Geometrische Objekte

- Machen Sie sich mit der Entwicklungsumgebung `KDevelop` vertraut. Lesen Sie dazu unsere Kurzanleitung `KDevelop` und gehen Sie die Punkte einzeln durch.
- Definieren Sie ein Projekt und schreiben Sie ein Programm `"Hallo Welt!"`.
- Schreiben Sie ein Programm, welches folgenden Text ausgibt:

```
********************************
*        Programmieren 1       *
*                              *
*      Vorname, Nachname       *
*        Matrikelnummer        *
********************************
```

  Dabei sollen Sie Ihren Namen und Ihre Matrikelnummer einsetzen.

- Definieren Sie Variablen vom Typ `double`, welche die Abmessungen eines Kegelstumpfes beschreiben. Für die Höhe und die Radien von Grund- und Deckfläche nehmen Sie folgende Werte an: $h = 7.5$, $r_1 = 13.2$, $r_2 = 6.4$.
  Anleitung:

  $$\text{Mantelfläche:} \quad M = \pi \cdot (r_1 + r_2) \cdot \sqrt{h^2 + (r_1 - r_2)^2}$$

  $$\text{Oberfläche:} \quad F = \pi \cdot (r_1^2 + r_2^2) + M$$

  $$\text{Volumen:} \quad V = \frac{\pi \cdot h}{3} \cdot (r_1^2 + r_1 \cdot r_2 + r_2^2)$$

  Zur Berechnung der Quadratwurzel $\sqrt{x}$ verwenden Sie die Funktion `sqrt(x)` aus der Mathematik-Bibliothek. Wie Sie Bibliotheken einbinden, finden Sie in der Kurzanleitung `KDevelop`, Kapitel 3 (Einbinden von Bibliotheken).

  In diesem Fall muss die Datei `math.h` eingebunden werden (`#include <math.h>`). Außerdem ist die Bibliothek *math* im Menü *Linker Options* hinzuzufügen (Kapitel 3.1).

- Geben Sie die Abmessungen des Kegelstumpfes und die berechneten Werte am Bildschirm aus, etwa in der Form:

```
Hoehe   = 7.5
Radius1 = 13.2
Radius2 = 6.4
Flaeche = ...
Volumen = ...
```

Referenzlösung

1 Allgemeines

In der ersten Übungseinheit lernen Sie die Entwicklungsumgebung `KDevelop` kennen und entwickeln ein erstes, einfaches C-Programm. In diesem Programm sind zuerst konstante Zeichenketten auszugeben, wie dies im Programm `"Hallo Welt!"` gezeigt wird, und danach arithmetische Ausdrücke auszuwerten und das Ergebnis auszugeben. Der Stoff für die erste Übungseinheit umfasst die Kapitel 1 bis 5 im Buch "Programmieren in C".

2 Textausgabe

Wir folgen dem Aufbau des Programms `"Hallo Welt!"` und geben jede Zeile mit einer eigenen `printf` Anweisung aus. Es werden ausschließlich konstante Zeichenketten ausgegeben, weshalb die Definition von Variablen hier noch nicht erforderlich ist.

```
/*** Datei: Banner.c   ***/

#include <stdio.h>
main ()
{
    printf("*******************************\n");
    printf("*        Programmieren 1       *\n");
    printf("*                              *\n");
    printf("*      Vorname, Nachname       *\n");
    printf("*        Matrikelnummer        *\n");
    printf("*******************************\n");
}
```

Es ist selbstverständlich möglich, zwei oder mehrere Zeilen mit einer einzigen `printf` Anweisung auszugeben, indem man in die Zeichenkette mehrere *newline*-Zeichen ("`\n`") einfügt. In diesem Beispiel würde dies allerdings die Lesbarkeit des Programms verringern.

3 Auswertung von Formeln

Im vorliegenden Beispiel sind die Oberfläche und das Volumen eines Kegelstumpfes zu berechnen. Der Kegelstumpf wird durch die drei Parameter h, r_1, r_2 charakterisiert. Da das Einlesen von Daten erst später im Stoff behandelt wird, müssen wir in dieser Übung die Parameter-Werte fest in das Programm schreiben ("*hart codieren*"). Zur Darstellung der reellen Zahlen verwenden wir Gleitpunkt-Zahlen, also Variablen und Konstanten vom Typ `double`.

Die Berechnung der Quadratwurzel oder einer Potenz erfolgt durch die Funktion `sqrt()` oder `pow()` aus der Mathematik-Bibliothek. Die Verwendung einer Funktion aus der Mathematik-Bibliothek erfordert, dass Sie erstens folgende Header-Datei einbinden

```
#include <math.h>
```

und zweitens, dass Sie im KDevelop (siehe Kurzanleitung für KDevelop) die Bibliothek *math* bei dem Punkt *Linker Flags* hinzufügen. Mit dieser Einstellung wird bewirkt, dass der Compiler `cc` mit der Option `-lm` aufgerufen wird:

```
cc -o KegelStumpf KegelStumpf.c -lm
```

In der ersten Version des Programms werden Zwischenergebnisse in eigenen Variablen gespeichert. Außerdem wird hier noch kein Versuch unternommen, die Anzahl der Additionen und Multiplikationen gering zu halten.

```
/*** Datei: KegelStumpf.c ***/

#include <stdio.h>
#include <math.h>
main ()
{
   /* Definition der Konstanten PI */
   const double PI = 3.14159265358979323846;

   /* Definition der Variablen */
   double hoehe;
   double radius1;
   double radius2;
   double mantel;
   double seite;
   double flaeche;
   double volumen;

   /* hier stehen die Aufrufe von printf (), siehe oben */

   /* Wertzuweisungen */
   hoehe   = 7.5;
   radius1 = 13.2;
   radius2 = 6.4;

   /* Berechnungen */
   seite   = sqrt(hoehe * hoehe + (radius1 - radius2) * (radius1 - radius2));
   mantel  = PI * (radius1 + radius2) * seite;
   flaeche = PI * (radius1 * radius1 + radius2 * radius2) + mantel;
   volumen = (PI * hoehe / 3) * (radius1 * radius1 + radius1 * radius2
                              +  radius2 * radius2);

   /* Ausgabe */
   printf("Hoehe   = %g\n", hoehe);
   printf("Radius1 = %g\n", radius1);
   printf("Radius2 = %g\n", radius2);
   printf("Flaeche = %g\n", flaeche);
   printf("Volumen = %g\n", volumen);
}
```

Nun versuchen wir, das Programm im Hinblick auf die Anzahl der Variablen und die Anzahl der Operationen zu verbessern. Die Zwischenergebnisse für die Seitenlänge des Mantels und die Mantelfläche müssen nicht in eigenen Variablen gespeichert werden. Wir führen aber eine Hilfsvariable `h` ein, die temporäre Zwischenergebnisse aufnehmen kann. Die Konstante `PI` kann im Programm selbst definiert werden, oder man verwendet die Konstante `M_PI`, welche in der Header-Datei `math.h` bereits vordefiniert ist. Weiters können die Variablen bereits bei ihrer Definition mit dem gewünschten Wert initialisiert werden.

```
/*** Datei: KegelStumpf.c ***/

#include <stdio.h>
#include <math.h>
main ()
{

    /* Definition und Initialisierung der Variablen */
    double hoehe   = 7.5;
    double radius1 = 13.2;
    double radius2 = 6.4;
    double h;              // Hilfsvariable
    double flaeche;
    double volumen;

    /* hier stehen die Aufrufe von printf (), siehe oben */

    /* Berechnungen */
    h = radius1 - radius2;

    /* an flaeche wird jetzt ein Teilergebnis zugewiesen */
    flaeche = (radius1 + radius2) * sqrt(hoehe * hoehe + h * h);
    h = radius1 * radius1 + radius2 * radius2;
    flaeche = M_PI * (h + flaeche);
    volumen = M_PI * hoehe * (h + radius1 * radius2) / 3;

    /* Ausgabe */
    printf("Hoehe   = %g\n", hoehe);
    printf("Radius1 = %g\n", radius1);
    printf("Radius2 = %g\n", radius2);
    printf("Flaeche = %g\n", flaeche);
    printf("Volumen = %g\n", volumen);
}
```

Die erste Programm-Version enthält noch 21 Anweisungen, während die verbesserte Version mit 16 auskommt.

