Inhaltsverzeichnis

1. Autobau („wmb“)

1.1. Lösungsidee

Es wurden 3 Klassen (car, tire, engine) implementiert. Diese Klassen haben den Zweck einen Autozusammenbau darzustellen. Die einzelnen Klassen beinhalten einige Informationen über das entsprechende Autoteil bzw. über das gesamte Auto. Die Klasse car stellt das gesamte zusammengebaute Auto dar. Car beinhaltet zwei zusätzlich zu einigen String und Integer Komponenten noch eine Datenkomponente vom Typ tire und eine Komponente vom typ engine.

Annahme: Grundsätzlich werden pro Auto nicht alle 4 Reifen extra gespeichert (in Form einer Datenstruktur) sondern lediglich ein gesamter Reifensatz.

Für jede Klasse wurde ein Konstruktor implementiert, der die jeweiligen Datenkomponenten mit den Übergabeparametern initialisiert. Wenn ein Objekt des der Klasse cars erstellt wird, so müssen auch zwei Objekte der Klassen tire und engine übergeben werden.

Für jede Klasse wurde der << Operator überschrieben, sodass eine Ausgabe aller Klassen mit *cout << Classname;* möglich ist.

Annahme: Da die Angabe meiner Ansicht nach etwas missverständlich formuliert wurde, war ich mir nicht zu 100% sicher, ob der >> Operator auch überschrieben werden musste.

Grundsätzlich wäre eine Lösung mit enums um einiges schöner, jedoch ist dies für die Ausgabe problematisch, da nur der „index“ der einzelnen Enums ausgegeben wird. Für eine korrekte Ausgabe müsste eine „Mapping“ Funktion geschrieben werde, die die enums in Strings umwandelt. Jedoch erschien mir dies für einen zu großen Overhead.

1.2. Code

Car.h

```
//
// Created by Michael Neuhold on 28.11.19.
//

#ifndef ADT_CAR_H
#define ADT_CAR_H

#include <iostream>
#include "./main.h"
#include "./tire.h"
#include "./engine.h"

using namespace std;

class car {
    friend std::ostream &operator<<(std::ostream &os,const car
&finished_car);
public:
    car(const string type, string color,const int serial_number,const
date_t production_date,const string production_place,const string
gearbox,const string type_of_drive,int top_speed, int weight, tire &,
engine &);
    ~car() = default;
private:
    string type;
    string color;
    int serial_number;
    date_t production_date;
    string production_place;
    string gearbox;
    string type_of_drive;
    int top_speed;
    int weight;
    tire car_tire;
    engine car_engine;
};

#endif //ADT_CAR_H
```

Car.cpp

```
//
// Created by Michael Neuhold on 28.11.19.
//

#include <iostream>
#include "car.h"
#include "./main.h"
#include "./tire.h"

using namespace std;

car::car(const string type, string color,const int serial_number,
        const date_t production_date,const string production_place,
        const string gearbox,const string type_of_drive, int
top_speed,
        int weight, tire &car_tire, engine &car_engine) :
```

```
car_tire(car_tire), car_engine(car_engine) {
    this -> type = type;
    this -> color = color;
    this -> serial_number = serial_number;
    this -> production_date = production_date;
    this -> production_place = production_place;
    this -> gearbox = gearbox;
    this -> type_of_drive = type_of_drive;
    this -> top_speed = top_speed;
    this -> weight = weight;
    this -> car_tire = car_tire;
    this -> car_engine = car_engine;
}

std::ostream &operator<<(std::ostream &os,const car &finished_car) {
    return os << "car: {\n"
                 "\ttype: " << finished_car.type <<
                 "\n\tcolor: " << finished_car.color <<
                 "\n\tserial number: " << finished_car.serial_number
<<
                 "\n\tproduction_date: " <<
finished_car.production_date.day <<
                 "." << finished_car.production_date.month <<
                 "." << finished_car.production_date.year <<
                 "\n\tproduction_place: " <<
finished_car.production_place <<
                 "\n\tgearbox: " << finished_car.gearbox <<
                 "\n\ttype_of_drive: " << finished_car.type_of_drive
<<
                 "\n\ttop_speed: " << finished_car.top_speed <<
                 "\n\tweight: " << finished_car.weight <<
                 "\n\t" << finished_car.car_tire <<
                 "\t" << finished_car.car_engine <<
                 " }" << std::endl;
}
```

