Martin Riat

Arithmetik mit grosser Stellenzahl

Ein kleines Programm in C++

Version 1.0
Burriana, Frühling 2003

Einführung

In den frühen Neunzigerjahren schrieb ich ein Programm für *MS-DOS*, das mit Brüchen rechnen konnte und mit 50 Dezimalstellen arbeitete. Das Programm war mit *Turbo Pascal* von *Borland* geschrieben, und dies ist auch heute noch die Programmiersprache, die meinem Temperament am ehesten entspricht.

Als ich im Jahr 2002 eine eigene Webseite erstellte mit der Adresse http://www.riat-serra.org, stellte ich das kleine Programm gratis zur Verfügung. Zu meinem Erstaunen interessierten sich recht viele Leute für das kleine Programm, so dass ich beschloss, diesem Kreis die Technik der Programmierung von mehrstelligen mathematischen Operationen in ihrer einfachsten Art zugänglich zu machen.

Aus didaktischen Gründen ist vermutlich die Pascal-Sprache den meisten anderen weit überlegen, wurde diese ja sogar speziell für die Didaktik der Programmierung geschaffen. Aber heute haben nur noch wenige Leute einen Pascal-Compiler zur Hand. Andererseits war es recht schwierig, anhand des fertigen Programms einen didaktischen Text zu gewinnen.

Ich hatte einen C++-Compiler[1] runtergeladen, den die Firma Borland gratis zur Verfügung stellte. Andererseits bestehen keine wesentlichen Unterschiede zwischen dem Borland-Compiler und dem Compiler von Microsoft[2]. Und nun beschloss ich, das Programm von Grund auf in C++ zu entwickeln.

Da es sich um mein erstes C++ Programm handelt, kann folgendes passieren:

'Echte' C++-Programmierer werden mir vermutlich einen unsauberen Stil vorwerfen.

Da ich die Programmiersprache während der Arbeit erlernt habe, komme ich mit einem Minimum an C++-Sprache aus, so dass es für einen Neuling umso leichter sein sollte, meinen Schritten zu folgen.

---

[1] Borland C++ 5.5.1 for Win32 Copyright (c) 1993, 2000 Borland

[2] Wer den Microsoft Visual C++ Compiler besitzt, kann die beiden #include-Zeilen durch folgende Zeile ersetzen:

```
#include <iostream>
```

Die Absicht hier beschränkt sich auf die Programmierung der Algorithmen, welche für unsere 4 Grundrechnungsarten eingesetzt werden. Der graphischen Darstellung des Programms am Bildschirm wird hier keine weitere Beachtung geschenkt, nicht zuletzt, weil diese stark systemabhängig ist. Zudem kann diese auch stark von der Stellenzahl abhängen, an die das Programm schlussendlich angepasst wird. Was die Stellenzahl anbelangt, ist zu sagen, dass diese nur durch den Compiler und die Hardware limitiert wird.

Genau so, wie ein Programm allmählich heranwächst, habe ich die einzelnen Etappen in einzelne Versionen abgespeichert, angefangen mit 00.CPP. Entsprechend habe ich die Kapitel dieses Essays durchnumeriert. Wie gesagt, geht es mir weder um die C++-Sprache, noch um die Erschaffung eines schönen Programms, sondern ausschliesslich um die Umsetzung in eine moderne Programmiersprache der arithmetischen Algorithmen, die wir üblicherweise auf unsere vier Grundrechnungsarten anwenden. Und wir werden sehen, dass diese Algorithmen, die jeder Zehnjährige intuitiv beherrschen sollte, doch recht verzwickt sind.

Wer je den Mechanismus einer mechanischen Rechenmaschine beobachtet hat, wird auch hier eine andere Umsetzung derselben Algorithmen entdecken.

Kritik ist immer erwünscht. Vor allem aber bin ich für die Hinweise auf Fehler dankbar. Diese können an riat@pobox.com gerichtet werden.

Der Autor lehnt jede Verantwortung ab für Schäden, die direkt oder indirekt durch die Anwendung seiner Software entstehen könnten.

Sollte jemand den Programmtext für ein eigenes Programm verwenden, bitte ich um Erwähnung der Quelle.

00

Die allererste Version unseres Quellentextes dient vor allem dazu, grundlegende Definitionen festzuhalten. Das Symbol // dient in C++ dazu, Kommentare in den Text einzufügen. Diese erstrecken sich jeweils bis zum Zeilenende. Ich habe die Kommentare weitgehend in englischer Sprache abgefasst, um von möglichst vielen Lesern verstanden zu werden.

Um den Überblick besser behalten zu können, nehme ich mir vor, hier jeder Version eine kurze Rubrik mit dem Titel "What's new?" voranzustellen, in der die wichtigsten Veränderungen kurz erwähnt werden. Trotzdem werden kleinere Änderung innerhalb einzelner Funktionen kommentarlos vorgenommen.

Nach den Include-Dateien, die von der Compiler-Version abhängen, definiere ich ein paar Typen und Konstanten, die ich allerdings im Laufe der Programmentwicklung ab und zu anpassen werde.

Da das Programm mit einer variablen Stellenzahl arbeiten können soll definiere ich den neuen Zahlentyp `numbertype` vorerst als `unsigned short int.` Bei sehr grosser Stellenzahl kann diese Definition entsprechend angepasst werden.

Das Programm soll vorerst mit Dezimalzahlen arbeiten. Für den Fall, dass es mal an eine andere Basis angepasst werden sollte, definiere ich hier mal die Konstante `basis` als 10. Ebenso definiere ich den Zahlentyp small, der gross genug sein muss, um die Basis zu enthalten, als `unsigned short int.`

Dann definieren wir eine Konstante des Typs `numbertype` und ordnen ihr den Wert 50 zu, welcher der Rechnung mit 50 Stellen entspricht. Bei der Bearbeitung des Programms werden wir diesen Wert auf einen für das Debuggen des Programms praktischeren Wert heruntersetzen.

Jetzt erfolgt die vermutlich wichtigste Definition unseres Programms, die der Klasse `big_number.` Die Ziffern werden im Array `cipher` gespeichert. Der wert `maxpos` legt die höchste signifikative Stelle fest. Ob wir diesen Wert später benötigen werden, wissen wir noch nicht. Um festzuhalten, ob wir es mit einer positiven oder einer negativen Zahl zu tun haben, schaffen wir die logische Variable `positive`.

Dann folgt die Definition der Methoden, die es erlauben werden mit den `big_number` zu arbeiten. Vorläufig ist nur eine provisorische Funktion `write` angegeben, die in diesem Stadium des Programms noch gar nichts vernünftiges macht, sondern nur "Test" an den Bildschirm schreibt. Allmählich wird diese Funktion so ausgebaut werden, dass sie die Zahlen der Klasse `big_number` in verständlicher Weise am Bildschirm darstellt.

