Documentation Group Project Structs (Assignment10)

Mario Neuner, Christoph Bichlmeier, Jakob Guentner
Leander Decristoforo

May 20, 2025

Contents

1 Part A: Elastic Collision **2**
1.1 Idea 2
1.2 Implementation 2
1.3 Output 2
1.4 Code 3

2 Part B and C: Rocket Equation **5**
2.1 Idea 5
2.2 Implementation 6
2.3 Output 6
2.4 Code 6

1 Part A: Elastic Collision

1.1 Idea

In this problem the task was to compute the elastic collision between two bodies, whereby the bodies are described as point masses. The idea for the solution of the problem was to bundle various data sets into structures to compute without losing the oversight over the given starting conditions.

1.2 Implementation

Bringing the idea with bundling the input data in structures in a more concrete way, we created different structures for describing the two bodies with the according starting conditions: The mass and the starting velocity. Furthermore, other structures hold the information about the starting and end state. In more detail, we created a function for computing the end state while the starting conditions serve as an input. The code in the function checks if the collision takes place in the first place and furthermore computes the end state if it does. In conclusion, the computed end state is returned to the main function and displayed in the command line.

1.3 Output

`The program compiles with "gcc -Wall -Wextra -Werror -Wpedantic -std=c18 stoss.c -o stoss".` When running `./stoss <m1> <v1> <m2> <v2>` you get the output:

"Kein Stoss! Nach dem Stoß: v_1=v1, v_2=v2"
or
"Nach dem Stoß: v_1=v1, v_2=v2"

1.4 Code

Listing 1: Elastic Collision

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

struct reiter
{
  float masse;
  float geschwindigkeit;
};

struct zustand
{
  struct reiter a;
  struct reiter b;
};

struct zustand stosse_reiter(struct zustand vorher);

int main(int argc, char **argv)
{

  if (argc != 5)
  {
    printf("Fehler:␣Falsche␣Anzahl␣an␣Parametern.\
       nVerwendung:␣%s␣<m1>␣<v1>␣<m2>␣<v2>\n", argv
       [0]);
    return 1;
  }

  struct reiter r_1 = {atof(argv[1]), atof(argv[2])
     }; // starting conditions input via command
     line
  struct reiter r_2 = {atof(argv[3]), atof(argv[4])
     };

  struct zustand vorher = {r_1, r_2}; // definition
     of prior state

  struct zustand nachher = stosse_reiter(vorher);

```

```c
  printf("Nach␣dem␣Stoss:␣v_1=%f,␣v_2=%f\n", nachher.a.geschwindigkeit, nachher.b.geschwindigkeit);

  return 0;
}

struct zustand stosse_reiter(struct zustand vorher) // check if collision takes place in the first place with three if-statements; computation of end state with corresponding values
{
  if (vorher.a.geschwindigkeit <= 0 && vorher.b.geschwindigkeit >= 0)
  {
    printf("Kein␣Stoss!\n");
    return vorher;
  }
  if (vorher.a.geschwindigkeit < 0 && vorher.b.geschwindigkeit < 0 && vorher.a.geschwindigkeit <= vorher.b.geschwindigkeit)
  {
    printf("Kein␣Stoss!\n");
    return vorher;
  }
  if (vorher.a.geschwindigkeit > 0 && vorher.b.geschwindigkeit > 0 && vorher.a.geschwindigkeit <= vorher.b.geschwindigkeit)
  {
    printf("Kein␣Stoss!\n");
    return vorher;
  }
  struct zustand nachher = vorher;

  nachher.a.geschwindigkeit = (vorher.a.masse * vorher.a.geschwindigkeit + vorher.b.masse * vorher.b.geschwindigkeit +
  vorher.b.masse * (vorher.b.geschwindigkeit - vorher.a.geschwindigkeit)) / (vorher.a.masse + vorher.b.masse);

  nachher.b.geschwindigkeit = (vorher.a.masse * vorher.a.geschwindigkeit + vorher.b.masse * vorher.b.geschwindigkeit +
  vorher.a.masse * (vorher.a.geschwindigkeit -
     vorher.b.geschwindigkeit)) / (vorher.a.masse +
     vorher.b.masse);