Engine.h

```
//
// Created by Michael Neuhold on 28.11.19.
//

#ifndef ADT_ENGINE_H
#define ADT_ENGINE_H

#include <iostream>
#include "./main.h"

class engine {
    friend std::ostream &operator<<(std::ostream &,const engine &);
public:
    engine(const int engine_number,const string fuel_type, int power,
double n_consumption,const date_t production_date);
    ~engine() = default;
private:
    int engine_number;
    int power;
    double n_consumption;
    string fuel_type;
    date_t production_date;
};

#endif //ADT_ENGINE_H
```

Engine.cpp

```
//
// Created by Michael Neuhold on 28.11.19.
//

#include "./main.h"
#include "./engine.h"

engine::engine(const int engine_number,const string fuel_type, int
power, double n_consumption,const date_t production_date) {
    this -> engine_number = engine_number;
    this -> fuel_type = fuel_type;
    this -> power = power;
    this -> n_consumption = n_consumption;
    this -> production_date = production_date;
}

std::ostream &operator<<(std::ostream &os,const engine &car_engine) {
    return os << "engine: { \n"
                 "\tengine_number: " << car_engine.engine_number <<
                 "\n\tfuel_type: " << car_engine.fuel_type <<
                 "\n\tpower: " << car_engine.power <<
                 "\n\tn_consuption: " << car_engine.n_consumption <<
                 "\n\tproduction_date: "<<
car_engine.production_date.day <<
                 "." << car_engine.production_date.month <<
                 "." << car_engine.production_date.year <<
                 "\n\t}" << std::endl;
}
```

Tire.h

```
//
// Created by Michael Neuhold on 28.11.19.
//

#ifndef ADT_TIRE_H
#define ADT_TIRE_H

#include <iostream>
#include "main.h"

using namespace std;

class tire {
    friend std::ostream &operator<<(std::ostream &os,const tire
&car_tire);
public:
    tire(const double rim_diameter,const int production_year,const
string speed_index,const string manufacturer);
    ~tire() = default;
private:
    double rim_diameter;
    int procduction_year;
    string speed_index;
    string manufacturer;
};

#endif //ADT_TIRE_H
```

Tire.cpp

```
//
// Created by Michael Neuhold on 28.11.19.
//

#include "tire.h"

tire::tire(const double rim_diameter,const int production_year,const
string speed_index,const string manufacturer) {
    this -> rim_diameter = rim_diameter;
    this -> procduction_year = production_year;
    this -> speed_index = speed_index;
    this -> manufacturer = manufacturer;
}

std::ostream &operator<<(std::ostream &os,const tire &car_tire) {
    return os << "tire: {\n"
                 "\trim diameter: " << car_tire.rim_diameter <<
              "\n\tproduction year: " << car_tire.procduction_year <<
              "\n\tspeed index: " << car_tire.speed_index <<
              "\n\tmanufacturer: " << car_tire.manufacturer <<
              "\n\t}" << std::endl;
}
```

Main.h (eigentlich types.h)

```
//
// Created by Michael Neuhold on 28.11.19.
//

#ifndef ADT_MAIN_H
#define ADT_MAIN_H

#include <iostream>

using namespace std;

// date type
typedef struct {
    int day;
    int month;
    int year;
} date_t;

// color enum
// enum color_t { black , white , red , yellow , grey };

// speed index enum
// enum speed_index_t { E , F , G , J , K , L , M , N , P , Q , R , S
, T , U , H , V , W , Y };

// fuel type enum
// enum fuel_t { diesel , benzin , electric };
```