Konvertieren von Maß-Einheiten

Entwickeln Sie ein C-Programm zur Umrechnung von Energie-Einheiten.

- Erstellen Sie am Anfang Ihres Programms einen Kommentar, der den Projektnamen, Ihren Namen, Ihre Matrikelnummer und das Datum enthält.
- Lesen Sie eine Zahl ein, die eine Energie in Joule (J) angibt. Programmieren Sie ein Menü, in dem der Benutzer folgende Einheiten zur Auswahl hat:

  Kalorien (cal)
  Elektronenvolt (eV)
  Kilowattstunden (kWh)
  British thermal units (Btu)

  Es gelten folgende Umrechnungsfaktoren:

  1 cal = 4.1868 J
  1 eV = $1.6022 \cdot 10^{-19}$ J
  1 kWh = $3.6 \cdot 10^{6}$ J
  1 Btu = 1055.1 J

  Geben Sie die Energie in der gewünschten Einheit aus.

Referenzlösung

Wir entwickeln ein Menü, in dem der Benutzer durch Eingabe einer Zahl einen Menüpunkt auswählt.

```
/*** Datei: EnergieKonversion.c ***/
#include <stdio.h>
main ()
{
   /* Konvertierungsfaktoren */
   const double convertCal = 4.1868;
   const double convertEV  = 1.6022e-19;
   const double convertKWh = 3.6e6;
   const double convertBtu = 1055.1;

   long   choice;
   double e1, e2;

   printf ("\n\tGeben Sie einen Energiewert in Joule ein: ");
   scanf  ("%lf", &e1);

   printf("\n\tWaehlen Sie die Energie-Einheit aus:\n\n");
   printf("\t\t1 ... Cal\n");
   printf("\t\t2 ... eV\n");
   printf("\t\t3 ... kWh\n");
   printf("\t\t4 ... Btu\n");
   printf("\n\tIhre Wahl: ");
   scanf ("%ld", &choice);
   printf("\n");

   switch (choice)
   {
      case 1:
        e2 = e1 / convertCal ;
        printf("\t%g Joule = %g Kalorien\n", e1, e2);
        break;
      case 2:
        e2 = e1 / convertEV ;
        printf("\t%g Joule = %g Elektronenvolt\n", e1, e2);
        break;
      case 3:
        e2 = e1 / convertKWh ;
        printf("\t%g Joule = %g Kilowattstunden\n", e1, e2);
        break;
      case 4:
        e2 = e1 / convertBtu ;
        printf("\t%g Joule = %g British thermal units\n", e1, e2);
        break;
      default:
        printf("\t+++ Falsche Eingabe! +++\n\n");
   }
   printf("\n");
}
```

Da die Konversions-Faktoren fest vorgegeben sind, werden sie als Konstanten vereinbart. Der Dialog mit dem Benutzer wird großzügig mit Leerzeilen und Einrückungen aufgelockert.

In der folgenden Variante werden im Menü Zeichen anstatt Zahlen verwendet. Beim Einlesen von Zeichen ist folgende Besonderheit der Eingabe-Funktionen von C zu berücksichtigen. Die Eingabe wird gepuffert und erst nach Betätigen der ENTER-Taste an das Programm übermittelt. Dies bewirkt, dass sich nach der Eingabe noch das Zeichen `newline`, also `'\n'`, im Eingabepuffer befindet. Dieses Zeichen muss mit einem weiteren Aufruf von `getchar()` überlesen werden:

```
// ...
scanf ("%lf", &zahl1);
getchar();                /* newline Zeichen ueberlesen */
```

Eine verkürzte Variante des Beispiels sieht dann wie folgt aus.

```
/*** Datei: EnergieKonversion.c    ***/
#include <stdio.h>
main ()
{
   /* Konvertierungsfaktoren  */
   const double convertCal = 4.1868;
   const double convertEV  = 1.6022e-19;
   const double convertKWh = 3.6e6;

   char   choice;
   double e1, e2;

   printf ("\n\tGeben Sie einen Energiewert in Joule ein: ");
   scanf  ("%lf", &e1);
   getchar();                /* newline Zeichen ueberlesen */

   printf("\n\tWaehlen Sie die Energie-Einheit aus:\n\n");
   printf("\t\t c ... Cal\n");
   printf("\t\t v ... eV\n");
   printf("\t\t w ... kWh\n");
   printf("\n\tIhre Wahl: ");
   scanf ("%c", &choice);
   getchar();                 /* newline Zeichen ueberlesen */

   printf("\n");
   switch (choice)
   {
      case 'c':
        e2 = e1 / convertCal ;
        printf("\t%g Joule = %g Kalorien\n", e1, e2);
        break;
      case 'v':
        e2 = e1 / convertEV ;
        printf("\t%g Joule = %g Elektronenvolt\n", e1, e2);
        break;
      case 'w':
        e2 = e1 / convertKWh ;
        printf("\t%g Joule = %g Kilowattstunden\n", e1, e2);
        break;
      default:
        printf("\t+++ Falsche Eingabe! +++\n\n");
   }
}
```

Prüfungsbeurteilung

Entwickeln Sie ein C-Programm zur Umrechnung eines Prüfungsergebnisses in eine Beurteilung. Bei einer schriftlichen Prüfung sind maximal 100 Punkte zu erreichen. Die Noten werden wie folgt vergeben:

Punkte	Note
0-40	Nicht genügend
41-55	Genügend
56-70	Befriedigend
71-85	Gut
86-100	Sehr gut

- Erstellen Sie am Anfang Ihres Programms einen Kommentar, der den Projektnamen, Ihren Namen, Ihre Matrikelnummer und das Datum enthält.
- Lesen Sie die Anzahl der Punkte ein und geben Sie die Beurteilung aus. Überprüfen Sie, ob die eingegebene Punktezahl gültig ist.

Referenzlösung

Zuerst ist aus der gegebenen Punkte-Anzahl die Beurteilung zu bestimmen. Dazu wird ein `if`-Rechen verwendet. Man kann die Abfragen sowohl aufsteigend (zuerst auf "Nicht genügend" abfragen) als auch absteigend anordnen (zuerst auf "Sehr Gut" abfragen).

```
/*** Datei: Beurteilung.c ***/
#include <stdio.h>
#include <stdlib.h>

main ()
{
  long punkte;
  long min = 10;
  long max = 100;
  int  seed;
  double r;

  printf("\n\tGeben Sie eine ganze Zahl ein: ");
  scanf("%d", &seed);

  /* initialisiere Zufallszahlengenerator */
  srand(seed);

  /* erzeuge Zufallszahl */
  r = rand() / (RAND_MAX + 1.0);

  punkte = min + (long) ((max - min + 1) * r);

  printf("\n\t%d Punkte sind ", punkte);
  if (punkte <= 40)
    printf("\"Nicht genuegend\"\n");
  else if (punkte <= 55)
    printf("\"Genuegend\"\n");
  else if (punkte <= 70)
    printf("\"Befriedigend\"\n");
  else if (punkte <= 85)
    printf("\"Gut\"\n");
  else
    printf("\"Sehr Gut\"\n");

  printf("\n");
}
```

In diesem Programm werden auch doppelte Hochkommata `(")` in einer Zeichenkette ausgegeben. Da dies innerhalb einer Zeichenkette Sonderzeichen sind, ist jeweils ein backslash Zeichen `'\'` voranzustellen.

