

Inhaltsverzeichnis

1.	NOVA Rechner	2
1.1.	Lösungsidee	2
1.2.	Code	2
1.3.	Testfälle	3
1.3.1.	Terminalausgabe	3
1.3.2.	Gegenüberstellung	3
2.	Umfang eines Polygons	4
2.1.	Lösungsidee	4
2.2.	Code	4
2.3.	Testfälle	6
2.3.1.	Ohne Parameterübergaben	6
2.3.2.	Gültige Eingabe 1	6
2.3.3.	Gültige Eingabe 2	6
2.3.4.	Fehlerbehandlung 1	7
2.3.5.	Fehlerbehandlung 2	7

1. NOVA Rechner

1.1. Lösungsidee

Der NOVA Rechner wird immer mit zwei Parametern aufgerufen. Der erste Parameter gibt die Kraftstoffart und der zweite Parameter gibt den durchschnittlichen Verbrauch auf 100km an. Wenn man nun auch den Namen des Programms an der Stelle 0 des Parametervektors berücksichtigt so muss das Programm 3 Parameter besitzen. Um die Kraftstoffart (Diesel und Benzin) zu unterscheiden wird für Diesel die Zahl 1 und für Benzin die Zahl 2 herangezogen. Um die NOVA in Prozent laut Angabe zu berechnen muss abhängig von der Kraftstoffart in die richtige Formel eingesetzt werden.

Im Sinn eines guten Programmierstils wurden einige Code Abschnitte in Funktionen ausgelagert. (Line, PrintErr, PrintRes, NOVA)

1.2. Code

```
nova.c

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

// separator
void Line(void) {
    for(int i = 0; i < 50; i++) printf("-");
    printf("\n");
    return;
}

// error message handling
void PrintErr(char msg[]) {
    printf("<< ERROR (%s) >>\n",msg);
}

// print formatted result to console
void PrintRes(char fuel[], double nova) {
    printf("FUEL: %s\n", fuel);
    printf("NOVA: %4.2f%%\n", nova);
}

// returns calculated NOVA in %
// factor depends on the fuel
double NOVA(double consumption, int factor){
    return (consumption - factor) * 2;
}

// 1.param: fuel
// 2.param: average consumption
int main(int argc, char *argv[]) {
    Line();
    if(argc == 3) {
        if(atoi(argv[1]) == 1){
```

```

    PrintRes("Diesel", NOVA(atof(argv[2]),2));
} else if(atoi(argv[1]) == 2) {
    PrintRes("Benzin", NOVA(atof(argv[2]),3));
} else {
    PrintErr("please check your input parameters");
}
} else {
    PrintErr("please enter parameters: fuel consumption");
}
Line();
return 0;
}

```

1.3. Testfälle

1.3.1. Terminalausgabe

Der erste Screenshot zeigt eine valide Parametereingabe. Der zweite Screenshot zeigt eine Fehlerbehandlung (zu wenige Parameter).

```

nova — -bash — 80x6
[Michaels-MacBook-Pro-3:nova michaelneuhold$ ./nova 2 5.5
-----
FUEL: Benzin
NOVA: 5.00%
-----
Michaels-MacBook-Pro-3:nova michaelneuhold$

```

```

nova — -bash — 80x5
[Michaels-MacBook-Pro-3:nova michaelneuhold$ ./nova
-----
<< ERROR (please enter parameters: fuel consumption) >>
-----
Michaels-MacBook-Pro-3:nova michaelneuhold$

```

1.3.2. Gegenüberstellung

Auto Bezeichnung	Kraftstoff	Herstellerverbrauch	Verbrauch	NOVA
Audi - A4 - A4 Avant Quattro 2.0 TDI Ultra S-Tronic	Diesel	6,5	11,91	18%
Volvo - V40 - Dynamic	Diesel	4,5	4,46	4,92%
Volkswagen - Up! - Take Up	Benzin	4,5	4,56	3,12%
Skoda - Octavia - Combi 1.5 TSI ACT	Benzin	4,9	5,70	5,4%
Porsche - 911 - Carrera S Cabrio	Benzin	9,1	9,28	12%

2. Umfang eines Polygons

2.1. Lösungsidee

Das Programm „Poly“ benötigt min. 6 Parameter (3 Koordinatenpaare → Dreieck). Jedoch muss bei der Eingabe darauf geachtet werden, dass die Anzahl der Parameter durch 2 teilbar ist, ansonsten sind die Koordinatenpaare unvollständig. Sollten diese Auflagen nicht erfüllt sein, so werden entsprechende Fehlermeldungen ausgegeben. Um den Umfang eines Polygons zu berechnen wird eine Schleife eingesetzt, in der immer zwei Koordinatenpaare in die entsprechende Formel:

$$U = \sum_{i=1}^n \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$$

eingesetzt werden. Der Umfang wird immer um das Ergebnis der Formel erhöht, bis die Abbruchbedingung der Schleife erreicht ist. Als letzter Schritt muss noch das Vieleck geschlossen werden. Um dies zu gewährleisten ist es notwendig den ersten und den letzten Punkt nochmals in die Formel einzusetzen.