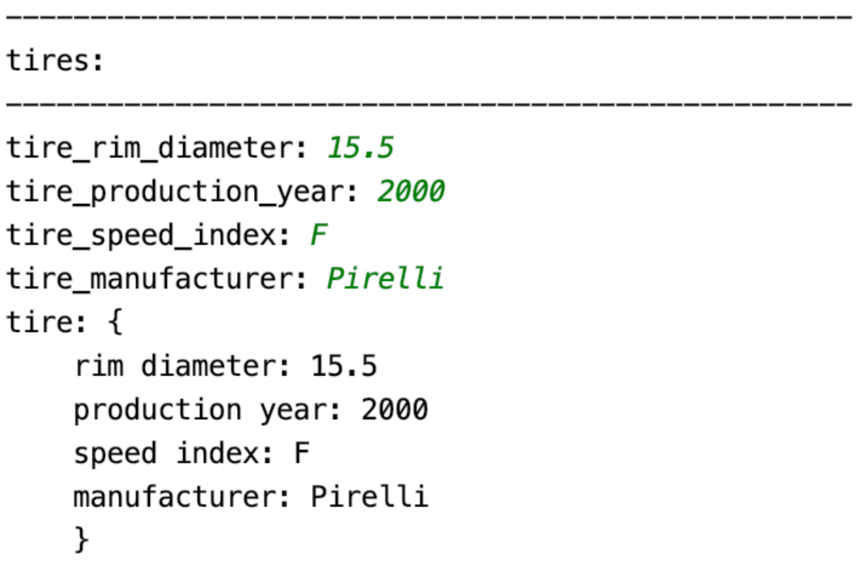
```
#endif //ADT_MAIN_H
```

Main.cpp

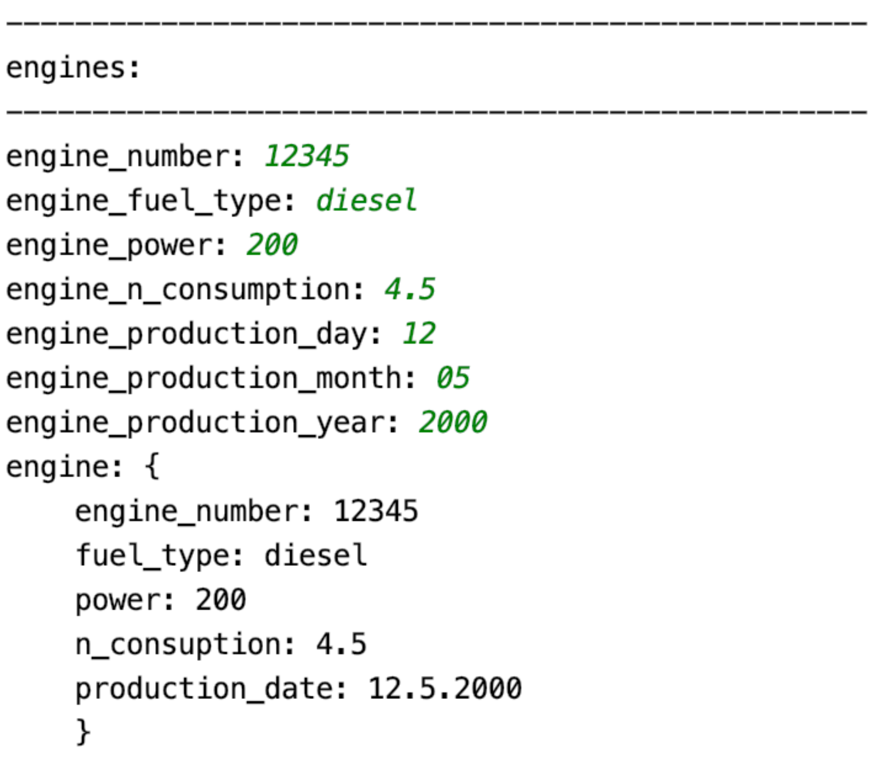
```
#include <iostream>
#include "./main.h"
#include "./tire.h"
#include "./engine.h"
#include "./car.h"

using namespace std;
s
void Separator() {
    for(int i = 0; i < 50; i++ ) {
        cout << "-";
    }
    cout << endl;
}
void Header(const string &header_text) {
    Separator();
    cout << header_text << endl;
    Separator();
}

int main() {

    Header("tires:");

    /*------------------------------------*/
    // input values
    double tire_rim_diameter;
    int tire_production_year;
    string tire_speed_index;
    string tire_manufacturer;
    /*------------------------------------*/
    cout << "tire_rim_diameter: ";
    cin >> tire_rim_diameter;
    cout << "tire_production_year: ";
    cin >> tire_production_year;
    cout << "tire_speed_index: ";
    cin >> tire_speed_index;
    cout << "tire_manufacturer: ";
    cin >> tire_manufacturer;
    /*------------------------------------*/

    // init new tries
        // double rim_diameter, int production_year, string
speed_index, string manufacturer
    tire t1(tire_rim_diameter, tire_production_year, tire_speed_index,
tire_manufacturer);
    // output of tires
    cout << t1;

    /*-----------------------------------------------------------------
----------------------------*/

    Header("engines:");

    /*------------------------------------*/
    // input values
    int engine_number;
    string engine_fuel_type;
    int engine_power;
    double engine_n_consumption;
    int engine_production_day;
    int engine_production_month;
    int engine_production_year;
    /*------------------------------------*/
    cout << "engine_number: ";
    cin >> engine_number;
    cout << "engine_fuel_type: ";
    cin >> engine_fuel_type;
    cout << "engine_power: ";
    cin >> engine_power;
    cout << "engine_n_consumption: ";
    cin >> engine_n_consumption;
    cout << "engine_production_day: ";
    cin >> engine_production_day;
    cout << "engine_production_month: ";
    cin >> engine_production_month;
    cout << "engine_production_year: ";
    cin >> engine_production_year;
    /*------------------------------------*/
    // init new engines
        // int engine_number, string fuel_type, int power, double
n_consumption, date_t production_date);
    date_t production_date_engine{ .day = engine_production_day ,
.month = engine_production_month , .year = engine_production_year };
    engine e1(engine_number,engine_fuel_type, engine_power,
engine_n_consumption, production_date_engine);
    // output of engines
    cout << e1;

    /*--------------------------------------------------------------------
----------------------------*/

    Header("car:");

    /*------------------------------------*/
    // input values
    string car_type;
    string car_color;
    int car_serial_number;
    string car_production_place;
    string car_gearbox;
    string car_type_of_drive;
    int car_top_speed;
    int car_weight;
    int car_production_day;
    int car_production_month;
    int car_production_year;
    /*------------------------------------*/
    cout << "car_type: ";
    cin >> car_type;
    cout << "car_color: ";
    cin >> car_color;
    cout << "car_serial_number: ";
    cin >> car_serial_number;
    cout << "car_production_place: ";
    cin >> car_production_place;
    cout << "car_gearbox: ";
    cin >> car_gearbox;
    cout << "car_type_of_drive: ";
    cin >> car_type_of_drive;
    cout << "car_top_speed: ";
    cin >> car_top_speed;
    cout << "car_weight: ";
    cin >> car_weight;
    cout << "car_production_day: ";
    cin >> car_production_day;
    cout << "car_production_month: ";
    cin >> car_production_month;
    cout << "car_production_year: ";
    cin >> car_production_year;
    /*------------------------------------*/
    // init new cars
        // string type, string color, int serial_number, date_t
production_date, string production_place, string gearbox, string
type_of_drive, int top_speed, int weight, tire &tire, engine &engine
);
    date_t production_date_car{ .day = car_production_day , .month =
car_production_month , .year = car_production_year };
    car
c1(car_type,car_color,car_serial_number,production_date_car,car_produc
tion_place,

car_gearbox,car_type_of_drive,car_top_speed,car_weight,t1,e1);
    // output of cars
    cout << c1;

    /*----------------------------------------------------------------------
----------------------------*/

}
```

