



Aufgabe 1. Lese noch einmal im Skript die Sektion 7.5 und implementiere doppelt verkettete Listen, die `double`-Variablen speichern.

```
1  /* Definiere hier angemessene Strukturen für einen
2     einzelnen Listeneintrag und die Liste selbst. */
3
4  /* Leere Liste erstellen */
5  LIST *list_create();
6
7  /* Element hinter E einfügen, NULL heißt am Anfang */
8  LISTNODE *list_insert(LIST *L, LISTNODE *E, double p);
9
10 /* Element am Anfang bzw. Ende einfügen */
11 LISTNODE *list_unshift(LIST *L, double p);
12 LISTNODE *list_push(LIST *L, double p);
13
14 /* Element am Anfang bzw. Ende entfernen und
15     die Daten zurück geben */
16 double list_shift(LIST *L);
17 double list_pop(LIST *L);
18
19 /* eine Element aus der Liste entfernen */
20 void list_delete(LIST *L, LISTNODE *E);
21
22 /* zwei Listen zusammenfügen */
23 LIST *list_merge(LIST *L, LIST *M);
24
25 /* Liste inklusive allen Elementen frei geben */
26 void list_free(LIST *L);
```



Aufgabe 2. In dieser Aufgabe geht es um numerische Integration.

- a) Implementiere eine Integrationsfunktion, die das Intervall $[a, b]$ in n gleich große Teile aufteilt, für diese Intervalle $[a_i, b_i]$ jeweils die Trapezsumme $\frac{(f(b_i)+f(a_i))(b_i-a_i)}{2}$ berechnet und diese aufsummiert:

```
1 double integrate(double a, double b,  
2 double (*f)(double), unsigned int n);
```

- b) Schreibe nun eine Funktion, die nicht die Anzahl der Teilintervalle erhält, sondern eine "Fehlertoleranz" e . Die Funktion die Aufteilung solange verfeinern, bis sich der approximierte Wert für das Integral durch eine Verfeinerung nur noch um weniger als e ändern würde.

Aufgabe 3.

- a) Implementiere die Addition, Multiplikation, Potenzen und Division komplexer Zahlen. Verwende dazu folgende Header-Datei:

```
1 #ifndef _COMPLEX__H  
2 #define _COMPLEX__H  
3  
4 typedef struct _COMPLEX {  
5     double real;  
6     double imag;  
7 } COMPLEX;  
8  
9 COMPLEX cplx_add(COMPLEX a, COMPLEX b);  
10 COMPLEX cplx_mul(COMPLEX a, COMPLEX b);  
11 COMPLEX cplx_pow(COMPLEX a, unsigned long n);  
12 COMPLEX cplx_div(COMPLEX a, COMPLEX b);  
13  
14 #endif
```

- b) Implementiere die Addition, Multiplikation und Division sowie das Kürzen rationaler Zahlen. Schreibe dazu erst die Header-Datei.