


```

        unsigned int *anzahlWhitespace);

/* Gibt ein Histogramm auf der Standard-Ausgabe aus. */
void gebeHistogrammAus(histogrammT quellHistogramm);

/* Gebe einen Wert mit Beschreibung auf der Standardausgabe aus. */
void gebeWertAus(char const * const beschreibung,
                 unsigned int wert);

```

Implementation der Schnittstelle histogramm.cc

```

#include "histogramm.h"           // Schnittstellendefinitionen
#include <math.h>                 // Fuer log10() (Logarithmus)
#include <iostream>               // Fuer cin und cout
#include <iomanip>                 // Fuer setw() (ostream-Feldbreite einstellen)
#include <ctype.h>                // Fuer isprint() etc. (character types)
#include <string>                 // fuer memset; <cstring> laut ISO

using namespace std;

/* Setzt alle Zaehler im Histogramm auf 0. */
void initialisiereHistogramm(histogrammT dasHistogramm) {
    /* Effizient, haengt aber von mehreren Interna ab! */
    /// Genauer: funktioniert nur, weil alle Zaehler 0 sein sollen und sich
    /// eine int-0 aus vier char-0en zusammensetzt und diese direkt
    /// hintereinander liegen.
    ///
    /// Der (void)-cast verdeutlicht, dass hier ein zurueckgegebener Wert
    /// ignoriert wird.
    (void)memset(dasHistogramm, 0, ANZAHL_ZEICHEN*sizeof(unsigned int));
}

/* Zaehlt die Zeichen der Standardeingabe. */
void erstelleHistogrammAusIStream(histogrammT zielHistogramm) {
    char gelesenesZeichen;
    /// Solange der Eingabestrom noch nicht leer ist, Zeichen lesen und
    /// zaehlen. Die eof()-Abfrage ist aber immer erst gueltig, *nachdem*
    /// ein Zeichen gelesen wurde.
    cin.get(gelesenesZeichen);
    while (!cin.eof()) {
        /// Zur Verdeutlichung wird das Zeichen explizit in einen int
        /// umgewandelt. C++ macht das sonst auch implizit. Der
        /// Inkrement-Operator erhoeht dann den entsprechenden Eintrag im
        /// Feld, weil _zuerst_ mit [] dieser Eintrag bestimmt wird und _erst_
        /// danach_ das ++ ausgewertet wird.
        ++zielHistogramm[(unsigned int)gelesenesZeichen];
        cin.get(gelesenesZeichen);
    }
}

/* Ermittelt ein haeufigstes Zeichen (nicht eindeutig). */
unsigned char haeufigstesZeichen(histogrammT dasHistogramm) {
    /// Prinzip: In einer Schleife ueber das Histogramm-Feld laufen und bei
    /// einem neuen Haeufigkeits-Maximum dieses merken (fuer die restlichen
    /// Vergleiche) und den Index als aktuell ermittelten Rueckgabewert
    /// merken. Der zuletzt gemerkte Wert hat dann die geforderte
    /// Eigenschaft. Wenn das Histogramm leer ist, kann irgendein Index
    /// geliefert werden.
    unsigned int maximaleHaeufigkeit = 0;
    char haeufigstesZeichen = 0;

    /// Die Standard-Schleife ueber die Elemente eines Feldes
    for (unsigned int zeichenIndex = 0;
         zeichenIndex < ANZAHL_ZEICHEN;
         ++zeichenIndex) {
        if (dasHistogramm[zeichenIndex] > maximaleHaeufigkeit) {
            /// Neues Maximum => merken
            haeufigstesZeichen = (char)zeichenIndex;
            maximaleHaeufigkeit = dasHistogramm[zeichenIndex];
        }
    }
    return haeufigstesZeichen;
}

/* Andere Statistiken */
void erstelleStatistiken(histogrammT dasHistogramm,

