P5 A2

Aufgabe 1: Funktionszeiger Schreiben Sie ein C-Programm zur Berechnung des Wertes eines bestimmten Integrals einer Funktion f(x) in den Grenzen a, b nach der SIMPSON-Regel!

Die SIMPSON-Regel für ein geradzahliges n (n>=2) lautet:

I= h/3 * (f(x0) + 4* f(x1) + 2* f(x2) + … +4* f(xn-3) + 2* f(xn-2) +4*f(xn-1) + f(xn))

mit: xi = a + i* h (i läuft von 0 bis n), h = (b – a)/n Die SIMPSON-Integrationsregel soll als C-Funktion simpson definiert werden. Eingangsparameter sind die Integrationsgrenzen a ,b , die Anzahl der Stützstellen n und die Integrantenfunktion f(x) als Funktionszeiger! Als Rückkehrwert ist der Integralwert vorzusehen. Testen Sie Ihr Programm an solchen Funktionen, bei denen man die bestimmten Integrale kennt!

```c
#include <stdio.h>
double meinefkt(double x)
{
    x;
    return 1.0;
}
double simpson(double(*fzeiger)(double), int n, double a, double b)
{
    double h, integral;
    int i;
    double faktor;


    if (n < 2 || n % 2 != 0)
    {
        printf("Parameterfehler");
    }
    h = (b - a) / (double)n;
    integral = 0.0;

    integral += (*fzeiger)(a);

    for (i = 0; i < n; i++){
        if (i % 2 == 0)faktor = 4.0;
        else faktor = 2.0;
        integral = integral + faktor*(*fzeiger)(a + i*h);
    }
    integral += (*fzeiger)(b);
    integral = integral*h / 3.0;
    return integral;
}
int main()
{
    double integralwert;


    integralwert = simpson(meinefkt, 100, 0, 5);

    printf("Integralwert:%lf", integralwert);


    getchar();
    return 1;
}
```

P11_A1

Schreiben Sie ein Programm, das aufsteigend sortierte Daten aus zwei Textdateien einliest, gemäß ihrer Sortierreihenfolge mischt und die Daten auf der Konsole und in eine dritte Textdatei ausgibt. Die beiden Textdateien zur Eingabe enthalten zeilenweise Angaben zu Marathonläufern mit deren erreichten Zeiten. Die Angaben sind in den Dateien bereits gemäß aufsteigender Zeiten sortiert. Jede Textdatei soll maximal 1000 Datensätze beinhalten. Das Format der Textdateien ist im folgenden Beispiel beschrieben: Max Schulze 3:21:54 Fritz Fuchs 3:23:03 Antje Schnellfuß 3:24:30 Alfred Drescher 3:26:50

Hinweise : □ Laden Sie sich die beiden bereitgestellten Eingabedateien laeufer1.txt und laeufer2.txt auf C:\TEMP\ herunter! □ Deklarieren eine Struktur (struct) zur Aufnahme der Daten zu einem Läufer, d.h. einer Zeile. Sie benötigen mehrere Variablen, bzw. mehrere Felder aus diesem Strukturtyp, je nach Ihrer Lösungsvariante. □ Zum Lesen der Daten aus einer Datei müssen Sie zuerst die jeweilige Datei öffnen, z.B. wie folgt: FILE *f1; f1 = fopen("C:\\TEMP\\laeufer1.txt","rt"); Eine einzelne Zeile können Sie mit der Funktion fscanf() wie folgt einlesen: fscanf(f1,"%s %s %d:%d:%d", vorname, name, &h,&m,&s); mit vorname, name als char-Feldern und h,m,s als Integer-Variablen