Die Funktion `rand()` liefert eine ganzzahlige Pseudo-Zufallszahl zwischen 0 und `RAND_MAX`. Die Konstante `RAND_MAX` wird in `stdlib.h` definiert und ist sehr viel größer als 1. Durch die Anweisung

```
double r = rand() / (RAND_MAX + 1.0);
```

erhält man eine gleichverteilte Zufallszahl im Intervall $[0, 1[$. Die Addition von `1.0` im Nenner bewirkt, dass das Intervall rechts offen ist. Es ist zu beachten, dass eine Funktion keine echten Zufallszahlen erzeugen kann, sondern nach einem bestimmten Algorithmus eben Pseudo-Zufallszahlen erzeugt. Würde man den Aufruf von `srand()` weglassen, würde `rand()` bei einmaligem Aufruf immer denselben Wert, und bei wiederholtem Aufruf immer dieselbe Zahlenfolge liefern.

1 Erzeugung einer diskreten Zufallszahl

Um eine gleichverteilte, diskrete Zufallszahl aus dem Intervall `[min, max]` zu erhalten, wendet man folgende Transformation auf die Zufallszahl `r` an.

```
long n = min + (long) ((max - min + 1) * r);
```

Dabei ist `(max - min + 1)` die Anzahl der ganzen Zahlen im Intervall `[min,max]`.

2 Erzeugung einer kontinuierlichen Zufallszahl

Um eine gleichverteilte, kontinuierliche Zufallszahl aus dem Intervall `[a, b[` zu erzeugen, verwendet man die folgende Transformation:

```
double rab = a + (b - a) * r;
```

Es wird zuerst die Länge des Intervalls durch Multiplikation mit `(a - b)` angepasst, und dann das neue Intervall um `a` verschoben.

Beispiel: Geometrische Reihe

Entwickeln Sie ein C-Programm, das folgende Summe berechnet:

$$S = 1 + q + q^2 + \ldots + q^n$$

Die Ordnung n und der Multiplikator q werden eingelesen. Werten Sie die obige Summe direkt mit Hilfe einer Schleife aus. Der Wert der Summe kann auch aus der Summenformel

$$S = \frac{1 - q^{n+1}}{1 - q}$$

bestimmt werden. Berechnen Sie den Wert der Summe auf beide Arten und geben Sie das Ergebnis aus.

Hinweise:

- Die Summe ist nur für positive Werte von n definiert.
- Die Glieder der Summe können auf einfache Weise durch Rekursion berechnet werden.

$$S = \sum_{j=0}^{n} a_j \,, \qquad a_0 = 1 \,, \qquad a_j = q \cdot a_{j-1}$$

- Vermeiden Sie eine Division durch Null.

Referenzlösung

Eine naive Form der Summenberechnung, die hier nicht empfohlen wird, ist im folgenden gezeigt. Jedes Element der Summe wird mit der Funktion `pow()` berechnet.

```
#include <stdio.h>
#include <math.h>

main()
{
   long   n;
   long   i;
   double multiplier;
   double sum;

   /* Einlesen der Parameter */
   printf("\nMultiplikator: ");
   scanf("%lf", &multiplier);
   printf("Hoechste Ordnung: ");
   scanf("%ld", &n);

   /* Ueberpruefung der Eingabe */
   if (n < 1)
     printf ("\n\t### Falsche Eingabe! ###\n\n");
   else
   {
      /* Berechnen der Summe */
      sum  = 1;
      for (i = 1; i <= n; i++)
        sum  = sum + pow(multiplier, i);

      /* Ausgabe des Resultats */
      printf("Summe = %g\n", sum);



      /* Direkte Berechnung der Summe */
      if (multiplier != 1.0)
      {
        sum = (1. - pow(multiplier,n+1)) / (1. - multiplier);

        /* Ausgabe der direkten Summe */
        printf("Summe, direkt = %g\n", sum);
      }
      else
      {
        printf ("\n\t### Division durch Null ! ###\n\n");
      }
   }
}
```

Beispiel: Reihen-Berechnung (Sinus)

Schreiben Sie ein C-Programm, das folgende Summe berechnet:

$$S = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots + (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

- Die Ordnung n und die Variable x werden eingelesen. Die Summe ist nur sinnvoll, wenn zumindest ein Summenglied existiert. Es muss also $n \geq 0$ gelten. Außerdem schränken wir x auf den Bereich $|x| \leq \pi$ ein. Entwickeln Sie zwei Funktionen, die einen Wert einlesen und den Wertebereich überprüfen:

```
long   readOrder()
double readArgument()
```

- Entwickeln Sie eine Funktion, welche die Summe mit Hilfe einer Schleife auswertet:

```
double sumSin(double x, long n)
```

  Die Glieder der Summe können auf einfache Weise durch Rekursion berechnet werden.

$$S = a_0 + a_1 + \ldots + a_n$$

$$a_0 = x$$

$$a_i = -\frac{x^2}{2i \cdot (2i+1)} \cdot a_{i-1}, \qquad i = 1, 2, \ldots n$$

- Geben Sie im Hauptprogramm den Wert der Summe aus. Diese Summe ergibt einen Näherungswert für die Funktion `sin(x)`. Geben Sie daher zum Vergleich auch den Wert von `sin(x)` aus.

Referenzlösung

Wir beginnen mit den Einlese-Funktionen. Da keine Argumente benötigt werden, bleibt die Argumentliste leer.

```
long readOrder()
{
   long n;

   /* Start der Kommunikation mit dem Benutzer */
   printf("Bitte geben Sie eine positive Zahl ein: ");
   scanf ("%ld", &n);

   /* Ueberpruefe den Wert der Zahl */
   while (n < 0)
   {
     printf("Falsche Eingabe!\n");
     printf("Bitte geben Sie eine positive Zahl ein: ");
     scanf ("%ld", &n);
   }

   return n;
}

double readArgument()
{
   double x;

   printf("Geben Sie x im Bereich [-PI, +PI] ein: ");
   scanf ("%lf", &x);

   while (fabs(x) > M_PI)
   {
     printf("Falsche Eingabe!\n");
     printf("Bitte geben Sie x im Bereich [-PI, +PI] ein: ");
     scanf ("%lf", &x);
   }

   return x;
}
```

Die beiden Funktionen sind gleich aufgebaut. Zuerst wird der Benutzer aufgefordert, eine Zahl einzugeben. Danach wird überprüft, ob die Zahl im vorgegebenen Wertebereich liegt. Falls nicht, wird eine Fehlermeldung ausgegeben und der Benutzer erneut aufgefordert, eine Zahl einzugeben. Dieser Vorgang wird wiederholt, bis der Benutzer eine gültige Zahl eingegeben hat.

