Aufgabe 1: McCarthy91-Funktion (2 Punkte)

Die McCarthy91-Funktion wurde 1970 von John McCarthy veröffentlicht. Sie ist als Beispiel für eine Rekursion, bei der es nicht einfach ist, mit formalisierten Methoden die Terminierung nachzuweisen. Sei n eine ganze Zahl, dann ist

$$M(n) = \begin{cases} n - 10 & \text{falls } n > 100 \\ M(M(n+11)) & \text{falls } n \leq 100. \end{cases}$$

Schreiben Sie eine C++-Funktion, die n als Argument hat und als Funktionswert $M(n)$ berechnet.

Aufgabe 2: Histogramm (5 Punkte)

Schreiben Sie eine Funktion `histogramm`, die Folgendes leistet: Die Funktion erhält ein Feld von Messwerten und die Länge des Feldes. Die Zahlen im Feld sind `double`-Werte und alle im Intervall $[0.5, 10.5)$.
Zählen Sie für jedes der zehn Intervalle $[n - 0.5, n + 0.5)$, $n = 1, 2, \ldots, 10$, wie häufig Zahlen in diesem Intervall vorkommen und geben Sie diese Häufigkeit auf dem Bildschirm aus.
Die Funktion habe keinen Rückgabewert.

Aufgabe 3: Rechteck-Klasse (8 Punkte)

Schreiben Sie eine Klasse `rechteck` mit folgenden Eigenschaften: Ein Rechteck wird durch eine ganzzahlige Breite und ganzzahlige Höhe charakterisiert. Diese Variablen sollen von außen unzugänglich sein. Programmieren Sie dazu folgende Methoden und freie Funktionen, so dass das untenstehende Programm funktioniert: (i) Setzen der Werte; (ii) Drehen eines Rechtecks um 90°, also Vertauschen von Breite und Höhe; (iii) Vergleich, ob zwei Rechtecke in der vorhandenen Orientierung identisch sind; (iv) Ermittlung der Länge der Diagonalen als `double`-Wert.

Die Klasse kommt in folgendem Programm zum Einsatz:

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

// Klasse rechteck und freie Funktion zu ergaenzen

int main()
{
    rechteck r1(4,5) ;   // Rechteck mit Breite, Höhe vereinbaren
    rechteck r2(5,4) ;    // ... und noch eines
    r1.rotiere() ;        // r1 wird um 90 Grad rotiert
    if (r1 == r2)
       cout << "r1 und r2 sind identisch." << endl ;
    cout << "Laenge der Diagonalen von r1: " << diagonale(r1) << endl ;
}
```

Schreiben Sie die benötigte Klasse `rechteck` und alle benötigten Konstruktoren, Mitgliedsfunktionen und freien Funktionen, die in dem gegebenen Hauptprogramm aufgerufen werden. Die Wurzel einer Zahl erhalten Sie mittels `sqrt`.

Aufgabe 4: Lexikographischer Vergleich (5 Punkte)

Schreiben Sie eine Funktion `lexicomp`, die Folgendes leistet: Die Funktion erhält als Argumente zwei `char`-Felder, die im folgenden Worte genannt werden, und deren Längen. Gültige Feldindizes gehen jeweils von 0 bis (*Länge* - 1). Die Funktion soll mit einem `int`-Rückgabewert folgendes ermitteln: -1, wenn das erste Wort in einem Wörterbuch eindeutig vor dem zweiten Wort einsortiert wäre; 1 im umgekehrten Fall und genau dann 0, wenn die beiden Worte identisch sind.
Sie können davon ausgehen, dass nur Kleinbuchstaben und keine Umlaute in den Worten vorkommen. Die Klasse `string` soll nicht verwendet werden.

Bsp: `welle` und `weltraum` $\longrightarrow -1$
`torte` und `tor` $\longrightarrow 1$
`positron` und `positron` $\longrightarrow 0$

Aufgabe 5: Was wird ausgegeben? (4 Punkte)

Setzen Sie im untenstehenden Programm an der markierten Stelle die letzte Ziffer Ihrer Matrikelnummer ein. Schreiben Sie dann *vollständig* auf, was von dem Programm ausgegeben wird.

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

struct list {
    list* p ;
    int i ;
} ;

list* f(int& j) {
    list* l = new list ;
    j += 5 ;
    l->i = j ;  // Hinweis: x->y ist identisch zu (*x).y
    l->p = 0 ;
    return l ;
}

list* f(list* p) {
    p->i = 19 ;
    int m = 12 ;
    p->p = f(m) ;
    return p->p ;
}

int main() {
    int m = 4 ;  // <- setzen Sie hier bitte die *letzte*
                 //    Ziffer Ihrer Matrikelnummer ein.
    list* p1 = f(m) ;
    cout << "Alpha: " << p1->i << endl ;
    list* p2 = f(p1) ;
    cout << "Beta: " << p1->i << endl ;
    cout << "Gamma: " << p1->p->i << endl ;
    cout << "Delta: " << m << endl << endl ;
}
```

Aufgabe 6: Befreundete Zahlen (6 Punkte)

Zwei *verschiedene* natürliche Zahlen heißen befreundet, wenn für beide Zahlen gilt: Die Summe der echten Teiler einer Zahl ergibt die andere Zahl. (Echte Teiler sind alle Teiler der Zahl außer der Zahl selbst.)