Als Kommentar zur ersten Version sind die Path-Instruktionen angegeben, unter denen das Programm mit dem Borland-Compiler Kompiliert werden kann. Es empfiehlt sich, diese Zeile in eine Batch-Datei zu kopieren, etwa mit dem Namen WEG.BAT.

Der Zweite Kommentar zitiert die Tastatureingabe zum Kompilieren und Linken des Programms. Auch diese Zeile (hier auf zwei Zeilen dargestellt) wird am besten in eine Batch-Datei mit dem Namen COMP.BAT kopiert, wobei "00.CPP" durch "%1.CPP" ersetzt wird, so dass der Vorgang durch den folgenden Befehl aufgerufen werden kann:

C:>COMP 00

Ich speichere hier den Quelltext ab und mache eine Kopie davon, mit dem Namen 01.CPP.

01

In dieser zweiten Fassung des Programms sollen mehrere Probleme gelöst werden: Vor allem sollen geeignete Methoden festgelegt werden, um eine Zahl des Typs `big_number` über die Tastatur einzugeben und auf dem Bildschirm wiederzugeben.

Vor allem die Eingabe über die Tastatur, die von der Funktion `get_number` kontrolliert wird, ist eine recht komplizierte Angelegenheit, auf die ich hier nicht näher eingehen werde. Es geht mir hier einfach darum, über eine Funktion zu verfügen, welche es erlaubt, die anderen Funktionen des Programms eingehend zu prüfen. Sollten später die hier vorgestellten Funktionen zur Schaffung eines kommerziellen Programms dienen, so müssten gerade die beiden Funktionen `get_number` und `write` vollständig umgeschrieben und vermutlich auch an das betreffende Betriebssystem angepasst werden. So empfehle ich dem Leser, die beiden Funktionen wie eine schwarze Kiste

(black box[3]) zu behandeln, sie also zu benutzen, ohne sich über ihren Aufbau Gedanken zu machen.

Die Funktion `put_to_zero` macht eine beliebige Zahl des Typs `big_number` zu 0: es werden einfach alle Ziffern zu 0 gesetzt und ebenso die Stellenzahl `maxpos`. Die logische Variable `positive` setzen wir definitionsgemäss auf `true` (wir wollen also die Null als positive Zahl auffassen.

In der Funktion `main` (Hauptfunktion) schaffen wir vorerst mal eine Variable des Typs `big_number` mit dem Namen `test`, indem wir schreiben:

```
big_number test;
```

Dann ordnen wir dieser Variablen einen Wert zu, indem wir diesen unter Benutzung der Funktion `get_number` über die Tastatur eingeben. Dies entspricht der Zeile:

```
test.get_number ();
```

Den eingegebenen Wert geben wir dann auf dem Bildschirm aus:

```
test.write ();
```

Schliesslich wird `test` zu 0 umgeformt und wieder auf dem Bildschirm ausgegeben:

```
test.put_to_zero ();
test.write ();
```

02

Im Kapitel 02 schreiben wir eine weitere Funktion, mit dem Namen `put_to_one`, die eine Zahl des Typs `big_number` zu 1 umformt. Die Funktion benutzt die Funktion `put_to_zero`.

Dann entstehen zwei logische Funktionen, welche zwei Zahlen miteinander vergleichen. Die Funktion `equal` gibt `true` zurück, falls die beiden Zahlen gleich gross sind. Entsprechend gibt die Funktion `bigger true` zurück, wenn die erste Zahl grösser als die zweite Zahl ist. Die vergleichenden Funktionen arbeiten mit dem absoluten Wert der Zahlen, gehen also nicht auf das Vorzeichen ein.

---

[3] In jedem sauber gestalteten Programm sollten die einzelnen Funktionen so gestaltet sein, dass sie ohne Kenntnis des inneren Aufbaus sofort eingesetzt werden können. Es muss lediglich eine Beschreibung vorliegen, aus der man ablesen kann, was eingegeben werden muss, um welche Ausgabe zu erhalten.

Nun spielen wir mit dem Gedanken, im nächsten Kapitel die erste eigentliche Operation zu definieren, die Addition zweier Zahlen.

03

Bevor wir die Addition von zwei Zahlen programmieren, müssen wir folgendes bedenken: wenn wir beispielsweise mit dreistelligen Zahlen arbeiten kann die Summe von Zwei Zahlen den Rahmen unserer Zahlenmenge sprengen. In diesem Fall erhalten wir kein verlässliches Resultat mehr. So können wir etwa 333 und 821 nicht mehr im Rahmen von dreistelligen Zahlen addieren. Wenn wir zur grösstmöglichen dreistelligen Zahl 1 addieren, erhalten wir Null. Dasselbe geschieht beim Kilometerzähler eines Autos, so dass wir nicht mit Sicherheit wissen können, wie viele Zyklen der Zähler zurückgelegt hat.

Das ist der Grund, warum ich an dieser Stelle eine neue Variable einführe. Die entsprechenden Funktionen müssen entsprechend angepasst werden. Die neue Variable ist eine logische Variable und heisst `overflow`. `overflow` wird am Anfang des Programms auf `false` gesetzt. Sobald `overflow true` ist, werden alle darauffolgenden Operationen abgebrochen. `overflow` ist eine globale Variable.

Die Addition zweier Zahlen wird hier von rechts nach links ausgeführt, wobei die zu behaltende Einheit `keep` auf die nächste Kolonne übertragen wird, genau gleich wie dies im üblichen Algorithmus geschieht, welcher in den ersten Jahren der Grundschule gelehrt wird. Ist nach dem Ausschöpfen der grösstmöglichen Stellenzahl der Wert `keep` nicht gleich Null, wird die variable `overflow` aktiviert (auf `true` gesetzt).

Ich führe hier auch einen neuen Zahlentyp ein, `medium`, der gross genug sein soll, um das Quadrat der Basis zu enthalten. So muss etwa im Fall der Dezimalarithmetik der Typ `small` gross genug sein, um die Zahlen 0 bis 10 zu enthalten, während `medium` die Zahlen 0 bis 100 enthalten muss.