Diese Funktionen können noch verbessert werden, indem man je einen Aufruf von `printf()` und `scanf()` einspart. Dazu müssen wir eine Schleife programmieren, in der die Schleifenbedingung im Schleifenblock selbst überprüft wird. Mit `while(1)` programmiert man eine Endlos-Schleife, die dann mit einer `break` Anweisung beendet wird.

```
long readOrder()
{
   long n;

   while(1)
   {
      printf("Bitte geben Sie eine positive Zahl ein: ");
      scanf ("%ld", &n);

      if (n >= 0) break;

      printf("Falsche Eingabe!\n");
   }
   return n;
}
```

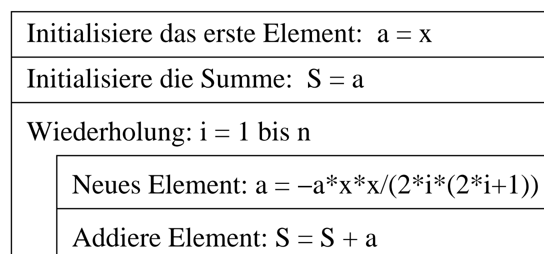
Im nächsten Schritt entwickeln wir die Funktion `sumSin()`, welche die eigentliche Summenberechnung durchführt. Das erste Glied der Summe ist bekannt: $a_0 = x$. Dividiert man zwei benachbarte Glieder, ergibt sich ein einfacher Ausdruck:

$$\frac{a_i}{a_{i-1}} = -\frac{x^2}{2i \cdot (2i+1)}, \qquad i \geq 1$$

Diesen Ausdruck benutzt man, um das neue Element a_i aus dem bekannten Element a_{i-1} zu bestimmen:

$$a_i = -\frac{x^2}{2i \cdot (2i+1)} \cdot a_{i-1}, \qquad i \geq 1$$

Damit lässt sich die Summe wie im folgenden Struktogramm berechnen:



Dieses Struktogramm wird nun in eine C-Funktion umgesetzt. Da die Anzahl der Schleifen-Wiederholungen vorgegeben ist, wird eine `for` Schleife verwendet.

```
double sumSin(double x, long n)
{
   long   i;
   double a;
   double sum;

   a   = x;
   sum = a;
   for (i = 1; i <= n; i++)
   {
      a   = -a * x * x / (2 * i * (2 * i + 1));
      sum = sum + a;
   }
   return sum;
}
```

Im letzten Schritt entwickeln wir das Hauptprogramm. In der C-Datei werden zuerst die Funktion `main()` und dann die weiteren Funktionen definiert. Diese Reihenfolge macht es erforderlich, dass die Funktionen vor ihrem Aufruf dem Compiler durch eine Deklaration bekanntgegeben werden.

```
#include <stdio.h>
#include <math.h>

/* Deklaration der Funktionen   */
long   readOrder();
double readArgument();
double sumSin(double x, long n);

int main()
{
  long   n;
  double x;
  double sum;
  double y;

  n = readOrder();
  x = readArgument();

  sum = sumSin(x, n);

  y = sin(x);

  printf("sum    = %g\n", sum);
  printf("sin(x) = %g\n", y);

  return 0;
}

/* Definition der Funktionen  */

long readOrder()
{
  ....
}

double readArgument()
{
  ...
}

double sumSin (double x, long n)
{
  ...
}
```

Da die verschiedenen Aufgaben wie das Einlesen der Parameter und die Summenberechnung durch Funktionen realisiert werden, ist das Hauptprogramm kurz und klar strukturiert.

Beispiel: Dezimal-Darstellung einer Zahl

Es sei eine positive ganze Zahl d gegeben. Gesucht sind die Dezimal-Stellen a_i der Zahl d.

$$d = a_0 + a_1 \cdot 10 + a_2 \cdot 10^2 + \ldots + a_N \cdot 10^N$$

- Zum Einlesen einer ganzen Zahl $n \geq 0$ entwickeln Sie eine Funktion `long readPos()`.
- Entwickeln Sie eine Funktion, welche die Stellen einer Zahl im Dezimal-System berechnet und in einem Feld speichert. Die einzelnen Stellen können durch folgende Anweisungen bestimmt werden:

  $a_0 = d$ Modulo 10 (Rest der Ganzzahl-Division von d/10)
  $d = d/10$ (Ganzzahl-Division)
  $a_1 = d$ Modulo 10
  $d = d/10$
  usw.
  bis d den Wert 0 annimmt.

  Vorschlag für den Funktionskopf:

  ```
  long dec2digit (long d, long digits[], long len)
  ```

  Die Länge `len` des Feldes wird als zusätzlicher Parameter übergeben.

  Der Rückgabewert der Funktion wird als Fehlerindikator verwendet. Falls die Zahl mit der gegebenen Anzahl von Stellen nicht dargestellt werden kann, soll die Funktion den Wert 1 zurückgeben, andernfalls 0. Achten Sie darauf, dass im Fehlerfall die Feldgrenzen nicht überschritten werden.
- Zum Ausgeben der berechneten Stellen soll eine Funktion `void printDigits (long digits[], long len)` entwickelt werden.
- Im Hauptprogramm werden die Funktionen aufgerufen. Zuerst wird eine nicht-negative Zahl eingelesen, diese in ihre Dezimal-Stellen zerlegt, und die Stellen ausgegeben.
- Führen Sie im Hauptprogramm eine Schleife ein, in welcher der Benutzer die Berechnung mit einer anderen Zahl wiederholen kann.

Erweiterungen

- Schreiben Sie eine Funktion, welche die umgekehrte Aufgabe durchführt: Aus einem Feld mit den Dezimal-Stellen einer natürlichen Zahl wird der Wert der Zahl berechnet. Vorschlag für den Funktionskopf:

  ```
  long digit2dec (long *d, long digits[], long len )
  ```

  Achten Sie darauf, dass der Bereich der Variablen vom Typ `long` nicht überschritten wird. Der größte mögliche Wert einer `long` Variablen ist `LONG_MAX` und ist in der Header Datei `limts.h` definiert. Falls die Zahl nicht in einer `long` Variablen gespeichert werden kann, soll die Funktion den Wert 1 zurückgeben, andernfalls 0. Zum Testen Ihrer Funktionen bilden Sie die Differenz zwischen der berechneten und der eingelesenen Zahl.

Referenzlösung

Einlesefunktion

Wir entwickeln eine Funktion zum Einlesen einer ganzen Zahl $n \geq 0$. Die Funktion führt zusätzlich eine einfache Fehlerbehandlung durch.

```
long readPos()
{
 /* Eingabe einer positiven ganzen Zahl */

  long d = -1;
  long r;

  while (1)
  {
    printf("Eingabe einer positiven ganzen Zahl: ");

    r = scanf("%ld", &d);

    /* Loeschen des Eingabebuffers */
    while (getchar() != '\n'); // lesen der Zeichen bis ein '\n' gefunden

    if (d >= 0 && r == 1) break;

    printf("Falsche Eingabe! \n");

  }
  return d;
}
```

Der Benutzer wird aufgefordert, eine Zahl einzugeben. Die Funktion `scanf()` wird aufgerufen und der Rückgabewert in `r` gespeichert. Anschließend werden solange Zeichen aus dem Eingabepuffer gelesen, bis ein newline-Zeichen gefunden wird. Bei einer korrekten Eingabe bleibt im Eingabepuffer nur das abschließende newline-Zeichen zurück, die Schleife wird in diesem Fall genau einmal durchlaufen. Bei einer fehlerhaften Eingabe, etwa wenn anstatt einer Zahl ein Buchstabe eingegeben wird, verändert `scanf()` den Inhalt des Eingabepuffers nicht, und der Puffer wird mit der `getchar()` Funktion geleert.

Der Rückgabewert von `scanf()` gibt an, wieviele Variablen korrekt konvertiert werden konnten. In unserem Beispiel ist nur eine `long` Variable zu konvertieren, also kann `scanf()` höchstens 1 liefern.