1.3. Testfälle

Bei diesem Testfall wird wie aus dem Main-File zu entnehmen alle Wert von der Konsole eingelesen und im Anschluss mit dem überladenen << Operator ausgegeben.



```
------------------------------------------------
tires:
------------------------------------------------
tire_rim_diameter: 15.5
tire_production_year: 2000
tire_speed_index: F
tire_manufacturer: Pirelli
tire: {
    rim diameter: 15.5
    production year: 2000
    speed index: F
    manufacturer: Pirelli
    }
```



```
------------------------------------------------
engines:
------------------------------------------------
engine_number: 12345
engine_fuel_type: diesel
engine_power: 200
engine_n_consumption: 4.5
engine_production_day: 12
engine_production_month: 05
engine_production_year: 2000
engine: {
    engine_number: 12345
    fuel_type: diesel
    power: 200
    n_consuption: 4.5
    production_date: 12.5.2000
    }
```

```
--------------------------------------------------
car:
--------------------------------------------------
car_type: Audi
car_color: black
car_serial_number: 98765
car_production_place: München
car_gearbox: Manuell
car_type_of_drive:
quattro
car_top_speed: 200
car_weight: 1480
car_production_day: 20
car_production_month: 3
car_production_year: 2003
car: {
    type: Audi
    color: black
    serial number: 98765
    production_date: 20.3.2003
    production_place: München
    gearbox: Manuell
    type_of_drive: quattro
    top_speed: 200
    weight: 1480
    tire: {
    rim diameter: 15.5
    production year: 2000
    speed index: F
    manufacturer: Pirelli
    }
    engine: {
    engine_number: 12345
    fuel_type: diesel
    power: 200
    n_consuption: 4.5
    production_date: 12.5.2000
    }
 }
```