Die Funktion `shift_left` führt eine Multiplikation einer Zahl des Typs `big_number` mit seiner Basis durch. In den ersten Jahrgängen der Grundschulen lernen die Kinder, dass beim Multiplizieren mit 10 eine Null hinter die zu multiplizierende Zahl kommt. Was viele Menschen nicht wissen, ist, dass diese Regel auch auf beliebige Zahlensysteme

verallgemeinert werden kann: Wird eine Zahl mit der Basis des Zahlensystems multipliziert, wird eine 0 an die zu multiplizierende Zahl angefügt. Die Multiplikation mit der Basis ist für die Programmierung des Algorithmus der Multiplikation wichtig.

Innerhalb dieser Funktion erscheinen die Zeilen:

```
keep = s / basis;
cipher [i] = s % basis;
```

Dazu ist zu bemerken, dass die Division mit dem Symbol / ein ganzzahliges Resultat liefert. Das Symbol % dient der Berechnung des Rests der Division.

04

Die Version 04 unseres Programms ist der Implementierung der Multiplikation gewidmet. Ich habe hier das Schema des in der Schule üblichen Algorithmus nachempfunden und jeweils zum provisorischen Resultat das Produkt einer einzelnen Stelle eines Faktors mit dem "verschobenen" anderen Faktor addiert. Um unnötige Kompliziertheit zu vermeiden habe ich die Stellenzahl der grösstmöglichen Stellenzahl gleichgesetzt. Dies geschieht durch die Zeile:

```
partial.maxpos = positions;
```

Am Ende muss der wirkliche Wert für maxpos bestimmt werden:

```
for (i = positions; i > 0; i--)
        {
          if (found == false)
               if (cipher [i] != 0)
            {
              maxpos = i;
              found = true;
            }
        }
      if (found == false) maxpos = 0;
```

Dann muss der Funktion der Wert von `result` zugeordnet werden:

```
// to copy the ciphers
       for (i = 0; i <= positions; i++)
         {
           cipher [i] = result.cipher [i];
         }
```

Um diese Aufgabe zu übernehmen wird später (ab Version 06) die Methode `big_assign` eingesetzt werden.

Bei der Programmierung der Multiplikation tritt die Parallele zwischen dem üblichen Algorithmus, unserer Umsetzung in C++-Sprache und dem Aufbau der mechanischen (und elektromechanischen) Rechenmaschinen, die in den Sechzigerjahren durch die elektronischen Rechner abgelöst wurden, besonders deutlich hervor.

05

Im Kapitel 05 werden die inversen Operationen zu der Addition und Multiplikation behandelt. Die Subtraktion hat eine ganz ähnliche Struktur wie die Addition des Kapitels 04. Bei der Division wird wieder der klassische Algorithmus nachempfunden, bei dem der Dividend bei feststehendem Divisor stellenweise erweitert wird.

C++ erlaubt es uns hier, den Rest als Referenz auszudrücken, so dass die entsprechende übertragene Variable verändert wird. Das geschieht durch voranstellen des Symbols & an den Variablennamen in der Deklaration der Methode.

06

Der erste Schritt in 06.CPP besteht darin, die Funktion `shift_left` als `private` zu definieren. Ich mache dies, weil diese Funktion ausschliesslich von der Funktion `div` aufgerufen wird. Künftige Programme werden nicht direkt auf diese Funktion zurückgreifen müssen.

Dann beginne ich, den Grössten Gemeinsamen Teiler, GgT, hier Greatest Common Divisor, `gcd,` zu programmieren. Ich mache das nach dem Euklides zugeschriebenen Algorithmus der wiederholten Division mit Rest, gemäss folgendem Beispiel:

88	=	1	*	**76**	+	12
76	=	6	*	12	+	**4**
12	=	3	*	4	+	0

Der letzte nicht verschwindende Rest, 4 ist der ggT der beiden Zahlen 88 und 76.

Mit den bescheidenen Kenntnissen, die ich von C++ habe, ist es mir nicht ohne weiteres möglich, am Ende der Funktion `gcd,` den Wert von `a` an die Funktion zu übergeben. Wie ich das schon in der Funktion `mult` machte, könnte ich die entsprechenden Werte einzeln kopieren. Aber ich ziehe es vor, dafür die Hilfsfunktion `big_assign` zu schreiben, die auch für spätere Funktionen problemlos eingesetzt werden könnte. Ich verzichte vorläufig darauf, die Hilfsfunktion in `mult` einzusetzen.

Die bisherige Funktion `write` enthielt ein paar Kontrollelemente; unter anderem wurde das Vorzeichen ausgedruckt, die Stellenzahl wurde angegeben (0 für einstellige, k für k+1-stellige Zahlen), alle vorangestellten Nullen wurden ausgedruckt. Das ist jetzt nicht mehr nötig, da diese Kontrollelemente vor allem der Überprüfung der arithmetischen Grundoperationen diente. Trotzdem habe ich vorläufig eine Kopie der Funktion `write` unter dem Namen `write_0` weitergeführt. Die neue Funktion `write` beschränkt sich auf die

signifikativen Stellen, druckt aber an Stelle der vorangestellten Nullen jeweils eine Leerstelle.

Auch die Funktion, die den ggT bestimmt, habe ich abgewandelt, um den euklidischen Algorithmus zeilenweise am Bildschirm auszudrucken; die neue Funktion heisst `Euclid.`

07

In der Stufe 07 wurde vorerst eine Funktion geschrieben, die das kgV von zwei Zahlen berechnet. Diese Funktion `lcm` (von *Lowest Common Multiple*) beruht auf der Tatsache, dass das Produkt von zwei Zahlen gleich ist dem Produkt aus kgV und ggT. Der ggT lässt sich also als Quotient des Produkts der beiden Zahlen und des ggT berechnen. Da der ggT beide Zahlen teilt, dividiere ich die erste Zahl durch den ggT, bevor ich mit der zweiten Zahl multipliziere. Damit kann in vielen Fällen ein `Overflow` vermieden werden.

Unter dem Titel *Other Functions* beginne ich ein Menüsystem aufzubauen, so dass später unter verschiedenen Optionen ausgewählt werden kann. Da ich keine kompatible Art finde, ein Zeichen direkt von der Tastatur abzulesen, versuche ich es vorerst mit der Instruktion

```
std::cin >> a_number;
```

Ich werde damit Probleme haben und sie in 08.CPP durch folgende Instruktion ablösen, die offenbar im Gegensatz zur ersten Version den Buffer leert:

```
cin.getline (s, 25);
```

08

Hier wird der erste Menüpunkt weiter ausgebaut. Ferner werden ein paar Hilfsfunktionen geschrieben. Da ich keine Bildschirmspezifischen Instruktionen gefunden habe, habe ich eine

Funktion `cls` (von *Clear Screen*) geschrieben, die nichts anderes macht als eine Anzahl Leerzeilen auszugeben.