Falls also `r == 1` gilt, wurde eine `long` Variable richtig konvertiert. Falls der Wert dieser Variablen ≥ 0 ist, wird die Schleife mit `break` verlassen und der Wert mit `return` zurückgegeben. Andernfalls wird eine Fehlermeldung ausgegeben und der Vorgang wiederholt.

Berechnung der Dezimalstellen

Die Funktion `dec2digit` besitzt drei Argumente: die Zahl, von der die Stellen (engl. digits) zu bestimmen sind, ein Feld, in dem die Stellen zu speichern sind, und die Länge des Feldes.

```
long dec2digit (long d, long digits[], long len)
{
  long i;

  for (i = 0; i <  len; i++)
    digits[i] = 0;

  for (i = 0; (i < len) && (d != 0); i++)
  {
    digits[i] = d % 10;   // Rest der Ganzzahl-Division
    d = d / 10;           // Ganzzahl-Division
  }

  if (d == 0)
    return 0; // Falls erfolgreich, Rueckgabe von 0
  else
    return 1;
}
```

Zuerst wird das Feld `digits[]` in einer `for`-Schleife mit Null initialisiert. Die Schleifenbedingung der zweiten `for`-Schleife besteht aus einer logischen Verknüpfung von zwei Bedingungen. Die Bedingung `i < len` verhindert, dass die Feldgrenze überschritten wird. Die Bedingung `d != 0` ist durch den Algorithmus vorgegeben.

Der Algorithmus terminiert ordnungsgemäß, wenn die wiederholte Ganzzahl-Division `d = d/10` schließlich `d == 0` als Ergebnis liefert. In diesem Fall konnte die Zahl vollständig in ihre Stellen zerlegt werden. Die Funktion `dec2digit` meldet die erfolgreiche Beendigung des Algorithmus durch den Rückgabewert 0.

Falls nach der `for`-Schleife die Bedingung `d != 0` gilt, dann wurde die Schleife auf Grund der Bedingung `i == len` abgebrochen. In diesem Fall konnte die Zerlegung wegen dem Erreichen der Feldgrenze nicht erfolgreich beendet werden, und es wird 1 zurückgegeben.

Ausgeben der Dezimalstellen

```
void printDigits (long digits[], long len)
{
  /* Ueberspringen von fuehrenden Nullen */
  len = len - 1 ;
  while ((digits[len]) == 0 && (len > 0))
      len = len - 1;

  while (len >= 0)
  {
    printf ("%1ld", digits[len]);
    len = len - 1;
  }
  printf ("\n\n");
}
```

Zuerst wird berücksichtigt, dass die höheren Stellen Null sein können, und dass die Ausgabe dieser Nullen keine Information liefert. Zum Beispiel soll anstatt von `000000987` nur `987` ausgegeben werden. Daher werden in der ersten `while`-Schleife diese Stellen übersprungen. In der zweiten `while`-Schleife werden die signifikanten Stellen ausgegeben. Da der Wert jeder Dezimalstelle im Bereich [0,9] liegt, ist für die Ausgabe das Format `%1ld` ausreichend. Da wir es gewöhnt sind, die Stellen von links nach rechts, also von der höchsten zur niedrigsten geordnet anzuschreiben, wird das Feld in absteigender Reihenfolge durchlaufen.

Berechnung der Zahl aus den Dezimalstellen

Wir überlegen uns die Lösung für den konkreten Fall $n = 3$ und verallgemeinern dann die Lösung für beliebiges n.

$$d = a_0 + a_1 \cdot 10 + a_2 \cdot 10^2 + a_3 \cdot 10^3$$

Wir formen den Ausdruck um, indem wir 10 herausheben,

$$d = a_0 + 10 \cdot (a_1 + 10 \cdot (a_2 + 10 \cdot a_3)))$$

und beginnen die Auswertung mit dem innersten Klammerausdruck:

$$\begin{aligned} d &= a_3 \\ d &= 10 \cdot d + a_2 \\ d &= 10 \cdot d + a_1 \\ d &= 10 \cdot d + a_0 \end{aligned}$$

Die wiederkehrenden Zuweisungen werden mit einer `for`-Schleife programmiert:

```
long d;
long a[4];

d = a[3];
for (i = 2; i >= 0; i = i - 1)
  d = 10 * d + a[i];
```

Die Erweiterung für eine allgemeine Anzahl von Stellen ist naheliegend:

```
#define N 20
long d;
long a[N];

d = a[N-1];
for (i = N - 2; i >= 0; i = i - 1)
  d = 10 * d + a[i];
```

Es kann der Fall eintreten, dass die angegebenen Dezimalstellen eine Zahl bilden, die den Wertebereich des Datentyps `long` überschreitet. Dieser Fall soll erkannt werden. In der Headerdatei `limits.h` ist der größte mögliche Wert einer `long` Variablen als `LONG_MAX` definiert.

```
long digit2Dec(long *d, long digits[], long len)
{
  long overflow = 0;
  long i;

  *d = digits[len - 1];
  for (i = len - 2; i >= 0 && !overflow; i = i - 1)
  {
    /* 10 * d + a[i] <= LONG_MAX                      */
    /* daher:   d <= (LONG_MAX - a[i]) / 10         */

    if (*d <= (LONG_MAX - digits[i]) / 10)
       *d = *d * 10 + digits[i];
    else
       overflow = 1;
  }

  return overflow;
}
```

In der `for`-Schleife wurde die zusätzliche Bedingung `!overflow` eingeführt. Wenn also `overflow` auf 1 gesetzt wird, wird die Schleife abgebrochen.

In der `for`-Schleife wird bereits vor der Auswertung des Ausdrucks $d = 10 \cdot d + a_i$ überprüft, ob der Wert innerhalb des erlaubten Bereichs liegen wird. Falls ja, wird der Ausdruck tatsächlich ausgewertet, andernfalls wird der Ausdruck nicht mehr ausgewertet und die Variable `overflow` gesetzt. Damit wird die Schleife verlassen. Da die Funktion im Fall eines Überlaufs 1 zurückliefern soll, kann gleich der Wert von `overflow` zurückgegeben werden.

Das Hauptprogramm

```
#include <stdio.h>
#include <limits.h>

#define MAXDIG 20

/* Funktions Deklarationen */
long readPos();
long dec2digit  (long  d, long digits[], long len);
void printDigits(         long digits[], long len);
long digit2Dec  (long *d, long digits[], long len);

main()
{
  long d = -1;
  long digits[MAXDIG];
  char c;

  do
  {
    /* Eingabe einer positiven ganzen Zahl */
    d = readPos();

    /* Berechnen der Stellen und Ueberpruefung auf Fehler */
    if (dec2digit(d, digits, MAXDIG) == 1)
      printf("Fehler. Es konnten nicht alle Stellen von %ld berechnet werden\n", d);
    else
    {
      printf("Die Dezimalstellen sind: ");
      printDigits(digits, MAXDIG);
    }

    if (digit2Dec(&d, digits, MAXDIG) == 1)
    {
      printf("Fehler: Ueberlauf bei der Berechnung der Zahlen ");
      printDigits(digits, MAXDIG);
    }
    else
      printf("Die Dezimalzahl ist: %ld\n", d);

    printf("Nocheinmal (J/N) ? ");
    c = getchar();
    getchar();
  } while (c == 'J' || c == 'j');
}

/* Funktionsdefinitionen */
```

Beispiel: Städte

Entwickeln Sie ein C-Programm, welches Informationen über Städte verwaltet. Zur Beschreibung einer Stadt (engl. *city*) soll folgender Verbund-Datentyp verwendet werden:

```
struct City_s
{
  char name[30];     // Name
  char country[5];   // Land
  long inhabitants; // Einwohner
};
```

- Legen Sie im Hauptprogramm ein Feld an und initialisieren Sie die ersten 5 Einträge:

```
#define NMAX 20
struct City_s cities[NMAX] =
{ {"Wien",        "A",    1.8e6 },
  {"Graz",        "A",    238e3 },
  {"Berlin",      "D",   3.45e3 },
  {"Zurich",      "Ch",   360e3 },
  {"Kopenhagen", "DK", 1.3e6   }
};
```

- Schreiben Sie eine Funktion, welche die Attribute einer Stadt ausgibt:

```
void printCity(struct City_s *c)
```

  Die Attribute sollen in einer Zeile ausgegeben werden.