Bsp: 220 und 284 sind befreundet:
220 hat echte Teiler: 1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110; deren Summe ist 284
284 hat echte Teiler: 1, 2, 4, 71, 142; deren Summe ist 220

Schreiben Sie ein vollständiges Programm, das durch Ausprobieren alle befreundeten Zahlenpaare, bei denen eine Zahl kleiner als 10000 ist, ermittelt und ausgibt. Die einzelnen Teiler der Zahlen brauchen nicht auf dem Bildschirm zu erscheinen. Allerdings soll jedes gefundene befreundete Zahlenpaar nur einmal ausgegeben werden.

Zur Lösung wird die Zerlegung der Aufgabenstellung in geeignete Funktionen empfohlen. Kommentieren Sie die einzelnen Abschnitte in Ihrem Programm, so dass sofort verständlich ist, was jeweils gemacht wird. Ansonsten kann der entsprechende Programmteil nicht bewertet werden.

Bitte geben Sie die Nummern der bearbeiteten Aufgaben deutlich lesbar an

A1

```
int M(int n){
    if(n>100) return (n-10); ✓0.5
    return M(M(n+11)); ✓1.5
}
```

/2

A2

```
#include<iostream>
using namespace std;
void histogramm(double Feld[], int l){ ✓0.5
    ~~int h[10]={0};~~
    for(int n=1; n<11; n++){ ✓1 // Schleife für jedes Intervall
    int z=0;
        for(int i=0; i<l; i++){ // Prüft jede Stelle im Array für das Intervall ✓
            if(Feld[i]>=(n-0.5) && Feld[i]<(n+0.5))
                ~~h[n]++;~~ z++; // Zähler erhöht
                // für das Intervall
                ✓2.5
        }
    cout << n << "_tes Intervall: " << ~~h[n]~~ z << "Zahlen" << endl; ✓1
}
}
```

/5

A3

```
#include <cmath>
class rechteck {

private:
    float b, h;

public:
    rechteck(float hoehe, float breite) {
        h = hoehe; b = breite;
    }
    void rotiere() {
        float tmp; tmp = h; h = b; b = tmp;
    }
    bool operator==(rechteck r) {
        if(r.h == h && r.b == b) { return 1; }
        return 0;
    }
    friend double diagonale(rechteck r) {
        return(sqrt(r.b * r.b + r.h * r.h));
    }
};
```

√0.5

√1

Eigentlich int

Hier auch int

√0.5

√0.5

√1.5

√1.5

√2.5

/8

A4

```
int lexicomp(char Wort1[], char Wort2[], int l1, int l2){
```

~~for(int i=0; i<l1; i++){~~

int i;

~~for(int j=0; j<l2; j++){~~

```
while(i<l1 && i<l2){                    √1.5   // Prüft beide Wörter, ob alle Buchstaben
                                               // gleich sind.
  if(Wort1[i] < Wort2[i]) return -1;
  if(Wort1[i] > Wort2[i]) return 1;     √1.5
  i++;
}
if(l1>l2) return 1;                     // Prüft, wenn beide Wörter gleiche Buch-
                                        // staben haben, welches Wort noch
if(l2>l1) return -1;                    // weiter geht.
return 0;                  √2
}
```

/5

A5

Alpha: 9 ✓

Beta: 19 ✓

Gamma: 17 ✓

Delta: 9 ✓

/4

A6

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

int teilersumme (int z){ // Funktion, die die Summe aller Teiler einer Zahl bildet
    int s=0;
    for(int i=1; i<z; i++){ // z ist kein echter Teiler von z.
        if(z%i==0) s+=i; // Schaut, ob alle Zahlen < z restlos teilbar sind.
    }                    // Wenn ja werden sie zur Summe dazuaddiert.
    return s;
}

int main(){
    for(int z1=2; z1<teilersumme(1000); z1++){ // Schleife beginnt bei 2, da 1 kein echter Teiler hat.
                                               // höchste zu berechnende Summe ist teilersumme(1000)
        for(int z2=2; z2<1000; z++){ // Durchläuft jede Zahl von 2-999
                                     // für jede Zahl von 2-teilersumme(1000)
            if(teilersumme(z1)==z2 && teilersumme(z2)==z1) // Prüft Bedingung für befreundete Zahl.
                cout<<z1<<"und"<<z2<<endl;
        }
    }
}
```

√3

/4