Die Funktion `wait` gibt eine Meldung am Bildschirm aus wartet darauf, dass `<Intro>` gedrückt wird. Die Funktion `gotoline` lässt den Cursor auf die nächste Zeile springen. `dash` druckt eine der maximalen Stellenzahl `positions` entsprechend lange Reihe von Bindestrichen am Bildschirm aus. Dann wird eine Marke ausgedruckt, die das Ende der Stellenkapazität anzeigt. Schliesslich wird der Cursor durch wiederholtes Ausdrucken des Symbols `\b` wieder an den Zeilenanfang zurückgestellt.

09

Hier wird das Menüsystem ausgebaut und die wesentlichen Optionen des Bruchrechnens nach den herkömmlichen Regeln programmiert. Zuletzt setze ich die Stellenzahl auf 50, indem ich setze:

```
const numbertype positions = 50;
```

Die einzelnen Bruchoperationen sind weitgehend selbsterklärend, so dass hier nicht mehr darauf eingegangen wird.

Es ist hier eine recht spartanische Version eines Bruchrechners entstanden, dessen Elemente aber die Grundlage bilden könnten für die Programmierung eines ausgefeilten Programms mit Windows-Design, Mausbedienung, Einfuhr und Ausfuhr von Daten mittels Dateien, etc. Warum auch nicht ein Kalkulationsblatt an die Rechnung mit gemeinen Brüchen adaptieren?

Eine weitere interessante Möglichkeit ist die Annäherung von irrationalen Zahlen, wie etwa von Pi, anhand von Zahlenfolgen.

10

Der Schritt, der in der Zehnten Version durchgeführt wurde, ist beinahe trivial. Durch setzen der Basis auf 2 und eine kleine Abänderung in der Funktion `get_number` bringe ich das Programm dazu, mit binären Zahlen zu rechnen. Und wenn wir mit einer Basis rechnen

möchten, die grösser als 10 ist? Auch das ist leicht zu erreichen. Die grösste Änderung müsste in `get_number` vorgenommen werden.

Möchten wir etwa mit Hexadezimalzahlen rechnen, müssten wir die Basis auf 16 setzen. In `get_number` müssten wir neben den Ziffern von 0 bis 9 auch die Buchstaben A, B, C, D, E und F akzeptieren. Diese müssten dann als 10, 11, 12, 13, 14 und 15 interpretiert werden. Die Ausgabe am Bildschirm mittels `write` müsste entsprechend angepasst werden.

