

Ausarbeitung Übung 01

Nova-Rechner

Lösungsidee

Durch die Simplizität der Aufgabenstellung ist die Lösungsidee minimal: Sowohl der Kraftstoffverbrauch als auch die Art des Kraftstoffes werden eingelesen und deren numerischer Wert aus der Zeichenkette extrahiert. Gemäß der Formel $(Consumption - x) * 200$ wird für $x = 3$ eingesetzt, wenn das zweite Programmargument dem Wert 2 entspricht bzw. $x = 2$, wenn ebendieses Argument dem Wert 1 entspricht.

Bei einer falschen Anzahl (ungleich drei) an Argumenten, einem Durchschnittsverbrauch kleiner gleich 0 oder einem Kraftstofftyp ungleich eins oder zwei wird das Programm abgebrochen.

Implementierung (nova.c)

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

/**
 * ALL types of fuel the program covers.
 */
typedef enum
{
    PETROL = 1,
    DIESEL = 2
} FuelType;

/**
 * Prints a short guide for the command line
 * interface.
 */
void print_cli_guide()
{
    printf("Program usage: nova <consumption> <fuel type>\n");
    printf("\t- <consumption> = average consumption per 100km.\n");
    printf("\t- <fuel type> = the type of fuel the car model uses (1 = Diesel, 2 = Petrol).");
}

/**
 * Calculates the NOVA according to the formula
 * specified on the exercise sheet.
 * @param avg_consumption The average consumption of a particular car model.
 * @param fuel_type The type of fuel that model uses.
 */
double nova(double avg_consumption, FuelType fuel_type)
{
    int fuel_specific_coefficient = 0;
    switch (fuel_type) {
        case PETROL:
            fuel_specific_coefficient = 3;
            break;
        case DIESEL:
            fuel_specific_coefficient = 2;
            break;
        default:
            printf("Unknown fuel type: %d\n", fuel_type);
            return 0;
    }
    return (avg_consumption - fuel_specific_coefficient) * 200;
}

/*
 * argc is expected to be equal to 3
 * argv[1] is expected to be the average consumption (a positive real number)
 * argv[2] is expected to be the type of fuel for a car model (1 | 2)
 */
```

```

int main(int argc, char *argv[])
{
    if (argc != 3) {
        printf("Invalid number of parameters!\n");
        print_cli_guide();
        return EXIT_FAILURE;
    }

    double avg_consumption = atof(argv[1]); // NOLINT(cert-err34-c)
    if (avg_consumption <= 0) {
        printf("The average fuel consumption / 100km should be a positive number.\n");
        return EXIT_FAILURE;
    }

    FuelType fuel_type = atoi(argv[2]); // NOLINT(cert-err34-c)
    if (fuel_type < 1 || fuel_type > 2) {
        printf("The available fuel types are: Diesel (1), Petrol (2).\n");
        print_cli_guide();
        return EXIT_FAILURE;
    }

    double result = nova(avg_consumption, fuel_type);
    printf("The Nova for the specified car is %.2f\n", result);

    return EXIT_SUCCESS;
}

```

Tests

```

PS C:\Users\Niklas Vest\Documents\FH\fh-hgb-ws1819\swo\ue01\src\nova> .\nova.exe
Invalid number of parameters!
Program usage: nova <consumption> <fuel type>
- <consumption> = average consumption per 100km.
- <fuel type> = the type of fuel the car model uses (1 = Diesel, 2 = Petrol).

```

```

PS C:\Users\Niklas Vest\Documents\FH\fh-hgb-ws1819\swo\ue01\src\nova> .\nova.exe 13.96 2
The Nova for the specified car is 2392.00

```

```

PS C:\Users\Niklas Vest\Documents\FH\fh-hgb-ws1819\swo\ue01\src\nova> .\nova.exe -8.3 1
The average fuel consumption / 100km should be a positive number.

```

```

PS C:\Users\Niklas Vest\Documents\FH\fh-hgb-ws1819\swo\ue01\src\nova> .\nova.exe 7.3 3
The available fuel types are: Diesel (1), Petrol (2).
Program usage: nova <consumption> <fuel type>
- <consumption> = average consumption per 100km.
- <fuel type> = the type of fuel the car model uses (1 = Diesel, 2 = Petrol).

```

Figure 1 Maserati 3200 GTA

```

PS C:\Users\Niklas Vest\Documents\FH\fh-hgb-ws1819\swo\ue01\src\nova> .\nova.exe 7.55 1
The Nova for the specified car is 910.00

```

Figure 2 Audio A8

```

PS C:\Users\Niklas Vest\Documents\FH\fh-hgb-ws1819\swo\ue01\src\nova> .\nova.exe 7.09 1
The Nova for the specified car is 818.00

```

Figure 3 BMW 630 xd GT

```

PS C:\Users\Niklas Vest\Documents\FH\fh-hgb-ws1819\swo\ue01\src\nova> .\nova.exe 6.36 1
The Nova for the specified car is 672.00

```

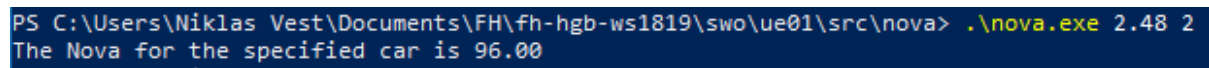
Figure 4 Jaguar R Sport