□ Nach dem vollständigen Lesen kann die Datei mit fclose(f1) geschlossen werden. □ Wenn die Entwicklungsumgebung die Verwendung s.g. Secure-Functions fopen_s(), fscanf_s() verlangt, dann kann durch die folgende erste Zeile im Programm die Verwendung der gewöhnlichen Funktionen erlaubt werden: #define _CRT_SECURE_NO_WARNINGS □ Als Ergebnis muss das Mischen die Daten aus den beiden Textdateien in einer nach Zeiten aufsteigend sortierten Folge bereitstellen.

```
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#define MAX 1000

struct laeufer {
	char vname[20], name[20];
	int h, m, s;
};
int einlesen(char *filename, struct laeufer *feld, int *anzahl)
{
	//da wir zweimal einlesen, wird das Einlesen als Funktion realisiert
	// Eingelesen wird auf ein Feld fester Länge das in der main Funktion
deklariert wurde

	FILE *f1 = fopen(filename, "rt");
	int n;
	struct laeufer lf;
	*anzahl = 0;
	if (f1)
		printf("Eingabedatei %s erfolgreich geoeffnet ... \n", filename);
	else
		return -1;

	while (!feof(f1))
	{
		n = fscanf(f1, "%s %s %d:%d:%d", lf.vname, lf.name, &lf.h, &lf.m,
&lf.s);
		if (n == 5)
		{
```

```
                    printf("Gelesen: %s %s %d:%d:%d \n", lf.vname, lf.name, lf.h,
lf.m, lf.s);

                    feld[*anzahl] = lf;
                    (*anzahl)++;
             }
             else
             {
                    //Fehlerhafte Zeile oder Dateiende
             }
      }

}
/*
void mischen(struct laeufer l1, int anz1, struct laeufer l2, int anz2, struct
laeufer l3, int *anz3)
{
//Vergleich in Funktion int ist_schneller( struct laeufer l1, struct laeufer
l2){...

if (ist_schneller(l1[i], l2[j]))
{
l3[k] = l1[i];
i++;
}
else
{
l3[k] = l2[j];
j++;
}


}

*/

int main()
{
      //FILE *f1, *f2;
      int n;
      int i, n1, n2, anzahl1, anzahl2, anzahl3;
      struct laeufer lf;
      struct laeufer laeufer1[MAX];
      struct laeufer laeufer2[MAX];
      struct laeufer laeufer3[MAX];

/*
// #ifdef - #endif inaktiver Präprozessorblock
#ifdef ALT

      f1 = fopen("C:\Users\Anxhela\Desktop\laeufer1.txt", "rt");
      if (f1)
             printf("Eingabedatei laeufer1 erfolgreich geoeffnet... \n\n");
      else
      {
             printf("Eingabedatei laeufer1.txt nicht gefunden... \n\n");
             return -1;
      }


      f2 = fopen("C:\Users\Anxhela\Desktop\laeufer2.txt", "rt");
```

```
	if (f2)
		printf("Eingabedatei laeufer2 erfolgreich geoeffnet... \n\n");
	else
	{
		printf("Eingabedatei laeufer2.txt nicht gefunden... \n\n");
		return -1;
	}

	while (!feof(f1))
	{
		n = fscanf(f1, "%s %s %d:%d:%d", lf.vname, lf.name, &lf.h, &lf.m,
&lf.s);
		if (n == 5)
		{
			printf("Gelesen: %s %s %d:%d:%d \n", lf.vname, lf.name, lf.h,
lf.m, lf.s);
		}
		else
		{
			//Fehlerhafte Zeile oder Dateiende
		}
	}

	printf("\n");

	while (!feof(f2))
	{
		n = fscanf(f2, "%s %s %d:%d:%d", lf.vname, lf.name, &lf.h, &lf.m,
&lf.s);
		if (n == 5)
		{
			printf("Gelesen: %s %s %d:%d:%d \n", lf.vname, lf.name, lf.h,
lf.m, lf.s);
		}
		else
		{
			//Fehlerhafte Zeile oder Dateiende
		}
	}


	fclose(f1);
	fclose(f2);


#endif

*/

	n1 = einlesen("C:\Users\Anxhela\Desktop\laeufer1.txt", laeufer1, &anzahl1);
	n2 = einlesen("C:\Users\Anxhela\Desktop\laeufer2.txt", laeufer2, &anzahl2);


	if (n1 == -1 || n2 == -1)
	{
		printf("Eingabedateien konnten nicht geoeffnet werden, Abbruch \n");

		getchar();
		return -1;
	}
	printf("Anzahl: %d, Anzahl2: %d\n", anzahl1, anzahl2);
```

```
	//Ausgabe der Laeufer aus den Arrays

	printf("Laeufer 1:\n");
	for (i = 0; i < anzahl1; i++)
		printf("%s \t %s \t %02d:%02d:%02d\n", laeufer1[i].vname,
laeufer1[i].name,
			laeufer1[i].h, laeufer1[i].m, laeufer1[i].s);

	printf("Laeufer 2:\n");
	for (i = 0; i < anzahl2; i++)
		printf("%s \t %s \t %02d:%02d:%02d\n", laeufer2[i].vname,
laeufer2[i].name,
			laeufer2[i].h, laeufer2[i].m, laeufer2[i].s);

	//jetzt können wir die Arrays in ein drittes Array mischen


	/*
	mischen(laeufer1, &anzahl1, laeufer2, &anzahl2, laeufer3, &anzahl3);

	printf("Rangfolge der Laeufer nach dem Mischen:\n");


	printf("Laeufer 2:\n");
	for (i = 0; i < anzahl3; i++)
	printf("%s \t %s \t %02d:%02d:%02d\n", laeufer3[i].vname, laeufer3[i].name,
	laeufer3[i].h, laeufer3[i].m, laeufer3[i].s);
	*/

	getchar();
	return 1;
}
```