09.CPP

Schliesslich sei hier der Programmtext der Version 09 unseres kleinen Programms wiedergegeben. Die Dateien 00.CPP bis 10.CPP, eine Kopie dieses Textes in PDF-Form, sowie die mit dem oben erwähnten Borland Compiler für das MS-DOS von Windows 98 kompilierten Versionen 09.EXE und 10.EXE befinden sich komprimiert in der Datei BRUCH-01.ZIP. Die Daten dürfen nur in Form der originalen komprimierten Datei weitergegeben werden.

```
// ________________________________________
// 09.cpp
// ----------------------------------------

// .... What's new? .......................
// More menu options: The operations with fractions
// Changed Euclid
// ........................................

#include <stdio.h>           // for Borland Compiler
#include <iostream.h>        // for Borland Compiler

// #include <iostream>       // for MS Visual C++

#define numbertype unsigned short int // Adapt to the number of positions
#define basis 10                      // Adapt to the numbering system
                                      // (decimal, hex, etc...)
#define small unsigned short int      // small must be great enough to
                                      // contain the basis
#define medium unsigned short int     // medium must be great enough to
                                      // contain the square of the basis
#define sufficient 200                // chane length for input

const numbertype positions = 50;      // a constant
bool overflow = false;                // a global variable


// - - - Auxiliary functions:  - - - - - - - - - - - - - - - - - - - - - - -


numbertype maxim (numbertype a, numbertype b);
void print (int number);
void wait ();
void cls ();
void gotoline ();
void dash ();
void space ();
```

```
numbertype maxim (numbertype a, numbertype b)
      // ---------------------------------
      // Determinates the maximum of two numbers
      // ---------------------------------
{
   numbertype max;

   if (a > b) max = a; else max = b;
   return (max);
}


void print (int number)
      // ---------------------------------
      // Number output on screen (for test purposes)
      // ---------------------------------
{
   std::cout  << number << " ";
}


void wait ()
      // ---------------------------------
      // Waits for <intro> to be pressed
      // ---------------------------------
{
    char s [sufficient];
    std::cout << "Press Intro";
    std::cin.getline (s, sufficient);
}


void cls ()
      // ---------------------------------
      // clear screen
      // ---------------------------------
{
    unsigned short int line;
    for (line = 0; line < 40; line++) std::cout << "\n";
}


void gotoline ()
      // ---------------------------------
      // put the cursor on the next line
      // ---------------------------------
{
    std::cout << "\n";
}


void dash ()
      // ---------------------------------
      // Makes dashed line of length = positions + 1
      // Puts the cursor to the beginning of the line
      // ---------------------------------
{
    unsigned short int i;
    for (i = 0; i <= positions; i++) std::cout << '_';
    std::cout << "þþþ";
    for (i = 0; i <= positions+3; i++) std::cout << "\b";
}
```

```
void space ()
      // ---------------------------------
      // Makes a space of the length of positions - 8
      // ---------------------------------
{
    unsigned short int i;
    for (i = 0; i <= positions - 8; i++) std::cout << ' ';
}

// - - - Class big_number: - - - - - - - - - - - - - - - - - - - - - - - - - - -


class big_number
      // ---------------------------------
      // Definition of class big_number
      // ---------------------------------
{
    small cipher [positions+1];
    numbertype  maxpos;
    bool positive;
  public:
    void get_number ();
    void write_0 ();
    void write ();
    void put_to_zero ();
    void put_to_one ();
    bool equal (big_number a, big_number b);
    bool bigger (big_number a, big_number b);
    void sum (big_number a, big_number b);
    void subt (big_number a, big_number b);
    void mult (big_number a, big_number b);
    void div (big_number a, big_number b, big_number &rest);
    void big_assign (big_number a);
    void gcd (big_number a, big_number b);
    void euclid (big_number a, big_number b);
    void lcm (big_number a, big_number b);
  private:
    void shift_left ();
};


// - - - big_number functions: - - - - - - - - - - - - - - - - - - - - - - - - -


void big_number::get_number ()
      // ---------------------------------
      // Number input from keyboard
      // ---------------------------------
{
   char s [sufficient];
   bool ok;
   bool final_reached;
   bool zero;               // the chain represents 0
   numbertype i, j, max;
   numbertype length;

   ok = false;

   while (ok == false)
       {
         final_reached = false;
         ok = true;
         zero = false;
         max = positions+1;

         // Initiating the string s to "BBBBBB...":
        for (j=0; j <= positions+3; j++) s [j] = 66;
        s [positions+4] = 0;
```

```
        std::cin.getline (s, sufficient);  // little change

         if ((s [0] == 48) && (s [2] == 66))   // If the chain represents 0
        {
           ok = true;
           zero = true;
         }
         else    // Usual case
         {
           for (i=0; i<=positions; i++)
             {
             if (s [i] == 0) final_reached = true;
             if (s [i] == 0) max = i;
             if (final_reached == false) if (s [i] < 48) ok = false;
             if (final_reached == false) if (s [i] > 57) ok = false;
             }
           if (s [positions+1] == 0) final_reached = true;
           if (final_reached == false) ok = false;
           if (s [0] == 0) ok = false;
           if (s [0] == 48) ok = false; // not to begin number with 0

           for (i = 0; i < positions + 2; i++)
             if (s [i] == 0) length = i;
           if (length > positions + 1) ok = false;
         }
       if (ok == false) std::cout << "Error" << "\n";
     }

   // initialize:

   for (i = 0; i <= positions; i++) cipher [i] = 0;

   if (zero == false)
   {
      for (i = 0; i < max; i++) cipher [i] = s [max-i-1] - 48;

      // Determine maxpos:
         ok = false;
         i = positions+1;
         while (ok == false)
           {
             i--;
             if (cipher [i] != 0) ok = true;
           }
         maxpos = i;
   }
   else
   {
      maxpos = 0;
   }

   // The number entered is considered positive:
      positive = true;
}


void big_number::write_0 ()
      // ---------------------------------
      // Public accessor function
      // writes the big_number to screen
      // ---------------------------------
{
    if (overflow == false)
    {
      numbertype poscounter;
      if (positive == true) std::cout <<  "+"; else std::cout << "-";
      for (poscounter = positions; poscounter > 0; poscounter--)
```

```
        std::cout  << cipher [poscounter];
        std::cout  << cipher [0];  // write the last one

      std::cout <<  "   ";            // Control to be eliminated later
      std::cout << maxpos; //    << "\n\n";  // -------

      std::cout << "\n";
    }
    else  std::cout  << "Overflow Error!";
}


void big_number::write ()
      // ---------------------------------
      // Public accessor function
      // writes the big_number to screen
      // ---------------------------------
{
    if (overflow == false)
    {
      numbertype poscounter;
      bool reached = false;

      // if (positive == true) std::cout <<  "+"; else std::cout << "-";
      for (poscounter = positions; poscounter > 0; poscounter--)