2. ADT Graph

2.1. Lösungsidee

Grundsätzlich wurde der Lösungsansatz von der Übung 5 übernommen. Graph und Vertex wurden als Klasse implementiert. Die Klasse Vertex beinhaltet lediglich eine Datenkomponente Payload und einen Getter für diese Datenkomponente. In der Klasse Graph stehen weiter Funktionen wie zum Beispiel das Hinzufügen von Vertex-Nodes und Edges bereit (wie in der Angabe beschreiben).

Um die eingefügten Vertex – Elemente eindeutig identifizieren zu können, wird nach dem Hinzufügen ein Objekt der Klasse handle_t zurückgegeben. Handle_t besitz eine Datenkomponente, welche beim Anlegen des Objektes mit dem Index des Feldes initialisiert wird, an dem sich der eingefügte Vertex befindet.

Es wurde eine Print Funktion implementiert, in der im Grunde „out" wie „cout" verwendet werden kann.

Der Dijkstra Shortest Path Algorithmus wurde rekursiv implementiert. Der Algorithmus kann für jeden angelegten Graphen angewandt werden. Voraussetzung für diese Implementierung ist, dass keine Vertex – Elemente gelöscht werden, da der Algorithmus mit den aufsteigenden Indexwerten arbeitet. Es wurde der rekursiven Funktion immer ein Hilfsfeld mit den bereits verwendeten Graph-Nodes mitgegeben (die nicht mehr besucht werden können). In der rekursiven Funktion wird dann immer das Feld „shortest path" aktualisiert, wenn der Pfad zu einer Node kleiner ist als der bisher eingetragene. Diese Implementierung stellt einen Greedy Algorithmus dar.

2.2. Code

Graph_t.h

```
//
// Created by Michael Neuhold on 29.11.19.
//

#ifndef ADT_GRAPH_T_H
#define ADT_GRAPH_T_H

#include "./vertex_t.h"
#include "./handle_t.h"

class graph_t {

public:

    graph_t() = delete;
    graph_t(std::string graph_name);
    ~graph_t();
    handle_t add_vertex(vertex_t vertex);
    void add_edge(handle_t const from, handle_t const to, int weight);
    int shortest_path (handle_t const from, handle_t const to) const;
    std::ostream & print (std::ostream & out) const;

private:
    std::string graph_name{"test1"};
    int vertex_count{0};
    vertex_t **vertex_list;
    int **matrix;
    void dijkstra(int from, int *vertex_visited, int *shortest_path, int
not_visited_count, int current_path) const;

};

#endif //ADT_GRAPH_T_H
```

