Aufgabe 1. Die coole Labyrinth-Aufgabe vom letzten Mal machen. Aber nicht das mit den Matrizen, nicht umbedingt jedenfalls, nein das muss nicht sein. Kann aber.

Aufgabe 2.

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

```
#ifndef _COMPLEX__H
#define _COMPLEX__H
typedef struct {
        double real;
        double imag;
} COMPLEX;

COMPLEX cplx_add(COMPLEX a, COMPLEX b);
COMPLEX cplx_mul(COMPLEX a, COMPLEX b);
COMPLEX cplx_pot(COMPLEX a, unsigned long n);
COMPLEX cplx_div(COMPLEX a, COMPLEX b);
#endif
```

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

Aufgabe 3. Implementiere „dynamische Arrays“. Also Funktionen, die es leicht ermöglichen mit dynamisch großen Arrays zu arbeiten. Verwende eine Header-Datei in folgendem Stil:

```
#ifndef _DBLARRAY__H
#define _DBLARRAY__H

typedef struct {
  double *data; /* eigentliches Array */
  int length;   /* erstes nicht verwendetes Element */
  int _size;    /* Menge der allokierten Elemente */
} DBLARRAY;

/* initialisiert eine Array-Datenstruktur */
DBLARRAY *dblarray_init();

/* gibt eine Array-Datenstruktur wieder frei */
void dblarray_free(DBLARRAY *);

/* setzt den Wert an der Stelle i auf val
 * falls nötig wird neuer Speicher allokiert und
 * alle Elemente bis dorthin mit 0 initialisiert */
int dblarray_set(DBLARRAY *, int i, double val);

/* setze das erste nicht initialisierte Element auf
 * val sollte es noch nicht existieren wird neuer
 * Speicher allokiert */
int dblarray_push(DBLARRAY *, double val);

/* gib den Wert an der Stelle i zurück */
double dblarray_get(DBLARRAY *, int i);

#endif
```

Teste deine prächtigen Arrays ausgiebig!