- Geben Sie alle gültigen Städte-Einträge aus, indem Sie im Hauptprogramm die Funktion `printCity` in einer `for`-Schleife aufrufen. Ein Eintrag ist dann gültig, wenn die Einwohnerzahl (engl. *inhabitants*) > 0 ist.
- Schreiben Sie eine Funktion, die zu einem gegebenen Namen die Stadt sucht:

```
long selectCity(struct City_s c[], long len, char *name)
```

  Die Funktion soll den Index im übergebenen Feld, hier `c[]`, zurückliefern. Falls die Stadt nicht gefunden wird, soll der Rückgabewert -1 betragen. Lesen Sie im Hauptprogramm den Namen einer Stadt ein und geben Sie entweder die gefundene Stadt oder eine Fehlermeldung aus.

- Schreiben Sie eine Funktion, welche die größte Stadt im Feld sucht:

```
long largestCity(struct City_s c[], long len)
```

  Die Funktion soll den Index im Feld zurückliefern, oder -1, falls das Feld keine gültigen Einträge besitzt.

Erweiterungen

Implementieren Sie im Rahmen eines neuen Projekts ein Menü mit 4 Punkten

(1) print all cities
(2) search city
(3) find largest city
(4) end

und führen Sie den gewählten Menüpunkt aus.

Referenzlösung

Die Funktionen

Zum Ausgeben der Attribute einer Stadt verwenden wir die Funktion `printCity`. Die Funktion liefert keinen Rückgabewert, wird also als `void` vereinbart. Als Parameter wird ein Zeiger auf eine Struktur übergeben.

```
void printCity(struct City_s *c)
{
  printf ("%20s, %2s, %ld Inhabitants\n",
            c->name, c->country, c->inhabitants);
}
```

Bei der Ausgabe der Zeichenketten wird die Ausgabebreite durch `%20s` und `%2s` mit 20 beziehungsweise mit 2 vorgegeben.

Wenn wir im Hauptprogramm an verschiedenen Stellen die Attribute einer Stadt ausgeben, ist es wünschenswert, dass diese Ausgaben gleich aussehen. Deshalb ist es sinnvoll, eine eigene Ausgabefunktion zu verwenden, auch wenn diese, wie in diesem Beispiel, nur aus einem einzigen Aufruf von `printf` besteht.

Um eine Stadt an Hand ihres Namens zu suchen, müssen Zeichenketten verglichen werden (Funktion `strcmp`).

```
long selectCity(struct City_s c[], long len, char *name)
{
  long i;

  for(i = 0; i < len; i++)
  {
    if(c[i].inhabitants > 0 && strcmp(c[i].name, name) == 0)
      return i;
  }
  return -1;
}
```

Es wird ein Feld von Strukturen, die Länge des Feldes, und der Name, nach dem gesucht werden soll, an die Funktion übergeben.

Bei der Suche wird zuerst festgestellt, ob der Eintrag gültig ist (`inhabitants > 0`), und falls dies zutrifft, der Name verglichen. Die Schleife wird verlassen wenn die `if`-Bedingung das erste Mal erfüllt ist. Sollte ein Name im Feld mehrmals vorkommen, wird der erste gefundene Eintrag zurückgeliefert. Falls die Schleife vollständig durchlaufen wurde, also kein vorzeitiger Ausstieg aus der Funktion aufgetreten ist, dann ist die Suche fehlgeschlagen und es wird -1 zurückgegeben.

Bei der Bestimmung der größten Stadt wird nach der größten Einwohnerzahl (Attribut `inhabitants`) gesucht.

```
long largestCity(struct City_s c[], long len)
{
  long i;
  long inhabMax =  0;
  long iMax     = -1;

  for(i = 0; i < len; i ++)
  {
    if (c[i].inhabitants > inhabMax)
    {
      iMax = i;
      inhabMax = c[i].inhabitants;
    }
  }
  return (iMax);
}
```

Das Hauptprogramm

Wir beginnen mit den Header-Dateien. Da wir eine Funktion für Zeichenketten verwenden (`strcmp`), ist die Datei `string.h` zu inkludieren.

Die Länge der Felder wird durch Präprozessor-Konstanten definiert. Anschließend wird ein neuer Datentyp, `struct City_s`, definiert. Es folgen die Funktionsdeklarationen; die Funktionen selbst werden erst nach dem Hauptprogramm definiert.

Die Funktion `clearInputBuffer` (dt. `leereEingabepuffer`) wird verwendet, um den Eingabepuffer im Fall einer fehlerhaften Eingabe zu leeren.

```
// file: referenz5.c

#include <stdio.h>
#include <string.h>

#define LEN 20     // maximum number of cities
#define MAXNAM 20 // length of city names

struct City_s      // define a new data type
{
  char name[MAXNAM];
  char country[5];
  long inhabitants;
};

/* function declarations */
void printCity  (struct City_s *c);
long selectCity (struct City_s c[], long len, char *name);
long largestCity(struct City_s c[], long len);
void clearInputBuffer();
```

```
main()
{
  long choice = 0;
  long i;
  char name[MAXNAM];
  long done = 0;

  struct City_s cit[LEN] =
    { {"Wien",    "A",  1.8e6    },
      {"Graz",    "A",  238e3    },
      {"Berlin", "D",  3.45e6   },
      {"Zurich", "Ch", 360e3    },
      {"Kopenhagen", "DK", 1.3e6}
    };

  do
  { /* the menu */
    printf("\n\n");
    printf("\t(1) Print all cities \n");
    printf("\t(2) search city \n");
    printf("\t(3) find largest city \n");
    printf("\t(4) end\n\n");
    printf("\nEnter your choice: ");

    scanf("%ld", &choice);
    clearInputBuffer();
    printf("\n");

    switch (choice)
    {
       case 1:
         for (i = 0; i < LEN; i ++)
           if (cit[i].inhabitants > 0) printCity(&cit[i]);
         break;
       case 2:
         printf("Enter the name of a city: ");
         scanf("%s", name);
         i = selectCity(cit, LEN, name);
         if (i < 0)
           printf("%s: no such city found\n", name);
         else
           printCity(&cit[i]);
         break;
       case 3:
         i = largestCity(cit, LEN);
         if (i < 0)
           printf("no cities in the data base\n");
         else
         {
           printf("The largest city found:\n\n");
           printCity(&cit[i]);
         }
         break;
```

```
        case 4:
           done = 1;
           break;
        default:
          printf("Invalid choice\n");
     }
   }
  while(!done);
}

void clearInputBuffer()
{
  while (getchar() != '\n');
}
```

Beispiel: Entfernen von C-Kommentaren

Entwickeln Sie ein C-Programm, das alle Mehrzeilen-Kommentare aus einem C-Quelltext entfernt. Mehrzeilen-Kommentare werden mit `/*` eingeleitet und mit `*/` abgeschlossen.

