Die alternative Lösung wäre ohne Pointer, dass heißt: `bool geburtstag(datum tag)` als Funktionskopf und Aufruf über `geburtstag(tag)`.

[2] Analoge Begründung.

2 Aufgabe 2

```
class matrix{
    private:
        double a;
        double b;
        double c;
    public:
        matrix(double x, double y, double z);
        matrix();
        matrix operator-(matrix subtrahend);
        matrix operator*=(double skalar);
        friend double det(matrix);
};
matrix::matrix()
{
}
matrix::matrix(double x, double y, double z)
{
    a = x; b = y; c = z;
}
matrix matrix::operator-(matrix subtrahend)
{
    matrix differenz;
    differenz.a = a - subtrahend.a;
    differenz.b = b - subtrahend.b;
    differenz.c = c - subtrahend.c;
    return differenz;
}
matrix matrix::operator*=(double skalar)
{
    matrix ergebnis;
    ergebnis.a = a*skalar;
    ergebnis.b = b*skalar;
    ergebnis.c = c*skalar;
    return ergebnis;
}
double det(matrix m)
{
    return (m.a*m.c - m.b*m.b);
}
```

3 Aufgabe 3

```
double integral(double* stuetzstellen, int n, double (*f)(double wert))
{
    double summe = 0;
    int i;
    for (i = 0; i <= n-1; i++)
    {
        summe += 0.5 * (f(stuetzstellen[i+1])+f(stuetzstellen[i]))*
            (stuetzstellen[i+1]-stuetzstellen[i]);
    }
    return summe;
}
```

4 Aufgabe 4

```
Type2 funktion(Type1 x)
{
    while (! test(x))
        x = p(x);
    return q(x);
}
```

5 Aufgabe 5

5.1 Aufgabe 5a

```
double getvalue(element* zeiger)
{
    return zeiger->val;
}
```

5.2 Aufgabe 5b

```
element* create(double wert)
{
    element neu;
    element* p = &neu;
    neu.val = wert;
    neu.right = 0;
    neu.left = 0;
    return p;
}
```

5.3 Aufgabe 5c

```
double findekleinsteselement(double a, double b, double c)
{
    return (((a<=b)&&(a<=c)) ? a : (((b<a) && (b<c)) ? b : c));
}

double minimum(element* zeiger)
{
    double mitte = zeiger->val;
    double linkerwert = mitte;
    double rechterwert = mitte;
    if (zeiger->left != 0)
   linkerwert = minimum(zeiger->left);
    if (zeiger->right != 0)
   rechterwert = minimum(zeiger->right);
    return findekleinsteselement(linkerwert, mitte, rechterwert);
}
```