Aufgabe 37

Verwaltung von Listen und Dateizugriff

a) Wie können Listen in C verwaltet werden? Wie wird der belegte Speicher wieder freigegeben?

In der Praxis wird häufig ein "Dummy"-Element als Listenanker verwendet, um eine Spezialbehandlung für die leere Liste zu vermeiden (vgl. Vorlesung).

Verzeigerung innerhalb der Struktur oder durch eigene Struktur:

```
struct test
{
  int zahl;
  struct test *next;
};
```

```
struct test
{
  int zahl;
};
struct list
{
  struct test *element;
  struct list *next;
};
```

In beiden fällen muß für jedes Listenelement dynamisch Speicher beschafft werden. Diese Aufgabe wird sinnvollerweise zusammen mit einer Initialisierung in eine eigene Funktion ausgelagert. Beispiel (für die obige Lösung mit 2 Strukturen):

```
struct list *new_list_elem()
{
  struct test *t;
  struct list *l;
  t = (struct test *)malloc(sizeof(struct test));
  l = (struct list *)malloc(sizeof(struct list));
  if(t==NULL || l==NULL)
    return NULL; /* alternativ: Fehlerausgabe/Abbruch */
  t->zahl = 0;
  l->element = t;
  l->next = NULL;
  return l;
}
```

Bei der Speicherfreigabe muß die Liste durchlaufen und Element für Element freigegeben werden

```
void free_list(struct list *l)
{
  struct list *current = l;
  while(l)
  {
    current = l;
    l = l->next;
    free(current->element);
    free(current);
  }
}
```


Eine weitere Möglichkeit bei der Verwendung von zwei Strukturen besteht darin, die erste Struktur direkt in die zweite Struktur zu integrieren (statt Zeiger):

```
struct test
{
  int zahl;
};
struct list
{
  struct test element; /* nicht struct test *element */
  struct list *next;
};
```

Die Funktionen zur Listenbearbeitung sehen dann z.B. so aus:

```
struct list *new_list_elem()
{
  struct list *l;
  l = (struct list *)malloc(sizeof(struct list));
  if(l==NULL)
    return NULL; /* alternativ: Fehlerausgabe/Abbruch */
  l->element.zahl = 0; /* (siehe unten) */
  l->next = NULL;
  return l;
}

void free_list(struct list *l)
{
  struct list *current = l;
  while(l)
  {
    current = l;
    l = l->next;
    free(current);
  }
}
```

Um obige Lösung für die Listenvariante mit nur einer Struktur anzupassen muß lediglich überall "struct list" durch "struct test" ersetzt werden und die Zeile

```
l->element.zahl = 0;
```

zu

```
l->zahl = 0;
```

abgeändert werden.

b) Was sollten Sie beachten, wenn eine Funktion Daten sowohl von der Standardeingabe als auch aus einer Datei einlesen können soll?

Die Funktion sollte als zusätzlichen Parameter einen Dateizeiger erhalten, dann liest sie bei Übergabe von `stdin` als Parameter von der Standardeingabe.
Eine weiterer Parameter (`quiet`) kann verwendet werden, um beim Lesen aus der Datei Bildschirmausgaben zu unterdrücken.

```
void read_test(FILE *f, struct test *t, int quiet)
{
  if(!quiet)
    printf("Zahl: ");
  fscanf(f, "%d", &t->zahl);
}
```

c) Wie sollten Fehlermeldungen ausgegeben werden?

Allgemeiner Fehler:

```
  fprintf(stderr, ...);
```

Fehler bei Aufruf einer System- / Bibliotheksfunktion:

```
  perror(...);
```

Bem.: Die Ausgaben auf `stdout` und `stderr` erscheinen normalerweise gemeinsam am Bildschirm, sollten aber dennoch getrennt behandelt werden, da die beiden Kanäle unterschiedliche Bedeutung haben und vom Benutzer unabhängig voneinander betrachtet werden können.