P12_A1

Schreiben Sie ein Programm, das Messzeiten und Messwerte von einer Binärdatei einliest, diese klassifiziert, intern speichert und weiterverarbeitet. Zur Speicherung soll dynamisch allokierter Speicher benutzt werden. Die vorgegebene Binärdatei (mreihen.bin) enthält nacheinander gespeicherte Datensätze die jeweils eine Kanalnummer und eine Zeitangabe (in Millisekunden, ausgehend vom Startzeitpunkt der Messung) und einen Messwert beinhalten.

Die Wertebereiche und C-Datentypen sind wie folgt festgelegt:

Element	Benutzer Wertebereich	Binärformat gemäß C-Datentyp
Kanalnummer	0 bis 255	unsigned char
Zeitangabe	0 bis 10000	unsigned int
Messwert	-1000.00 bis +1000.00	float

Es ist nicht vorab bekannt, wie viele Datensätze in der Eingabedatei bereitgestellt werden. Es ist lediglich bekannt, dass in einer Datei Messungen aus maximal drei unterschiedlichen Kanälen enthalten sind. Welche Kanalnummern genutzt werden ist ebenfalls nicht vorab bekannt. Sie sollen die Messwerte nach Kanälen getrennt fortlaufend in dynamisch angeforderten Speicherbereichen ablegen und auf der Konsole ausgeben! Hinweise zu den Teilschritten: Deklarieren Sie einen Struktur-Typ und Variablen zur Aufnahme der Datensätze! Lesen Sie die Datensätze mit der Funktion fread() schrittweise auf eine Strukturvariable! Die Daten werden bereits im internen Darstellungsformat auf die Strukturvariabe gelesen. Bestimmen Sie, wie viele unterschiedliche Kanäle, welche Kanalnummern und wie viele Messungen je Kanal vorhanden sind! Dabei werden alle Datensätze der Datei in einem s.g. ersten Pass gelesen, die Werte aber noch nicht gespeichert. Allokieren Sie für die vorhandenen (maximal drei) Kanäle Speicherplatz, damit die Messungen aufgenommen werden können! Nutzen Sie dazu die Funktion malloc() ! Lesen Sie die Datei in einem zweiten Pass noch einmal und speichern Sie die Werte in den nun in den für die Kanäle bereitgestellten Speicherbereichen! Zur Ergebniskontrolle können Sie die Anzahl und Nummern der Kanäle auf der Konsole ausgeben und für jeden Kanal die Messzeiten und Messwerte anzeigen lassen. Die Datei enthält eine relativ kleine Menge Datensätze, so dass die Ausgabe auf der Konsole kurz ist.

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

#define DATEINAME "C:\\Users\\Anxhela\\Desktop\\mreihen.bin"

struct mess {
	unsigned char kanal;
	unsigned int zeit;
	float wert;
};

void registriere_messung(struct mess e, int n_kanaele, int benutzte_knummern[],
int anz_je_kanal[])
{


	int i;
	for (i = 0; i < n_kanaele; i++)
	{
```

```
            if (anz_je_kanal[i] == 0)  //Achtung, das funktioniert nur für das
erste Feldenelement
            {

                  anz_je_kanal[i] = 1;
                  benutzte_knummern[i] = e.kanal;
                  break;
            }

            else
            {
                  if (benutzte_knummern[i] == e.kanal)
                  {
                        anz_je_kanal[i]++;
                        break;
                  }
            }
      }
}
int main()
{