- Der Name der C-Datei soll vom Benutzer eingegeben werden.
- Das Programm ohne Kommentare soll in einer Datei gespeichert werden. Der Name dieser Datei ist aus dem ursprünglichen Dateinamen durch Anhängen von `"-nc"` zu bilden (Abkürzung für *no comment*, also für die kommentarlose Datei). Zum Beispiel soll der Eingabe-Datei `"meinProg.c"` die Ausgabe-Datei `"meinProg.c-nc"` zugeordnet werden.
- Öffnen Sie die Ein- und Ausgabe-Dateien, entfernen Sie die Kommentare aus dem C-Quelltext und speichern Sie den modifizierten C-Quelltext in der Ausgabe-Datei.

Erweiterungen

- Entfernen Sie auch die Einzeilen-Kommentare, also jene Kommentare, die mit `//` eingeleitet und mit dem Zeilenende abgeschlossen werden.

Referenzlösung

Das Beispiel wird im Hauptprogramm gelöst. Es werden keine Funktionen definiert.

Namen der Dateien und Öffnen der Dateien

Im ersten Teil des Hauptprogramms wird der Name der C-Quelltext-Datei eingelesen und der Name der Ausgabe-Datei erzeugt.

```
#include <stdio.h>
#include <string.h>

#define LEN 20

int main()
{
  FILE *infilep;   // file handle for input file
  FILE *outfilep; // file handle for output file

  char ch;
  char infile [LEN+1];   // file name of input file
  char outfile[LEN+4];   // file name of output file

  long state = 0;
  long i;

  /* read name of input file */
  printf("enter a file name: ");
  scanf("%20s", infile);    // read not more than 20 characters using (%20s)

  /* assemble name of output file */
  strcpy(outfile, infile);
  strcat(outfile, "-nc");

  /* open input and output file */
  infilep  = fopen(infile, "r");
  if(infilep == 0)
  {
    fprintf(stderr, "cannot open %s\n", infile);
    return 1;
  }

  outfilep = fopen(outfile, "w");
  if(outfilep == 0)
  {
    fprintf(stderr, "cannot open %s\n", outfile);
    fclose(infilep);
    return 1;
  }
  printf("writing output file \"%s\"\n", outfile);
```

Am Beginn werden die beiden Datenströme `infilep` und `outfilep` vereinbart. Der Name der Eingabe-Datei darf aus maximal `LEN` Zeichen bestehen. Da eine Zeichenkette mit dem Null-Byte abgeschlossen wird, vereinbaren wir `infile` mit der Länge `LEN+1`. Beim Namen der Ausgabe-Datei werden noch die 3 Zeichen `-nc` angefügt, weshalb die Maximal-Länge dieser Zeichenkette mit `LEN+4` vereinbart wird.

Beim Einlesen des Dateinamens mit `scanf` wird durch die Formatangabe `%20s` sichergestellt, dass nicht mehr als 20 Zeichen gelesen werden und somit die Feldgrenze von `infile` nicht überschritten werden kann.

Anschließend wird der Name der Ausgabe-Datei ermittelt. Es wird der Name der Eingabe-Datei auf `outfile` kopiert (Funktion `strcpy`) und die Zeichenkette `"-nc"` angehängt (Funktion `strcat`).

Mit `fopen` werden die beiden Datenströme geöffnet. Falls beim Öffnen ein Fehler auftritt, wird das Hauptprogramm mit dem Rückgabewert `1` beendet.

Entfernen der Kommentare

Im zweiten Teil des Hauptprogramms wird die Eingabe-Datei Zeichen für Zeichen gelesen (Funktion `getc`) und nach den Zeichen-Kombinationen `/*` und `*/` untersucht.

```
    ch = getc(infilep);  // read 1st character from file

    while(ch != EOF)     // check for end-of-file
    {
      if(state == 0)
      {
        /* state == 0:  outside the comment */
        if(ch == '/')
        {
           ch = getc(infilep);     // look ahead for next character
           if(ch == EOF) break;
           if(ch == '*')           // look for beginning of comment
             state = 1;            // new state: inside the comment
           else
           {
              putc('/', outfilep);
              putc(ch,  outfilep);
           }
        }
        else
          putc(ch, outfilep);
      }
      else
      {
        /* state == 1: inside the comment */
        if(ch == '*')
        {
           ch = getc(infilep);    // look ahead for next character
           if(ch == EOF) break;
           if(ch == '/')           // look for end of comment
             state = 0;            // new state: outside the comment
        }
      }
      ch = getc(infilep);
    }

    /* close input and output file */
    fclose(infilep);
    fclose(outfilep);

    return 0;
  }
```

In der `while`-Schleife wird zwischen zwei Zuständen (engl. *states*) hin- und hergeschaltet. Der Zustand `0` bedeutet, dass Zeichen *außerhalb* eines Kommentars gelesen werden, während im Zustand `1` Zeichen *innerhalb* eines Kommentars gelesen werden.

Die `while`-Schleife wird verlassen, falls `getc` den Wert `EOF` zurückliefert, wenn also das Datei-Ende erreicht ist.

Nach der `while`-Schleife werden die Datenströme geschlossen (`fclose`) und die fehlerfreie Beendigung des Hauptprogramms mit dem Rückgabewert 0 angezeigt.

Erweiterung

Das Hauptprogramm:

```
// file:  referenz6e.c

#include <stdio.h>
#include <string.h>

#define LEN 20

int main(int argc, char *argv[])
{
  FILE *infilep;  // file handle for input file
  FILE *outfilep; // file handle for output file

  char ch;
  char infile [LEN+1];  // file name of input file
  char outfile[LEN+4];  // file name of output file

  long i;
  long state  = 0;
  int  retval = 0; // return value of main

  if(argc == 1)
  {
    fprintf(stderr, "no file name given\n");
    return 1;
  }

  for(i = 1; i < argc; i++) // loop over all arguments of main
  {
    if(strlen(argv[i]) > LEN)
    {
      fprintf(stderr, "file \"%s\" ignored, name too long\n", argv[i]);
      retval = 1;
      continue;     // continue with next file name
    }

    /* name of input file */
    strcpy(infile, argv[i]);

    /* assemble name of output file */
    strcpy(outfile, infile);
    strcat(outfile, "-nc");
```

```
/* open input and output file */
infilep  = fopen(infile, "r");
if(infilep == 0)
{
  fprintf(stderr, "cannot open %s\n", infile);
  retval = 1;
  continue;
}

outfilep = fopen(outfile, "w");
if(outfilep == 0)
{
  fprintf(stderr, "cannot open %s\n", outfile);
  fclose(infilep);
  retval = 1;
  continue;
}

printf("writing output file \"%s\"\n", outfile);

ch = getc(infilep);  // read 1st character from file

while(ch != EOF)     // check for end-of-file
{
  if(state == 0)
  {
    /* state == 0:  outside the comment */
    if(ch == '/')
    {
       ch = getc(infilep);     // look ahead for next character

       if(ch == EOF) break;

       if(ch == '*')           // beginning of multi-line comment
         state = 1;            // new state: inside multi-line comment
       else if (ch == '/')     // beginning of one-line comment
         state = 2;            // new state: inside one-line comment
       else
       {
          putc('/', outfilep);
          putc(ch,  outfilep);
       }
    }
    else
      putc(ch, outfilep);
  }
  else if(state == 1)
  {
    /* state == 1: inside multi-line comment */
    if(ch == '*')
    {
       ch = getc(infilep);    // look ahead for next character
```

```
          if(ch == EOF) break;

          if(ch == '/')            // look for end of comment
            state = 0;             // new state: outside the comment
        }
      }
      else
      {
        /* state == 2: inside one-line comment */
        if(ch == '\n')             // look for end of one-line comment
        {
          state = 0;               // new state: outside the comment
          putc(ch, outfilep);
        }
      }
      ch = getc(infilep);
    }
    /* close input and output file */
    fclose(infilep);
    fclose(outfilep);
  }
  return retval;
}
```

Die Namen der Eingabe-Dateien werden als Argumente an die Funktion `main` übergeben. Zuerst wird der Fall überprüft, dass kein Dateiname angegeben wird. In diesem Fall wird eine Fehlermeldung auf den Standard-Fehlerstrom geschrieben und das Hauptprogramm verlassen (`return 1`).