          {
            if (cipher [poscounter] != 0) reached = true;
            if (reached == true)
              std::cout  << cipher [poscounter];
            else
              std::cout  << " ";
          }
        std::cout  << cipher [0];  // write the last one
    }
    else  std::cout  << "Overflow Error!";
}


void big_number::put_to_zero ()
      // ---------------------------------
      // Public accessor function
      // puts the big_number to 0
      // ---------------------------------
{
    numbertype poscounter;

    for (poscounter = positions; poscounter > 0; poscounter--)
      cipher [poscounter] = 0;
      cipher [poscounter] = 0;  // the last one (position 0)
    maxpos = 0;
    positive = true;
}


void big_number::put_to_one ()
      // ---------------------------------
      // Public accessor function
      // puts the big_number to 1
      // ---------------------------------
{
  put_to_zero ();
  cipher [0] = 1;
}
```

```
bool big_number::equal (big_number a, big_number b)
      // ---------------------------------
      // Gives true if the numbers are equal
      // ---------------------------------
{
  numbertype i;
  bool identic;
  identic = true;

  if (a.maxpos != b.maxpos) identic = false;
  for (i=0; i<=positions; i++)
    {
      if (a.cipher [i] != b.cipher [i]) identic = false;
    }

  return (identic);
}


bool big_number::bigger (big_number a, big_number b)
      // ---------------------------------
      // Gives true if    a > b
      // ---------------------------------
{
  numbertype k;
  bool is_bigger;
  bool enough;

  k = maxim (a.maxpos, b.maxpos);
  enough = false;
  is_bigger = false;

  while (enough == false)
    {
      if (a.cipher [k] < b.cipher [k]) enough = true;
      if (a.cipher [k] > b.cipher [k])
        {
          is_bigger = true;
          enough = true;
        }
      if (k == 0) enough = true;
      k--;
    }
  return (is_bigger);
}


void big_number::big_assign (big_number a)
      // ---------------------------------
      // Assign a big_number to another,
      // as used in gcd
      // ---------------------------------
{
   numbertype i;

   // to copy the ciphers
   for (i = 0; i <= positions; i++)
     {
       cipher [i] = a.cipher [i];
     }

   maxpos = a.maxpos;
   positive = a.positive;
}
```

```
void big_number::sum (big_number a, big_number b)
      // ---------------------------------
      // The sum of a and b
      // ---------------------------------
{
   numbertype i, n;
   medium s, keep;

   if (overflow != true)
   {
     put_to_zero ();                                  // sum = 0
     n = maxim (a.maxpos, b.maxpos);
     keep = 0;
     for (i = 0; i <= n; i++)
       {
         s = a.cipher [i] + b.cipher [i] + keep;
         keep = s / basis;          // in Pascal it would be: s DIV basis
         cipher [i] = s % basis;    // in Pascal it would be: s MOD basis
       }

     if (keep != 0) if (n == positions) overflow = true;
     if (keep != 0) if (n < positions)
        {
          n++;
          cipher [n] = keep;
        }
     maxpos = n;
   }
}


void big_number::subt (big_number a, big_number b)
      // ---------------------------------
      // The difference of a and b
      // Only works if a >= b
      // ---------------------------------
{
   numbertype i, last;
   medium d, keep;
   bool found = false;

   if (overflow != true)
   {
     put_to_zero ();                                  // subt = 0
     last = maxim (a.maxpos, b.maxpos);
     keep = 0;
     for (i = 0; i <= last; i++)
       {
         if (a.cipher [i] >= (b.cipher [i] + keep) )
           {
             d = a.cipher [i] - b.cipher [i] - keep;
             keep = 0;
           }
         else
           {
             d = basis + a.cipher [i] - b.cipher [i] - keep;
             keep = 1;
           }
         cipher [i] = d;
       }
     if (keep != 0) overflow = true;

     // to determinate maxpos:
     for (i = positions; i > 0; i--)
         {
           if (found == false) if (cipher [i] != 0)
             {
               maxpos = i;
```

```
              found = true;
            }
        }
    if (found == false) maxpos = 0;

  }
}



void big_number::shift_left ()
     // ---------------------------------
     // Multiplication with the basis
     // ---------------------------------
{
  numbertype i;
  bool is_zero;

  if (overflow != true)
  {
    if (maxpos == positions)
      overflow = true;
    else
    {
      if ((maxpos == 0) && (cipher [0] == 0))        // && is AND
             is_zero = true;
      else   is_zero = false;

      for (i = maxpos + 1; i > 0; i--) cipher [i] = cipher [i - 1];

      cipher [0] = 0;
      if (is_zero == false) maxpos++;
    }
  }
}



void big_number::mult (big_number a, big_number b)
     // ---------------------------------
     // The product of a and b
     // ---------------------------------
{
  numbertype i, j, k;
  medium keep, p;
  bool exit = false;
  bool found = false;

  big_number partial, result, auxiliar;

  if (overflow != true)
  {
    put_to_zero ();                                  // sum = 0
    partial.put_to_zero ();
    result.put_to_zero ();
    auxiliar.put_to_zero ();

    // If one of the factors is zero, the product is zero:
    if (equal (partial, a) == true) exit = true;
    if (equal (partial, b) == true) exit = true;

    // If both factors different from zero:
    if (exit == false)
    {
        for (j = 0; j <= b.maxpos; j++)
        {
          partial.put_to_zero ();
          partial.maxpos = positions;  // Even if the first positions may be 0
```

```
            keep = 0;
            for (i = 0; i <= positions; i++)
              {
                k = i + j;
                p = a.cipher [i] * b.cipher [j] + keep;
                if (k <= positions) partial.cipher [k] = p % basis;
                keep = p / basis;
                if (k > positions) if (p > 0) overflow = true;
              }
            if (keep != 0) if (k <= positions)
              {
                partial.cipher [k+1] = keep;
              }

            auxiliar = result;

            result.sum (auxiliar, partial);
          }                                   //  for (j = 0; j <= positions; j++)

        // to copy the ciphers
        for (i = 0; i <= positions; i++)
          {
            cipher [i] = result.cipher [i];
          }

        // to determinate maxpos:
        for (i = positions; i > 0; i--)
          {
            if (found == false) if (cipher [i] != 0)
              {
                maxpos = i;
                found = true;
              }
          }
        if (found == false) maxpos = 0;
      }                                                    // if (exit == false)
    }                                                    // if (overflow != true)
}



void big_number::div (big_number a, big_number b, big_number &rest)
      // ---------------------------------
      // The quotient of a and b, a / b
      //   rest is passed by reference and so
      //   its global value can be changed
      // ---------------------------------
{
   numbertype i, pos;
   big_number aux, z;
   bool enough = false, exit = false, found = false;
   small times;


   if (overflow != true)
   {
     z.put_to_zero ();     // to compare with zero

     if (b.equal (b, z) == true)
        {
          overflow = true;
          exit = true;
        }

     if (b.bigger (b, a) == true)
        {
           put_to_zero ();  // puts quotient to zero
           rest = a;
```

```
            exit = true;
         }