      FILE *f;
      int n;
      struct mess element;
      int benutzte_kanalnummern[3];
      int werte_je_kanal[3] = { 0, 0, 0 };


      struct mess *k0werte, *k1werte, *k2werte;

      f = fopen(DATEINAME, "rb");

      while (!feof(f))

      {
            n = fread(&element, sizeof(element), 1, f);
            if (n == 1)
            {
                  //hier schauen, welche Daten in dem element stecken

                  printf("Gelesen:   kanal=%d    zeit=%d    wert=%f\n",
element.kanal, element.zeit, element.wert);

                  //hier soll das Programm die Anzahl der benutzten kanäle,
                  //Kannalnummern und die Nazahl der Werte je Kanal
                  //herausbekommen
                  //                        Max. 3 Kanäle, Array  int[3]
Array int[3]
                  registriere_messung(element, 3, benutzte_kanalnummern,
werte_je_kanal);
            }
      }
      //fclose(f);


      if (werte_je_kanal[0])
            printf("Kanal:%d  Anzahl:%d\n", benutzte_kanalnummern[0],
werte_je_kanal[0]);
      if (werte_je_kanal[1])
```

```
		printf("Kanal:%d  Anzahl:%d\n", benutzte_kanalnummern[1],
werte_je_kanal[1]);
	if (werte_je_kanal[2])
		printf("Kanal:%d  Anzahl:%d\n", benutzte_kanalnummern[2],
werte_je_kanal[2]);



	//nun Speicher für die verschiedene Kanäle allockieren
	//ober deklarieren struct mess *k0werte;
	k0werte = (struct mess*) malloc(werte_je_kanal[0] * sizeof(struct mess));
//drei zeiger für die Anfangsadresse der dynamischen Array int[3]
	k1werte = (struct mess*) malloc(werte_je_kanal[1] * sizeof(struct mess));
	k2werte = (struct mess*) malloc(werte_je_kanal[2] * sizeof(struct mess));

	//statt k0werte, k1werte kann auch ein Array von Zeigern benutzt werden
kwerte[i], i=0..2
	//Datei nochmal lesen und Werte auf Arrays verteilen

	rewind(f); //Achtg. dazu muss fclose an das Ende verschoben bwerden


	int idx_0 = 0;
	while (!feof(f))
	{
		n = fread(&element, sizeof(struct mess), 1, f);
		if (n == 1)
		{
			if (element.kanal == benutzte_kanalnummern[0])
			{

				k0werte[idx_0] = element;
				idx_0++;
			}
			//das glaiche für die anderen Kanäle

		}
	}

	fclose(f);

	getchar();
	return 1;
}
```

P13_A1

Erweitern Sie ein gegebenes Programm (prak13_vorlage.c), das vorbereitet wurde für

- das Entgegennehmen von Ortsnamen (char-Feld mit 40 Zeichen) und Temperaturen (float) von der Konsole und für das Speichern dieser Einträge
- für das Löschen eines Eintrags, der durch seinen Ort angegeben wird.
- und für das Auflisten aller Orte mit deren Temperaturen

Die Einträge sollen in einer linearen Liste gespeichert werden. Dazu wird für jeden Eintrag ein dynamisch allokierter Speicherbereich verwendet, der einen Zeiger auf den nächsten Eintrag enthält. Programmieren Sie schrittweise die folgenden Operationen und Varianten: a) Einfügen neuer Einträge am Ende der Liste. Die Reihenfolge der Elemente ergibt sich dann aus der Reihenfolge bei der Eingabe. Dazu ist die Funktion einfuegen() zu erweitern. b) Das Löschen von Einträgen. Dazu ist die Funktion loeschen() zu erweitern. c) Das Anzeigen aller Einträge indem die lineare Liste durchlaufen wird. Dazu können Sie die vorbereitete Funktion anzeigen() erweitern. Testen Sie schrittweise die Funktion Ihres Programms.

Zusatzaufgaben: Implementieren Sie weitere Varianten des Einfügens in der vorbereiteten Funktion einfuegen_sort(): d) Einfügen an einer Position, dass die Listenelemente nach aufsteigenden Temperaturen angeordnet werden. e) Einfügen gemäß einer lexikografischen Ordnung der Ortsnamen. Die Ortsnamen können mit der Funktion strcmp() aus der C-Standardbibliothek verglichen werden.

EINFÜGEN:

```
/* Funktion einfuegen() */
/* Erzeugt eine neues Listenelement und kettet es ein */
/* Rückgabe 1 wenn erfolgreich, sonst 0 */
int einfuegen(element_t **liste, char *ort, float t)
{
  element_t *neu, *akt;
  neu = (element_t*)malloc(sizeof (element_t));
  if (!neu)
    return 0;

  neu->temp = t;
  strcpy(neu->ort, ort);
  neu->next = NULL; //wir fügen das Element am Ende ein, damit hat es keinen Nachfolger


  //jetzt einketten
  if (*liste == NULL)
  {
    *liste = neu;  //hier sieht man, dass liste per Call-by-Ref übergeben werden musste
  }
  else
  {
    akt = *liste;
    while (akt->next != NULL) //'Vorwärts hangeln' bis zum letzten Element
```

```
        akt = akt->next;          //Damit ist neues Element als Nachfolger von bislang
                                  //letzten Element eigekettet
   }

   return 1;  //hier signalisieren, dass ein Element eingefügt wurde
}
```

ANZEIGEN:

```
/* Funktion anzeigen() */
/* Durchlaufen aller Elemente und Ausgabe */
void anzeigen(element_t *liste)
{
   element_t *akt; //Hilfszeiger zum Ablaufen aller elemente

   akt = liste;
   while (akt != NULL)
   {
      //Ausgabe
      printf("%s %f\n", akt->ort, akt->temp);
      akt = akt->next;
   }
}
```

LÖSCHEN:

```
int loeschen(element_t **liste, char *ort)
{
   element_t *akt, *vorg;
   akt = *liste;
   vorg = NULL;    //Vorgänger von akt

   while (akt)
   {
      if (strcmp(ort, akt->ort) == 0)
      {
         //element akt ausketten und freigeben
         if (vorg) //dann zeigt akt nicht auf das erste Element
         {
            vorg->next = akt->next;  //'Bypass'-Zeiger um akt zu überspringen
            free(akt);
         }
         else   //dann hat akt keinen Vorgänger
         {
            *liste = akt->next;  //hier ändert sich listenanker, deshalb wurde er per Call-by-Fer
übergeben
            free(akt);
         }
         return 1;
      }
      vorg = akt;
```

```
        akt = akt->next;

    }

    return 0;
}
```

DAS GANZE PROGRAMM

```
/* prak13_vorlage.cpp */
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define EINGABE_LEN 60
#define LEN 40


// ------------------------------------------------------------------------
// Ihre Aufgabe:
// Erweitern Sie die Funktionen einfuegen(), loeschen() und anzeigen()
// Benutzen Sie eine lineare Liste zur Speicherung der Daten
// ------------------------------------------------------------------------


typedef struct element {
   char ort[LEN];
   float temp;
   struct element *next;
} element_t;

void eingabe_anpassen(char *k)
{
   char *newlinec;

   while (k[0] == ' ')
      strcpy(k, &k[1]);

   newlinec = strchr(k, '\n');

   if (newlinec)
      *newlinec = '\0';
}

void kommando_anpassen(char *k)
```

```
{
   unsigned int i;

   eingabe_anpassen(k);

   for (i = 0; i<strlen(k); i++)
      k[i] = tolower(k[i]);

}

/* Funktion einfuegen() */
/* Erzeugt eine neues Listenelement und kettet es ein */
/* Rückgabe 1 wenn erfolgreich, sonst 0 */
int einfuegen(element_t **liste, char *ort, float t)
{
   element_t *neu, *akt;
   neu = (element_t*)malloc(sizeof (element_t));
   if (!neu)
      return 0;

   neu->temp = t;
   strcpy(neu->ort, ort);
   neu->next = NULL; //wir fügen das Element am Ende ein, damit hat es keinen Nachfolger


   //jetzt einketten
   if (*liste == NULL)
   {
      *liste = neu;  //hier sieht man, dass liste per Call-by-Ref übergeben werden musste
   }
   else
   {
      akt = *liste;
      while (akt->next != NULL) //'Vorwärts hangeln' bis zum letzten Element
         akt = akt->next;


      akt->next = neu;                        //Damit ist neues Element als Nachfolger von bislang
                              //letzten Element eigekettet
   }

   return 1;  //hier signalisieren, dass ein Element eingefügt wurde
}

/* Funktion einfuegen() */
/* Erzeugt eine neues Listenelement und kettet es */
```