In der folgenden `for`-Schleife wird über die Dateinamen iteriert. Da wir die Dateinamen als Felder mit fester Länge vereinbart haben, prüfen wir die Längen der übergebenen Zeichenketten `argv[i]`. Damit wird sichergestellt, dass bei den folgenden Aufrufen von `strcpy` und `strcat` die Feldgrenzen nicht überschritten werden.

Falls ein Fehler erkannt wird, wird der Rest des Schleifenblocks nicht mehr ausgeführt, sondern mit dem nächsten Dateinamen fortgesetzt (`continue` Anweisung).

In der `while`-Schleife werden nun drei Zustände unterschieden:

`state = 0`	außerhalb eines Kommentars,
`state = 1`	innerhalb eines Mehrzeilen-Kommentars,
`state = 2`	innerhalb eines Einzeilen-Kommentars.

Die Variable `retval` wird im Fall eines Fehlers auf `1` gesetzt, andernfalls behält sie den Initialwert `0`. Der Wert von `retval` wird am Ende von `main` als Rückgabewert verwendet.

Softwareprojekt: Studentenkartei

Entwickeln Sie ein C-Programm, welches die Studentenkartei eines Instituts verwaltet.

Das Programm soll aus dem Hauptprogramm `main.c`, dem Modul `student.c` und der zugehörigen Header-Datei `student.h` bestehen. Die Datei `main.c` soll nur die Funktion `main` enthalten, alle anderen Funktionen sollen in der Datei `student.c` implementiert werden.

Die Header-Datei `student.h` besteht aus allen notwendigen Präprozessoranweisungen sowie Definitionen von Verbund-Datentypen und Funktionsdeklarationen, die Sie zur Erstellung des Programms benötigen.

```
/* Datei: student.h */

#define NMAX   10   /* Maximale Datenbankeintraege */
#define EMAX   10   /* Maximale exam Elemente */
#define SMAX   50   /* Maximale Anzahl von Zeichen  */
#define FALSE  0
#define TRUE   1

typedef struct Exam_s
{
  long points;       /* exam Punkte */
  long mark;         /* exam Note */
} Exam_t;

typedef struct Student_s
{
  char       firstname[SMAX];  /* Vorname     */
  char       lastname[SMAX];   /* Familienname */
  long       id;               /* Matrikelnummer */
  long       exams;            /* Anzahl der Pruefungen */
  Exam_t     exam[EMAX];       /* Pruefungs-Daten */
} Student_t;

/* Funktionsdeklarationen */
void initializeAll        (Student_t st[], long len);
long readBinaryStudentData(Student_t st[], long len, char *fileName);
long saveBinaryStudentData(Student_t st[], long len, char *fileName);
void editStudent          (Student_t st[], long position);
void printStudent         (Student_t st[], long position);
void clearInputBuffer     (void);
```

Referenzlösung

In dieser Übung erstellen Sie ein komplettes Softwareprojekt. Für bereits besprochene Teile dieses Projektes schlagen Sie bitte in vorangegangenen Referenzbeispielen nach.

Die Funktionen

Um alle Datensätze korrekt zu initialisieren, ist folgende Funktion in `student.c` zu implementieren:

```
void initializeAll(Student_t st[], long len)
{
   long outer_loop, inner_loop;

   for (outer_loop = 0; outer_loop < len; outer_loop++)
   {
     st[outer_loop].firstname[0]='\0';
     st[outer_loop].lastname[0]='\0';
     st[outer_loop].id=0;
     st[outer_loop].exams=0;
     for (inner_loop = 0; inner_loop < EMAX; inner_loop++)
     {
       st[outer_loop].exam[inner_loop].points=0;
       st[outer_loop].exam[inner_loop].mark=0;
     }
   }
}
```

Um das gesamte Datenbankfeld einzulesen oder in eine Datei zu speichern, implementieren Sie folgende Funktionen in `student.c`:

```
long readBinaryStudentData(Student_t st[], long len, char *fileName);
long saveBinaryStudentData(Student_t st[], long len, char *fileName);
```

Diese Funktionen sollen mittels den C-Bibliotheksfunktionen `fread()` und `fwrite()` das Datenbankfeld aus der angegebenen Datei laden oder schreiben.

```
long readBinaryStudentData(Student_t  st[], long len, char *fileName)
{
   FILE * file_p;
   long number;

   file_p = fopen(fileName, "r");
   if (file_p == NULL)
   {
      fprintf(stderr, "Error opening file %s\n", fileName);
      return 1;
   }

   number = fread(st, sizeof(Student_t), len, file_p);
   fclose(file_p);

   return number;
}
```

Die zugehörige Funktion `saveBinaryStudentData()` ist analog zu implementieren.

Das Hauptprogramm

Das Hauptprogramm wird in `main.c` implementiert. Ein Student kann an diesem Institut bis zu 10 Prüfungen machen. Für jede Prüfung wird die Punkte-Anzahl (*points*) und die Note (*mark*) gespeichert.

Beachten Sie bitte die jeweiligen Fehlerabfragemechanismen.

```
main()
{
   Student_t datenbank[NMAX];

   long choice = 0, choice2 = 0;
   long done = 0;
   char filename[SMAX];

   do
   {
     printf("\n");
     printf("\t(1) Load student data \n");
     printf("\t(2) Save student data \n");
     printf("\t(3) Edit student data \n");
     printf("\t(4) Print student data \n");
     printf("\t(5) Initialize all \n\n");

     printf("\t(0) End\n\n");
     printf("\tEnter your choice: ");

     scanf("%ld", &choice);
     clearInputBuffer();

     switch (choice)
     {
        case 1:
          printf("\n   Please enter filename: ");
          scanf("%s", filename);
          clearInputBuffer();

          if (readBinaryStudentData(datenbank, NMAX, filename) != NMAX)
          {
            fprintf(stderr, "Error reading all data sets. Exit!\n");
            return -1;
          }
          break;

        case 2:
          printf("\n   Please enter filename: ");
          scanf("%s", filename);
          clearInputBuffer();
```

```
            if (saveBinaryStudentData(datenbank,NMAX,filename) != NMAX)
            {
              fprintf(stderr,"Error writing all data sets. Exit!\n");
              return -1;
            }

            break;

          case 3:
            printf("\n   Please enter position: ");
            scanf("%ld", &choice2);
            clearInputBuffer();

            editStudentData(datenbank, choice2);
            break;

          case 4:
            printf("\n   Please enter position: ");
            scanf("%ld", &choice2);
            clearInputBuffer();

            printStudentData(datenbank,choice2);
            break;

          case 5:
            initializeAll(datenbank, NMAX);
            break;


          case 0:
             done = 1;
             break;

          default:
            fprintf(stderr,"Invalid choice\n");
        }
      }
    while(!done);

  return 0;
}
```