      if (exit == false)
         {
           put_to_zero ();
           aux.put_to_zero ();
           rest.put_to_zero ();

           pos = positions;
           aux.cipher [0] = a.cipher [pos];

           while (enough == false)
             {
               times = 0;
               while (   (aux.equal (aux, b) == true)
                      || (aux.bigger (aux, b) == true))       // || means OR
                 {
                   times++;
                   aux.subt (aux, b);
                 }
               cipher [pos] = times;
               if (pos > 0) pos--;
                            else enough = true;
               if (enough == false)
                 {
                   aux.shift_left ();
                   aux.cipher [0] = a.cipher [pos];
                 }
             }                                          // while (enough == false)
           rest = aux;

           // to determinate maxpos of quotient:
           for (i = positions; i > 0; i--)
             {
               if (found == false) if (cipher [i] != 0)
                 {
                   maxpos = i;
                   found = true;
                 }
             }
           if (found == false) maxpos = 0;

           // to determinate maxpos of rest:
           found = false;
           for (i = positions; i > 0; i--)
             {
               if (found == false) if (rest.cipher [i] != 0)
                 {
                   rest.maxpos = i;
                   found = true;
                 }
             }
           if (found == false) rest.maxpos = 0;

         }                                                 // if (exit == false)
   }                                                      // if (overflow != true)
}


void big_number::gcd (big_number a, big_number b)
      // ---------------------------------
      // The Greatest Common Divisor of a and b
      // ---------------------------------
{
   big_number zero, quotient, rest;

   zero.put_to_zero ();
```

```
   if (equal (a, zero) == true) overflow = true;
   if (equal (b, zero) == true) overflow = true;

   if (overflow != true)
     {
       rest.put_to_one (); // A arbitrary value different from 0
       while (equal (rest, zero) != true)
         {
           quotient.div (a, b, rest);
           a = b;
           b = rest;
         }
       big_assign (a);  // Result of the method is a
     }
}



void big_number::euclid (big_number a, big_number b)
      // ----------------------------------
      // The Greatest Common Divisor of a and b
      // and the representation of the Euclidic algorithm
      // ----------------------------------
{
   big_number zero, quotient, rest;
//   unsigned short int lines;

   zero.put_to_zero ();
   if (equal (a, zero) == true) overflow = true;
   if (equal (b, zero) == true) overflow = true;

   if (overflow != true)
     {
       rest.put_to_one (); // A arbitrary value different from 0

       gotoline ();

//       lines = 1;

       while (equal (rest, zero) != true)
         {

//           if ((lines % 1) == 0)
//             {
//                lines = 1;
//                wait ();
//             }
//           lines++;

           quotient.div (a, b, rest);

               a.write ();
               gotoline ();
               space ();
               std::cout  << "=";
               gotoline ();

               quotient.write ();
               gotoline ();
               space ();
               std::cout  << "*";
               gotoline ();

               b.write ();
               gotoline ();
               space ();
               std::cout  << "+";
               gotoline ();
```

```
                rest.write ();
                gotoline ();
                gotoline ();
                gotoline ();

            a = b;
            b = rest;
            wait ();
          }                                                    // while
        big_assign (a);  // Result of the method is a
      }
}


void big_number::lcm (big_number a, big_number b)
      // ---------------------------------
      // The Lowest Common Multiple of a and b
      // (a * b) / (gcd of a and b)
      // ---------------------------------
{
   big_number zero, max, q, r;

   zero.put_to_zero ();
   if (equal (a, zero) == true) overflow = true;
   if (equal (b, zero) == true) overflow = true;

   if (overflow != true)
     {
       max.gcd (a, b);
       q.div (a, max, r);  // max divides a, so now we only have to
                           // multiply q and b
       mult (q, b);
     }
}



// - - - Other functions - - - - - - - - - - - - - - - - - - - - - - - - - -

void menu ();
void menu_01 ();
void menu_02 ();
void menu_03 ();
void menu_04 ();   // Addition
void menu_05 ();   // Subtraktion
void menu_06 ();   // Multiplikation
void menu_07 ();   // Division


void menu ()
      // ---------------------------------
      // Menu text
      // ---------------------------------
{
  std::cout << "Bruchrechner\n";
  std::cout << "------------\n";
  std::cout << "\n";
  std::cout << "1) Ggt und kgV von zwei Zahlen\n";
  std::cout << "2) Euklidischer algorithmus\n";
  std::cout << "3) Kürzen eines Bruchs\n";
  std::cout << "4) Addieren von zwei Brüchen\n";
  std::cout << "5) Subtrahieren von zwei Brüchen\n";
  std::cout << "6) Multiplizieren von zwei Brüchen\n";
  std::cout << "7) Dividieren von zwei Brüchen\n";
}
```

```
void menu_01 ()
      // ---------------------------------
      // ggT und kgV
      // ---------------------------------
{
  cls ();
  std::cout << "GgT und kgV von zwei Za hlen\n";
  std::cout << "--------------------------\n";
  gotoline ();

  big_number a, b, ggt, kgv;

  gotoline ();
  std::cout << "Geben Sie die erste  Zahl ein ";
  gotoline (); dash ();
  a.get_number ();

  gotoline ();
  std::cout << "Geben Sie die zweite Zahl ein ";
  gotoline (); dash ();
  b.get_number (); gotoline ();

  ggt.gcd (a, b);
  kgv.lcm (a, b);


  std::cout << "Der ggT der beiden Zahlen lautet: \n";
  ggt.write ();
  gotoline ();
  gotoline ();

  std::cout << "Das kgV der beiden Zahlen lautet: \n";
  kgv.write ();

}


void menu_02 ()
      // ---------------------------------
      // Euklides
      // ---------------------------------
{
  cls ();
  std::cout << "Euklidischer Algorithmus\n";
  std::cout << "------------------------\n";
  gotoline ();

  big_number a, b;

  gotoline ();
  std::cout << "Geben Sie die erste  Zahl ein ";
  gotoline (); dash ();
  a.get_number ();

  gotoline ();
  std::cout << "Geben Sie die zweite Zahl ein ";
  gotoline (); dash ();
  b.get_number (); gotoline ();

  a.euclid (a, b);
  gotoline ();
  std::cout << "Der ggT der beiden Zahlen lautet: \n";
  a.write ();
  gotoline ();


}
```

```
void menu_03 ()
      // ---------------------------------
      // Bruch kürzen
      // ---------------------------------
{
  cls ();
  std::cout << "Einen Bruch kürzen\n";
  std::cout << "------------------\n";
  gotoline ();

  big_number a, b, ggt, r;

  gotoline ();
  std::cout << "Geben Sie den Zähler des Bruchs ein ";
  gotoline (); dash ();
  a.get_number ();

  gotoline ();
  std::cout << "Geben Sie den Nenner des Bruchs ein ";
  gotoline (); dash ();
  b.get_number (); gotoline ();

  ggt.gcd (a, b);
  a.div (a, ggt, r);
  b.div (b, ggt, r);

  std::cout << "Der Zähler des gekürzten Bruchs lautet: \n";
  a.write ();
  gotoline ();
  gotoline ();

  std::cout << "Der Nenner des gekürzten Bruchs lautet: \n";
  b.write ();

}


void menu_04 ()
      // ---------------------------------
      // Summe
      // ---------------------------------
{
  cls ();