```
/* aufsteigend sortiert ein */
/* Rückgabe 1 wenn erfolgreich, sonst 0 */
int einfuegen_sort(element_t **liste, char *ort, float t)
{

   return 0;
}

/* Funktion loeschen() */
/* Ausketten eines Elements und Freigabe */
/* Rückgabe: 1 wenn Eintrag gefunden und geloescht, sonst 0  */
int loeschen(element_t **liste, char *ort)
{
   element_t *akt, *vorg;
   akt = *liste;
   vorg = NULL;    //Vorgänger von akt

   while (akt)
   {
      if (strcmp(ort, akt->ort) == 0) return akt;
      {
         //element akt ausketten und freigeben
         if (vorg) //dann zeigt akt nicht auf das erste Element
         {
            vorg->next = akt->next;  //'Bypass'-Zeiger um akt zu überspringen
            free(akt);
         }
         else   //dann hat akt keinen Vorgänger
         {
            *liste = akt->next;  //hier ändert sich listenanker, deshalb wurde er per Call-by-Fer
übergeben
            free(akt);
         }
         return 1;
      }
      vorg = akt;
      akt = akt->next;

   }

   return 0;
}

/* Funktion anzeigen() */
/* Durchlaufen aller Elemente und Ausgabe */
void anzeigen(element_t *liste)
```

```
{
   element_t *akt; //Hilfszeiger zum Ablaufen aller elemente

   akt = liste;
   while (akt != NULL)
   {
      //Ausgabe
      printf("%s %f\n", akt->ort, akt->temp);
      akt = akt->next;
   }
}

int main()
{
   char kommando[EINGABE_LEN];
   char temperatur_s[LEN];
   char ort[LEN];
   float t;
   int n;

   // Zeiger liste dient als Ankerzeiger auf die Liste
   // und gleichzeitig als Handle zum Zugriff auf die Liste
   element_t *liste = NULL;

   do {
      printf("Funktionsauswahl: N(euer Eintrag) L(oeschen) A(usgabe) E(nde)");
      fgets(kommando, EINGABE_LEN, stdin);

      /* fuehrende Leerzeichen entfernen und Text in Kleinbuchstaben umwandeln */
      kommando_anpassen(kommando);

      if ((strlen(kommando) == 1 && strncmp(kommando, "n", 1) == 0) ||
         strncmp(kommando, "neuer eintrag", 14) == 0)
      {
         printf("Neuer Eintrag:\n");
         printf("Position:"); fgets(ort, LEN, stdin);
         eingabe_anpassen(ort);
         printf("Temperatur:"); fgets(temperatur_s, LEN, stdin);
         sscanf(temperatur_s, "%f", &t);

         n = einfuegen(&liste, ort, t);

         // n = einfuegen_sort(&liste, ort, t);  /* Einfuegen gemaess einer Sortierreihenfolge */

         if (n == 1)
            printf("Element eingefuegt.\n");
```

```
            else
               printf("Kein Element eingefuegt\n");

            continue;
         }
         if ((strlen(kommando) == 1 && strncmp(kommando, "l", 1) == 0) ||
            strncmp(kommando, "löschen", 7) == 0 || strncmp(kommando, "loeschen", 8) == 0)
         {
            printf("Loeschen:\n");
            printf("Ort:"); fgets(ort, LEN, stdin);
            eingabe_anpassen(ort);

            n = loeschen(&liste, ort);
            if (n>0)
               printf("%d Element(e) geloescht\n", n);
            else
               printf("Nichts geloescht!\n");

            continue;
         }
         if ((strlen(kommando) == 1 && strncmp(kommando, "a", 1) == 0) ||
            strncmp(kommando, "anzeigen", 4) == 0)
         {
            printf("Anzeige aller Elemente:\n");

            anzeigen(liste);

            continue;
         }
         if (strncmp(kommando, "e", 1) == 0 || strncmp(kommando, "ende", 4) == 0)
         {
            printf("Ende\n");
            break;
         }

      } while (1);
      return 0;
   }
```