```

PS C:\Users\Niklas Vest\Documents\FH\fh-hgb-ws1819\swo\ue01\src\nova> .\nova.exe 11.78 2
The Nova for the specified car is 1956.00

```

Figure 5 Nissan 300zx TT



```
PS C:\Users\Niklas Vest\Documents\FH\fh-hgb-ws1819\sw0\ue01\src\nova> .\nova.exe 2.48 2
The Nova for the specified car is 96.00
```

Figure 6 Yamasaki[sic!] YM50 8B

Umfang eines Polygons

Lösungsidee

Punkte werden in der Form `<x>:<y>` ohne eckige Klammern als Argument an das Programm übergeben wobei x und y beliebige ganze Zahlen sein können. Jedes einzelne Zeichen welches in der Powershell / im Windows Command Prompt nicht als Sonderzeichen behandelt wird und ferner keine Ziffer ist kann als Separator zwischen Koordinaten dienen. Jedes (mathematische) Punkt-Literal wird dann einzeln an eine „Parser-Funktion“ übergeben, welche aus der Zeichenkette (dem Programm Argument) die X und Y Komponenten ausliest und etwaige Fehler meldet. Um den Umfang auszurechnen wird immer der jeweils vorhergehende Punkt zwischengespeichert damit mittels Satzes des Pythagoras die Distanz zwischen den Punkten errechnet werden kann. Der letzte Punkt wird insofern speziell behandelt, dass er mit dem ersten Vertex des Polygons „verknüpft“ wird um die Fläche abzuschließen (i. e. Die Länge der letzten Kante zu berechnen). Um der Angabe gerecht zu werden wird jeder Punkt zusätzlich formatiert ausgegeben (siehe Tests).

Die zuvor angesprochene Parser-Funktion liest die übergebene Zeichenkette – also das Punkt-Literal – bis zum ersten nicht-numerischen Zeichen ein und speichert den aus dieser Sub-Zeichenkette extrahierten numerischen Wert als x-Koordinate. Das nächste Zeichen wird übersprungen, weswegen ein Doppelpunkt als Separator nicht verpflichtend ist. Danach wird ein zweiter Versuch unternommen, ein Zahlenliteral zu lesen. Das Ergebnis wird diesmal als y-Koordinate gespeichert. Sollten die Eingabewerte nicht den Vorgaben entsprechen, kann somit eine präzise Fehlerbeschreibung ausgegeben werden.

Anmerkung: Die Standardfunktion `sscanf` könnte diese Funktionalität zwar übernehmen. Ich bin jedoch nicht sicher, ob sie die Aufgabe nicht so weit abkürzt, dass die eigentliche Herausforderung und damit auch meine Punkte verloren gehen.

Implementierung

polycirc.c

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include "cli_parser.h"

/**
 * Prints a short guide for the command line
 * interface.
 * @param prgm_name The name of the program (argv[0])
 */
void print_cli_guide(const char *prgm_name)
{
    printf("Program usage: %s <point> <point> <point> [<point> ...]\n", prgm_name);
    printf("\t- <point> = <x>:<y>.\n");
    printf("\t- <x> and <y> can be an arbitrary integer number.");
}

int main(int argc, char *argv[])
{
    if (argc < 4) {
        printf("polycirc needs at least 3 points to calculate the circumference!\n");
        print_cli_guide(argv[0]);
        return EXIT_FAILURE;
    }

    double distance = 0;
    // initialize to 1 so errors on the first point
```

```

    // literal are logged correctly
    int i = 1;
    ParseError error = NONE;
    // store first point in extra variable so the
    // last point can be "connected to it"
    Point initial = parse_point_literal(argv[1], &error);
    Point buff = initial;
    if (error == NONE) {
        printf("Point %d: (%d,%d)\n", i, buff.x, buff.y);
        i = 2;
        while (i < argc && error == NONE) {
            Point new_point = parse_point_literal(argv[i], &error);
            distance += distance_between(buff, new_point);
            buff = new_point;
            // Note: no increment without error ONLY
            // because error is a termination condition
            if (error == NONE) {
                printf("Point %d: (%d,%d)\n", i, buff.x, buff.y);
                ++i;
            }
        }
    }

    switch (error) {
        case NONE:
            // add last edge of the polygon to the total distance
            distance += distance_between(buff, initial);
            printf("The total circumference is %f\n", distance);
            break;
        case X_COORD:
            printf("The x coordinate on point %d is missing.\n", i);
            break;
        case Y_COORD:
            printf("The y coordinate on point %d is missing.\n", i);
            break;
        default:
            printf("Encountered unknown parsing error: %d\n", error);
            break;
    }

    return error;
}

```

cli_parser.c

```

#ifndef UE01_CLI_PARSER_H
#define UE01_CLI_PARSER_H

#include <ctype.h>

/**
 * An enumeration of errors that can
 * occur during the point parsing
 * process.
 */
typedef enum ParseError
{
    NONE = 0,
    X_COORD = -1,
    Y_COORD = -2
} ParseError;

/**
 * A compound data type for handling
 * points more eloquently.
 */