  std::cout << "Zwei Brüche addieren\n";
  std::cout << "--------------------\n";
  gotoline ();

  big_number z1, n1, z2, n2, kgv, ggt, r, faktor;
  big_number zero;
  zero.put_to_zero ();

  gotoline ();
  std::cout << "Geben Sie den Zähler des ersten Bruchs ein ";
  gotoline (); dash ();
  z1.get_number ();

  gotoline ();
  std::cout << "Geben Sie den Nenner des ersten Bruchs ein ";
  gotoline (); dash ();
  n1.get_number (); gotoline ();

  gotoline ();
  std::cout << "Geben Sie den Zähler des zweiten Bruchs ein ";
  gotoline (); dash ();
  z2.get_number ();

  gotoline ();
```

```
  std::cout << "Geben Sie den Nenner des zweiten Bruchs ein ";
  gotoline (); dash ();
  n2.get_number (); gotoline ();

  kgv.lcm (n1, n2);     // Gemeinsamer Nenner

  faktor.div (kgv, n1, r);  // Mit diesem 'faktor' muss der erste
                            // Bruch erweitert werden
  z1.mult (faktor, z1);     // Ergibt den neuen Zähler

  faktor.div (kgv, n2, r);  // Mit diesem 'faktor' muss der zweite
                            // Bruch erweitert werden
  z2.mult (faktor, z2);     // Ergibt den neuen Zähler

  z1.sum (z1, z2);          // z1 ist jetzt der neue Zähler,
  n1 = kgv;                 // kgv ist der neue Nenner

  // Der Bruch muss noch gekürzt werden, aber nur wenn der Zähler
  // von 0 verschieden ist:
  if (zero.equal (zero, z1) == false)
    {
      ggt.gcd (z1, n1);
      z1.div (z1, ggt, r);
      n1.div (n1, ggt, r);
    }
  else n1.put_to_one ();  // Nenner 1 setzen, falls Zähler == 0

  std::cout << "Der Zähler des Bruchs lautet: \n";
  z1.write ();
  gotoline ();
  gotoline ();

  std::cout << "Der Nenner des Bruchs lautet: \n";
  n1.write ();

}


void menu_05 ()
      // ---------------------------------
      // Differenz
      // ---------------------------------
{
  cls ();
  std::cout << "Zwei Brüche subtrahieren\n";
  std::cout << "------------------------\n";
  gotoline ();

  big_number z1, n1, z2, n2, kgv, ggt, r, faktor;
  bool ist_negativ = false;
  big_number zero;
  zero.put_to_zero ();

  gotoline ();
  std::cout << "Geben Sie den Zähler des ersten Bruchs ein ";
  gotoline (); dash ();
  z1.get_number ();

  gotoline ();
  std::cout << "Geben Sie den Nenner des ersten Bruchs ein ";
  gotoline (); dash ();
  n1.get_number (); gotoline ();

  gotoline ();
  std::cout << "Geben Sie den Zähler des zweiten Bruchs ein ";
  gotoline (); dash ();
  z2.get_number ();
```

```
  gotoline ();
  std::cout << "Geben Sie den Nenner des zweiten Bruchs ein ";
  gotoline (); dash ();
  n2.get_number (); gotoline ();

  kgv.lcm (n1, n2);     // Gemeinsamer Nenner

  faktor.div (kgv, n1, r);  // Mit diesem 'faktor' muss der erste
                            // Bruch erweitert werden
  z1.mult (faktor, z1);     // Ergibt den neuen Zähler

  faktor.div (kgv, n2, r);  // Mit diesem 'faktor' muss der zweite
                            // Bruch erweitert werden
  z2.mult (faktor, z2);     // Ergibt den neuen Zähler

  if (z1.bigger (z1, z2) == false) ist_negativ = true;
  // In diesem Fall müssen die Zähler vertauscht werden
  if (ist_negativ == true)
    {
       r  = z1;
       z1 = z2;
       z2 = r;
    }

  z1.subt (z1, z2);         // z1 ist jetzt der neue Zähler,
  n1 = kgv;                 // kgv ist der neue Nenner

  // Der Bruch muss noch gekürzt werden, aber nur wenn der Zähler
  // von 0 verschieden ist:
  if (zero.equal (zero, z1) == false)
    {
      ggt.gcd (z1, n1);
      z1.div (z1, ggt, r);
      n1.div (n1, ggt, r);
    }
  else n1.put_to_one ();  // Nenner 1 setzen, falls Zähler == 0

  std::cout << "Der Zähler des Bruchs lautet: \n";
  z1.write ();
  gotoline ();
  gotoline ();

  std::cout << "Der Nenner des Bruchs lautet: \n";
  n1.write ();

  gotoline ();
  if (ist_negativ == true)
     if (zero.equal (zero, z1) == false)  // 0 wird nicht als negativ
                                          // bezeichnet
        std::cout << "Der Bruch ist negativ! \n";

}


void menu_06 ()
      // ---------------------------------
      // Multiplizieren von Brüchen
      // ---------------------------------
{
  cls ();
  std::cout << "Zwei Brüche multiplizieren\n";
  std::cout << "--------------------------\n";
  gotoline ();

  big_number z1, n1, z2, n2, zzz, nnn, ggt, r;
  big_number zero;
  zero.put_to_zero ();
```

```
  gotoline ();
  std::cout << "Geben Sie den Zähler des ersten Bruchs ein ";
  gotoline (); dash ();
  z1.get_number ();

  gotoline ();
  std::cout << "Geben Sie den Nenner des ersten Bruchs ein ";
  gotoline (); dash ();
  n1.get_number (); gotoline ();

  gotoline ();
  std::cout << "Geben Sie den Zähler des zweiten Bruchs ein ";
  gotoline (); dash ();
  z2.get_number ();

  gotoline ();
  std::cout << "Geben Sie den Nenner des zweiten Bruchs ein ";
  gotoline (); dash ();
  n2.get_number (); gotoline ();

  zzz.mult (z1, z2);  // Neuer Zähler
  nnn.mult (n1, n2);  // Neuer Nenner

  // Der Bruch muss noch gekürzt werden, aber nur wenn der Zähler
  // von 0 verschieden ist:
  if (zero.equal (zero, zzz) == false)
    {
      ggt.gcd (zzz, nnn);
      zzz.div (zzz, ggt, r);
      nnn.div (nnn, ggt, r);
    }
  else nnn.put_to_one ();  // Nenner 1 setzen, falls Zähler == 0

  std::cout << "Der Zähler des Bruchs lautet: \n";
  zzz.write ();
  gotoline ();
  gotoline ();

  std::cout << "Der Nenner des Bruchs lautet: \n";
  nnn.write ();

}



void menu_07 ()
      // ----------------------------------
      // Dividieren von Brüchen
      // Multiplizieren mit Reziprokwert
      // ----------------------------------
{
  cls ();
  std::cout << "Zwei Brüche dividieren\n";
  std::cout << "----------------------\n";
  gotoline ();

  big_number z1, n1, z2, n2, zzz, nnn, ggt, r;
  big_number zero;
  zero.put_to_zero ();

  gotoline ();
  std::cout << "Geben Sie den Zähler des ersten Bruchs ein ";
  gotoline (); dash ();
  z1.get_number ();

  gotoline ();
  std::cout << "Geben Sie den Nenner des ersten Bruchs ein ";
  gotoline (); dash ();
```

```
  n1.get_number (); gotoline ();

  gotoline ();
  std::cout << "Geben Sie den Zähler des zweiten Bruchs ein ";
  gotoline (); dash ();
  z2.get_number ();

  gotoline ();
  std::cout << "Geben Sie den Nenner des zweiten Bruchs ein ";
  gotoline (); dash ();
  n2.get_number (); gotoline ();

  // Reziprokwert
  r = z2; z2 = n2; n2 = r;

  zzz.mult (z1, z2);  // Neuer Zähler
  nnn.mult (n1, n2);  // Neuer Nenner

  // Der Bruch muss noch gekürzt werden, aber nur wenn der Zähler
  // von 0 verschieden ist:
  if (zero.equal (zero, zzz) == false)
    {
      ggt.gcd (zzz, nnn);
      zzz.div (zzz, ggt, r);
      nnn.div (nnn, ggt, r);
    }
  else nnn.put_to_one ();  // Nenner 1 setzen, falls Zähler == 0

  std::cout << "Der Zähler des Bruchs lautet: \n";
  zzz.write ();
  gotoline ();
  gotoline ();

  std::cout << "Der Nenner des Bruchs lautet: \n";
  nnn.write ();

}
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -


int main ()
      // *********************************
      //      Main function
      // *********************************
{
  char s [25];

  cls ();

  menu ();
  std::cout << "\n\n";
  std::cout << "Wählen Sie eine Option und drücken Sie <Intro> ";

  cin.getline (s, 25);

  if (s [0] == 49)   menu_01 ();   // ggT, kgV
                                   // ASCII-code of "1" is 49
  if (s [0] == 50)   menu_02 ();   // Euklides
  if (s [0] == 51)   menu_03 ();   // Kürzen
  if (s [0] == 52)   menu_04 ();   // Addieren
  if (s [0] == 53)   menu_05 ();   // Subtrahieren
  if (s [0] == 54)   menu_06 ();   // Multiplizieren
  if (s [0] == 55)   menu_07 ();   // Dividieren

  gotoline ();

return 0;
}
```