Graph_t.cpp

```
//
// Created by Michael Neuhold on 29.11.19.
//

#include "graph_t.h"
#include "vertex_t.h"
#include "handle_t.h"

/*-------------------------------------------------------------------------------*/

graph_t::graph_t(std::string graph_name) {
    // graph name
    this -> graph_name = graph_name;
    // allocate memory
    this -> vertex_list = new vertex_t*[this -> vertex_count];
}

/*-------------------------------------------------------------------------------*/

handle_t graph_t::add_vertex(vertex_t vertex) {
    vertex_t *v = new vertex_t {vertex.get_payload()};

    this -> vertex_count++;
    if(this -> vertex_count - 1 == 0) {
        this -> vertex_list = new vertex_t*[1];
        this -> vertex_list[this -> vertex_count - 1] = v;

        this -> matrix = new int*[1];
        this -> matrix[0] = new int[1];
        this -> matrix[0][0] = 0;
    } else {
        this -> vertex_list = (vertex_t**)realloc(this -> vertex_list, sizeof(vertex_t*)
* this -> vertex_count);
        this -> vertex_list[this -> vertex_count - 1] = v;
        this -> matrix = (int**)realloc(this -> matrix, sizeof(int*) * this ->
vertex_count);
        for(int i = 0; i < this -> vertex_count - 1; i++) {
            this -> matrix[i] = (int*)realloc(this -> matrix[i], sizeof(int) * this ->
vertex_count);
            this -> matrix[i][vertex_count - 1] = 0;
        }
        this -> matrix[vertex_count - 1] = new int[vertex_count];
        for(int i = 0; i < vertex_count; i++) {
            this -> matrix[vertex_count - 1][i] = 0;
        }
    }

    handle_t handler(vertex_count - 1);
    return handler;
}

/*----------------------------------------------------------------------------*/

std::ostream & graph_t::print (std::ostream & out) const {
    out << this -> graph_name << "\n";
    out << " ";
    for(int i = 0; i < this -> vertex_count; i++){
        out << " " << this -> vertex_list[i] -> get_payload();
    }
    out << std::endl;
    for(int i = 0; i < this -> vertex_count; i++) {
        out << this -> vertex_list[i] -> get_payload() << " ";
        for(int j = 0; j < this -> vertex_count; j++) {
            out << matrix[i][j] << " ";
        }
        out << "\n";
    }
    return out;
}

/*----------------------------------------------------------------------------*/

void graph_t::add_edge(handle_t const from, handle_t const to, int const weight) {
    this -> matrix[from.getIdentifier()][to.getIdentifier()] = weight;
}

/*----------------------------------------------------------------------------*/
// shortest path algorithm

static bool in_visited(int i, int* vertex_visited, int n) {
    int j = 0;
    while(j < n) {
        if(i == vertex_visited[j]) {
            return true;
        }
        j++;
    }
    return false;
}

void graph_t::dijkstra(int from, int *vertex_visited, int *shortest_path, int
not_visited_count, int current_path) const {
    if(not_visited_count == 0) {
        return;
    }
    int current_shortest_path = INT_MAX;
    int current_shortest_index = INT_MAX;
    for(int i = 0; i < this -> vertex_count; i++) {
        if(this -> matrix[from][i]) {
            if(shortest_path[i] > current_path + this -> matrix[from][i]) {
                shortest_path[i] = current_path + this -> matrix[from][i];
            }
            if(current_shortest_path > this -> matrix[from][i] && !in_visited(i,
vertex_visited, this -> vertex_count - not_visited_count)) {
                current_shortest_path = this -> matrix[from][i];
                current_shortest_index = i;
            }
        }
    }
    if(current_shortest_index != INT_MAX) {
        vertex_visited[this -> vertex_count - not_visited_count] =
current_shortest_index;
        dijkstra(current_shortest_index, vertex_visited, shortest_path,
not_visited_count - 1, current_path + current_shortest_path);
    } else {
        return;
    }
}

int graph_t::shortest_path (handle_t const from, handle_t const to) const {
    int *vertex_vistited = new int[this -> vertex_count];
    int *shortest_path = new int[this -> vertex_count];
    for(int i = 0; i < this -> vertex_count;i++) {
        shortest_path[i] = INT_MAX;
    }
    shortest_path[from.getIdentifier()] = 0;
    int current_path{0};
    dijkstra(from.getIdentifier(),vertex_vistited,shortest_path, this -> vertex_count,
current_path);

    // return to-entry from shortest path "table"
    return shortest_path[to.getIdentifier()];
}

/*--------------------------------------------------------------------------------*/

graph_t::~graph_t() {
    // free allocated memory
    for(int i = 0; i < this -> vertex_count; i++) {
        delete [] this -> matrix[i];
    }
    delete [] this -> matrix;
    delete [] this -> vertex_list;
}

/*--------------------------------------------------------------------------------*/
```

Handle_t.h

```
//
// Created by Michael Neuhold on 01.12.19.
//

#ifndef ADT_HANDLE_T_H
#define ADT_HANDLE_T_H


class handle_t {

public:
    handle_t(int index);
    ~handle_t() = default;
    int getIdentifier() const;

private:
    int identifier;

};

#endif //ADT_HANDLE_T_H
```

Handle_t.cpp

```
//
// Created by Michael Neuhold on 01.12.19.
//

#include "handle_t.h"

handle_t::handle_t(int index) {
    this -> identifier = index;
}

int handle_t::getIdentifier() const {
    return identifier;
}
```