```

```

typedef struct Point
{
    int x;
    int y;
} Point;

/**
 * Calculates the distance between two points.
 * @param p1 One pointy boi.
 * @param p2 The other pointy boi.
 */
double distance_between(Point p1, Point p2);

/**
 * Creates a new point and places it at
 * the origin. Use this function to
 * initialize Point variables.
 */
Point origin();

/**
 * Parses an integer number literal starting
 * at the supplied pointer. If <i>start</i> is
 * a null pointer, 0 is returned.
 * @param start A pointer to the first character which
 * is part of the number literal.
 * @param length A (nullable) pointer to the variable
 * which will be overridden with the length
 * of the literal parsed from the supplied string.
 * If the number literal is invalid, length is set
 * to 0.
 */
int parse_number_literal(const char *const start, size_t *length);

/**
 * Parses a string and generates a point from it
 * if it matches the format "%d,%d".
 * @param arg The string holding the point literal.
 * @param error A (nullable) pointer to the variable
 * which will be overridden with an error code
 * if any errors occur.
 */
Point parse_point_literal(const char *const arg, ParseError *error);

#endif //UE01_CLI_PARSER_H

cli_parser.c
#include "cli_parser.h"
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

#define nullptr 0

double distance_between(Point p1, Point p2)
{
    int x_dist = p2.x - p1.x;
    int y_dist = p2.y - p1.y;
    return sqrt(x_dist * x_dist + y_dist * y_dist);
}

Point origin() {
    Point p = {0, 0};
    return p;
}

```

```

}

int parse_number_literal(const char *const start, size_t *length)
{
    int dest = 0;
    if (start != nullptr) {
        size_t cpy_length = 0;
        // NumberLiteral = [ '-' ] Digit { Digit } .
        if (start[cpy_length] == '-') {
            cpy_length = 1;
        }
        // cpy_length should span the whole Literal!
        while (isdigit(start[cpy_length])) {
            ++cpy_length;
        }
        if (length != nullptr) {
            if (cpy_length == 1 && start[0] == '-') {
                *length = 0; // indicate error
            } else {
                *length = cpy_length;
            }
        }
        char *temp = (char *) malloc(cpy_length + 1);
        strncpy(temp, start, cpy_length);
        dest = atoi(temp); // NOLINT(cert-err34-c)
        free(temp);
    }
    return dest;
}

Point parse_point_literal(const char *const arg, ParseError *error)
{
    Point p = origin();
    if (arg != nullptr) {
        if (error != nullptr) {
            *error = NONE;
        }
        size_t literal_length = 0;
        p.x = parse_number_literal(arg, &literal_length);
        if (error != nullptr && literal_length == 0) {
            *error = X_COORD;
        }
        // coordinates are separated by a comma so
        // the next number literal starts at the length
        // of the last number + 1!
        p.y = parse_number_literal(arg + literal_length + 1, &literal_length);
        if (error != nullptr && literal_length == 0) {
            *error = Y_COORD;
        }
    }
    return p;
}

```

Tests

Anmerkung: Alle Tests wurden mit GeoGebra auf Richtigkeit überprüft.

```

PS C:\Users\Niklas Vest\Documents\FH\fh-hgb-ws1819\swo\ue01\src\poly> .\polycirc.exe
polycirc needs at least 3 points to calculate the circumference!
Program usage: polycirc <point> <point> <point> [<point> ...]
- <point> = <x>:<y>.
- <x> and <y> can be an arbitrary integer number.

```

```
PS C:\Users\Niklas Vest\Documents\FH\fh-hgb-ws1819\swo\ue01\src\poly> .\polycirc.exe 1:2 2:3
polycirc needs at least 3 points to calculate the circumference!
Program usage: polycirc <point> <point> <point> [<point> ...]
- <point> = <x>:<y>.
- <x> and <y> can be an arbitrary integer number.
```

```
PS C:\Users\Niklas Vest\Documents\FH\fh-hgb-ws1819\swo\ue01\src\poly> .\polycirc.exe 1:2 2:3 :4
Point 1: (1,2)
Point 2: (2,3)
The x coordinate on point 3 is missing.
```

Abbildung 1 Durch die spezielle Parser-Funktion können detaillierte Fehler ausgegeben werden.

```
PS C:\Users\Niklas Vest\Documents\FH\fh-hgb-ws1819\swo\ue01\src\poly> .\polycirc.exe 1:2 2:3 5:4
Point 1: (1,2)
Point 2: (2,3)
Point 3: (5,4)
The total circumference is 9.048627
```

```
PS C:\Users\Niklas Vest\Documents\FH\fh-hgb-ws1819\swo\ue01\src\poly> .\polycirc.exe -3.4 13:3 -9$-8 100#100
Point 1: (-3,4)
Point 2: (13,3)
Point 3: (-9,-8)
Point 4: (100,100)
The total circumference is 334.873046
```

Abbildung 2 Unterschiedliche Separatoren möglich