```

```

        unsigned int *anzahlBuchstaben,
        unsigned int *anzahlWhitespace) {
/// Initialisiere die Rueckgabeparameter.
/// Da Zeiger uebergeben werden, muessen sie mit '*' dereferenziert
/// werden.
///
if (0==anzahlBuchstaben || 0==anzahlWhitespace) {
    cerr << "NULL-Parameter in erstelleStatistiken." << endl;
    return;
}
*anzahlBuchstaben = 0;
*anzahlWhitespace = 0;
for (unsigned int zeichenIndex = 0;
     zeichenIndex < ANZAHL_ZEICHEN;
     ++zeichenIndex) {
    /// Teste den aktuellen Index als Zeichen auf eine Eigenschaft
    /// (Buchstabe oder Whitespace) und addiere gegebenenfalls die
    /// Vorkommen dieses Buchstabens zu den bisher gezaehlten Vorkommen
    /// dieser Zeichenklasse.
    if ( isalpha((unsigned char)zeichenIndex) ) {
        /// addiere zur bisher ermittelten Anzahl Buchstaben (auf die
        /// anzahlBuchstaben zeigt) die Anzahl Vorkommen des aktuellen
        /// Buchstabens.
        *anzahlBuchstaben += dasHistogramm[zeichenIndex];
    }
    if ( isspace((unsigned char)zeichenIndex) ) {
        *anzahlWhitespace += dasHistogramm[zeichenIndex];
    }
}
}

/* Gibt ein Histogramm auf der Standard-Ausgabe aus. */
void gebeHistogrammAus(histogrammT quellHistogramm) {
    /// Die groesste auszugebende Zahl
    unsigned int const MAXIMALE_HAEUFIGKEIT =
        quellHistogramm[(unsigned int)haeufigstesZeichen(quellHistogramm)];
    /// Breite der Zahlenausgabe fuer eine huebschere Tabelle bestimmen
    /// Die reineigentlich vorzeichenlosen Konstanten sind hier ints, weil
    /// setw (wofuer sie benutzt werden) laut ISO C++ ein int-Argument
    /// nimmt.
    int const BREITE_ANZAHL = (int)(log10(
        double(MAXIMALE_HAEUFIGKEIT>0 ? MAXIMALE_HAEUFIGKEIT:1)
    ))+1;
    int const BREITE_CODE = (int)(log10(
        double(ANZAHL_ZEICHEN>0 ? ANZAHL_ZEICHEN:1)
    ))+1;

    /// Das Histogramm als Tabelle Zeile fuer Zeile ausgeben.
    cout << "Die gelesenen Zeichen und ihre Anzahlen:" << endl;
    for(unsigned int zeichenIndex = 0;
        zeichenIndex < ANZAHL_ZEICHEN;
        ++zeichenIndex) {
        /// Eine Zeile ausgeben im Format: "'Zeichen': Anzahl"
        if (0 != quellHistogramm[zeichenIndex]) {
            cout << "'";          /// Soviel ist klar...
            if ( isprint((char)zeichenIndex) ) {
                /// Das Zeichen ist druckbar, also drucke es
                cout << (char)zeichenIndex;
            } else {
                /// Das Zeichen wuerde gedruckt schlecht aussehen (etwa ein
                /// Zeilenumbruch, Tabulator oder BELL), also drucke nur seine
                /// Nummer.
                cout << "<undruckbares Zeichen, Code "
                    << setw(BREITE_CODE) << zeichenIndex << ">";
            }
            /// Mit setw wird die Breite des Feldes definiert, in dem die Zahl
            /// ausgegeben wird. Die Ausgabe-Bibliothek fuellt das Feld von
            /// links mit Leerzeichen auf, so dass die Werte rechtsbueendig
            /// untereinander stehen.
            cout << "' : " << setw(BREITE_ANZAHL)
                << quellHistogramm[zeichenIndex] << endl;
        }
    }
}

```

```

/* Gebe einen Wert mit Beschreibung auf der Standardausgabe aus. */
void gebeWertAus(char const * const beschreibung,
                 unsigned int wert) {
    /// Die Formatierung ist denkbar einfach:
    cout << beschreibung << ": " << wert << endl;
}

/* Hauptprogramm */
int main() {
    histogrammT gezaehlteZeichen;
    unsigned int anzahlWhitespace,
    anzahlBuchstaben;
    initialisiereHistogramm(gezaehlteZeichen);
    erstelleHistogrammAusIStream(gezaehlteZeichen);
    erstelleStatistiken(gezaehlteZeichen, &anzahlBuchstaben, &anzahlWhitespace);
    gebeHistogrammAus(gezaehlteZeichen);
    gebeWertAus("Anzahl Buchstaben", anzahlBuchstaben);
    gebeWertAus("Anzahl Whitespace", anzahlWhitespace);
    return 0;
}

```

9. Aufgabe (10 Punkte)

- (a)
1. 1, 1, 1
 2. 1, 3, 3
 3. 1, 3, 3
- (b)
1. Es sieht so aus, als ob dieses Programm den maximalen Funktionswert von $x*x - 2*x$ berechnet. Das Problem ist, daß `werte` und `max` vom Typ `unsigned int` sind, also keine negativen Werte annehmen können. Für $x=1$ ist der Funktionswert aber -1 und C++ erlaubt es, diesen Wert einer `unsigned int`-Variablen zuzuweisen. Dabei wird einfach das Bitmuster der -1 als `unsigned int` interpretiert. Die Ausgabe des Programms ist also (bei 32-Bit-ints)

Maximum: 4294967295

2. Hier soll ein String `original` nach `kopie` kopiert werden. In der `while`-Schleife werden alle Zeichen des Strings außer dem abschließenden Nullbyte kopiert. Nach dem Kopieren ist also nicht mehr sichergestellt, daß die Kopie mit einem Nullbyte beendet wird, deswegen *kann* das `cout` *manchmal* fehlerhafte Ausgaben erzeugen oder sogar abstürzen. Fehler dieser Art gehören zu den am schwierigsten zu findenden.
3. Es sollte heißen `if (i==max)`. C++ erlaubt die Zuweisung `if (i=max)`, Compiler sollten allerdings eine Warnung ausgeben (z.B. “possibly unintended assignment”). Diese Schleife terminiert sofort, da der Wert der Zuweisung (`i=max`) ungleich 0 ist (der Wert einer Zuweisung ist der zugewiesene Wert).
4. Falsch ist `i:=0` statt `i=0` (Syntaxfehler). Die `while`-Schleife wird nie terminieren, da das `i++` nicht in der Schleife ist (die geschweiften Klammern vor `cout` und nach `i++` wurden vergessen). Die Einrückung von Zeilen hat für den C++-Compiler keine Bedeutung.
5. Hier wurde statt eines logischen Unds (`&&`) ein bitweises (`&`) verwendet. Da `3 & 4` Null ist, würde hier nicht “beide ungleich 0” ausgegeben werden.
6. Hier hätte die Bedingung richtig heißen müssen `if (!(a==0 || b==0))`. Die angegebene Bedingung entspricht `if ( (!a)==0 || b==0 )` und da ja `b==0` ist, wird hier fälschlicherweise “beide ungleich 0” ausgegeben.