Vertex_t.h

```
//
// Created by Michael Neuhold on 29.11.19.
//

#ifndef ADT_VERTEX_T_H
#define ADT_VERTEX_T_H

#include <iostream>

class vertex_t {
public:
    vertex_t() = delete;
    vertex_t(std::string payload);
    ~vertex_t() = default;
    std::string get_payload();

private:
    std::string payload;

};

#endif //ADT_VERTEX_T_H
```

Vertex_t.cpp

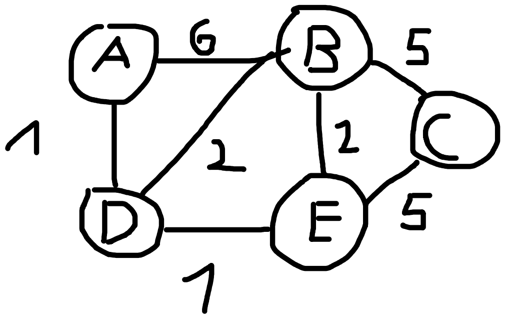
```
//
// Created by Michael Neuhold on 29.11.19.
//

#include "vertex_t.h"

/*---------------------------------------------------------------------------------*/

vertex_t::vertex_t(std::string payload) {
    this -> payload = payload;
}

/*---------------------------------------------------------------------------------*/

std::string vertex_t::get_payload() {
    return this -> payload;
}
```

Main.h

```
#include <iostream>
#include "./vertex_t.h"
#include "./graph_t.h"

using namespace std;

void Separator() {
    for (int i = 0; i < 50; i++) {
        cout << "-";
    }
    cout << endl;
}

void Header(const string &header_text) {
    Separator();
    cout << header_text << endl;
    Separator();
}

int main() {
    Header("graphs");

    // create new graph
    graph_t g1("Graph 1");
    graph_t g2("Graph 2");

    // create new vertex
    vertex_t v1("A");
    vertex_t v2("B");
    vertex_t v3("C");
    vertex_t v4("D");
    vertex_t v5("E");

    vertex_t v6("X");
    vertex_t v7("Y");
    vertex_t v8("Z");

    // add vertex to graph
    handle_t A = g1.add_vertex(v1);
    handle_t B = g1.add_vertex(v2);
    handle_t C = g1.add_vertex(v3);
    handle_t D = g1.add_vertex(v4);
    handle_t E = g1.add_vertex(v5);

    handle_t X = g2.add_vertex(v6);
    handle_t Y = g2.add_vertex(v7);
    handle_t Z = g2.add_vertex(v8);

    // print graph
    g1.print(std::cout);
    Separator();
    g2.print(std::cout);
    Separator();

    // add edges to graph
    g1.add_edge(A,B,6); // A -> B
    g1.add_edge(A,D,1); // A -> D
    g1.add_edge(D,B,2); // D -> B
    g1.add_edge(D,E,1); // D -> E
    g1.add_edge(E,C,5); // E -> C
    g1.add_edge(E,B,2); // B -> C
    g1.add_edge(B,C,5); // E -> B

    g2.add_edge(X,Y,4);
    g2.add_edge(Y,Z,3);

    // print graph
    g1.print(std::cout);
    Separator();
    g2.print(std::cout);
    Separator();

    // shortest path
    cout << "g1 shortest path from A to C: " << g1.shortest_path(A,C) << endl;
    cout << "g1 shortest path from A to B: " << g1.shortest_path(A,B) << endl;

    Separator();

}
```

2.3. Testfälle

```
/Users/michaelneuhold/Documents/FH/Semester/03_Semest
--------------------------------------------------
graphs
--------------------------------------------------
Graph 1
  A B C D E
A 0 0 0 0 0
B 0 0 0 0 0
C 0 0 0 0 0
D 0 0 0 0 0
E 0 0 0 0 0
--------------------------------------------------
Graph 2
  X Y Z
X 0 0 0
Y 0 0 0
Z 0 0 0
--------------------------------------------------
Graph 1
  A B C D E
A 0 6 0 1 0
B 0 0 5 0 0
C 0 0 0 0 0
D 0 2 0 0 1
E 0 2 5 0 0
--------------------------------------------------
Graph 2
  X Y Z
X 0 4 0
Y 0 0 3
Z 0 0 0
--------------------------------------------------
g1 shortest path from A to C: 7
g1 shortest path from A to B: 3
--------------------------------------------------

Process finished with exit code 0
```


A
B
C
D
E
6
5
1
2
2
5
1