  return nachher;
}
```

2 Part B and C: Rocket Equation

A Rocket is given with the values mass of the rocket, mass of the fuel, speed(t=0), speed of fuel ejection, rate of fuel ejection. These values have to be given beforehand. In exercise (b) the final Speed of the rocket when it runs out of fuel has to be calculated The exercise (c) asked to calculate the distance travelled and the time spend to reach the final speed.

2.1 Idea

The program simulates a rocket accelerating by ejecting fuel. As fuel is ejected, the rocket loses mass and gains velocity as shown in the rocket equation. The simulation calculates small changes in mass and time and adds them up until all the fuel is used up. For exercise (b) we calculate the speed gained for a small weight loss by expelling fuel, then we add the weight losses up, till mass of fuel equals zero. In exercise (c) we implement a time and distance by introducing a (zeitschritt) which calculates the small-time interval determined by the weight loss created expelling fuel. As a result, the time spend till all fuel was ejected is calculated and the distance travelled can be determined by integrating over the speed by time.

2.2 Implementation

The Program is using a struct (Rakete) to store its physical properties like mass, velocity, fuel, and distance to simulate the motion of a rocket accelerating by ejecting fuel. The simulation starts with userprovided input values like dry mass, fuel mass, fuel ejection velocity, and fuel ejection rate. In a loop, it repeatedly calculates the rocket's velocity change and distance traveled over small time steps (zeitschritt), based on the mass of fuel lost (masseschritt). The loop continues until fuel mass equals zero. When this happens, the program prints the rocket's final velocity, the time it took to reach it, and the total distance traveled.

2.3 Output

`The program compiles with "gcc -Wall -Wextra -Werror -Wpedantic -std=c18 rakete.c -o rakete".` When running `./rakete <masse_leer> <masse_treibstoff> <geschwindigkeit_treibstoff> <massenverlustrate_treibstoff>` you get the output:

"Endgeschwindigkeit: ...
Endgeschwindigkeit erreicht nach: t=..., x=..."

2.4 Code

Listing 2: Rocket

```c
    #include <stdio.h>
    #include <stdlib.h>
    // Authors: Jakob G., Christoph B., Mario N.,
       Leander D.

    // Struct for rocket
    typedef struct {
        double masse_leer;
        double geschwindigkeit_rakete;

        double masse_treibstoff;
        double geschwindigkeit_treibstoff;
        double massenverlustrate_treibstoff;

        double time;
        double distance;
    } Rakete;

    int masseschritt(Rakete *rocket, double delta_masse);
    int zeitschritt(Rakete *rocket, double delta_t);

    int main(int argc, char **argv) {

        if (argc != 5) {
            printf("Fehler:␣Falsche␣Anzahl␣an␣Parametern.\nVerwendung:␣%s␣<masse_leer>␣<masse_treibstoff>␣<geschwindigkeit_treibstoff>␣<massenverlustrate_treibstoff>\n", argv[0]);
            return 1;
        }

        Rakete prototyp = {
            .masse_leer = atof(argv[1]),
            .geschwindigkeit_rakete = 0,
            .masse_treibstoff = atof(argv[2]),
            .geschwindigkeit_treibstoff = atof(argv[3]),
            .massenverlustrate_treibstoff = atof(argv[4])};

        Rakete rakete = prototyp;

        //while (masseschritt(&rakete, 1e-5));

        //printf("Endgeschwindigkeit: %f\n", rakete.geschwindigkeit_rakete);

        /*
            Berechnen Sie hier wann und wo die Endgeschwindigkeit erreicht wurde
        */
        // printf("Endgeschwindigkeit erreicht nach: t=%f, x=%f\n", ...);
        while (zeitschritt(&rakete, 0.0001));
```

```
        printf("Endgeschwindigkeit:␣%f\n", rakete.geschwindigkeit_rakete);
        printf("Endgeschwindigkeit␣erreicht␣nach:␣t=%f,␣x=%f\n", rakete.time, rakete.distance);
        return 0;
    }

    int masseschritt(Rakete *rakete, double delta_masse) {
        if (delta_masse > rakete->masse_treibstoff) {
            return 0;
        }

        double dv_r = 0;    // Var for change in velocity
        // Calculate the change in velocity
        dv_r = (rakete->geschwindigkeit_treibstoff * delta_masse) / (rakete->masse_treibstoff + rakete->masse_leer);

        // Update the rocket's mass and velocity
        rakete->masse_treibstoff -= delta_masse;
        rakete->geschwindigkeit_rakete += dv_r;

        return 1;
    }

    int zeitschritt(Rakete *rakete, double delta_t) {
        double delta_m = delta_t * rakete->massenverlustrate_treibstoff;
        if (delta_m > rakete->masse_treibstoff) {
            return 0;
        }
        // Update the rocket's time and distance
        rakete->distance += rakete->geschwindigkeit_rakete * delta_t;
        rakete->time += delta_t;

        // Compute the new velocity
        masseschritt(rakete, delta_m);

        return 1;
    }
```