Im Sinn eines guten Programmierstils wurden einige Code-Abschnitte in Funktionen ausgelagert. (Line, PrintErr, PrintInfo, PrintCoo)

2.2. Code

```
poly.c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

// separator
void Line(void) {
    for(int i = 0; i < 50; i++) printf("-");
    printf("\n");
    return;
}

// error message handling
void PrintErr(char msg[]) {
    printf("<< ERROR (%s) >>\n", msg);
    return;
}

// no parameters --> print info
void PrintInfo(char name[]) {
    Line();
    printf("How to use %s\n", name);
    printf("please add at least 3 coordinates x & y to the parameter list\n");
    printf("1. example: %s 1 1 -1 1 0 -1 \n", name);
    printf("2. example: %s x y x y x y ... \n", name);
```

```

Line();
return;
}

// print coordinates formatted
void PrintCoo(int x, int y) {
    printf("[ x=%+d | y=%+d ]\n", x, y);
}

int main(int argc, char *argv[]) {
    int x_co, y_co, x_co_plus1, y_co_plus1;
    double u = 0;

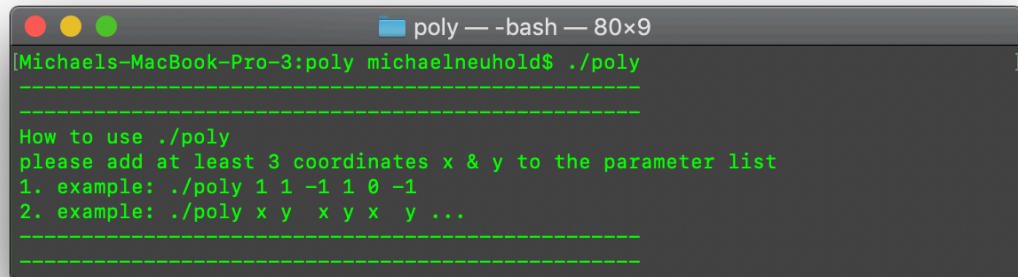
    Line();
    if(argc >= 7) {
        if((argc-1)%2==0) {
            for(int i = 3; i < argc; i=i+2) {
                x_co = atoi(argv[i-2]);
                y_co = atoi(argv[i-1]);
                x_co_plus1 = atoi(argv[i]);
                y_co_plus1 = atoi(argv[i+1]);
                PrintCoo(x_co, y_co);
                // calculate u iterativ
                u += sqrt((pow((x_co_plus1 - x_co), 2) + pow(y_co_plus1 - y_co, 2)));
            }
            // print last coordinate
            PrintCoo(atoi(argv[argc-2]), atoi(argv[argc-1]));
            // close polygon
            u += sqrt((pow((atoi(argv[argc-2]) - atoi(argv[argc-4])), 2) + pow(atoi(argv[argc-3]) - atoi(argv[argc-1]), 2)));
            Line();
            printf("U: %f\n", u);
        } else {
            PrintErr("odd number of coordinates");
        }
    } else if (argc == 1) {
        PrintInfo(argv[0]);
    } else {
        PrintErr("too few arguments");
    }
    Line();
    return 0;
}

```

2.3. Testfälle

2.3.1. Ohne Parameterübergaben

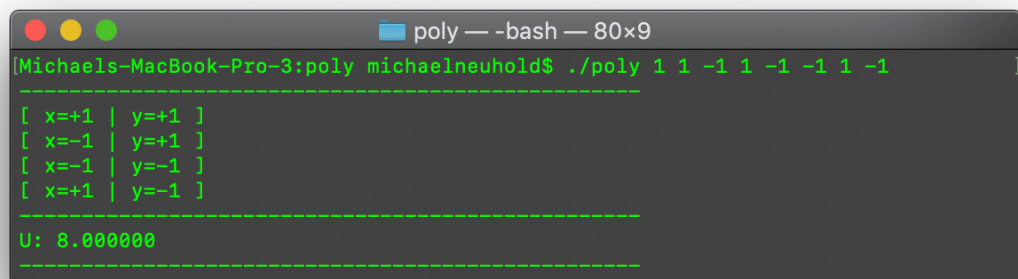
Dieser Testfall zeigt das Verhalten des Programms, wenn der Benutzer keine Koordinaten übergibt. Es wird eine kurze Hilfestellung mit Beispielen ausgegeben.



```
poly — -bash — 80x9
[Michaels-MacBook-Pro-3:poly michaelneuhold$ ./poly]
-----
How to use ./poly
please add at least 3 coordinates x & y to the parameter list
1. example: ./poly 1 1 -1 1 0 -1
2. example: ./poly x y x y x y ...
-----
```

2.3.2. Gültige Eingabe 1

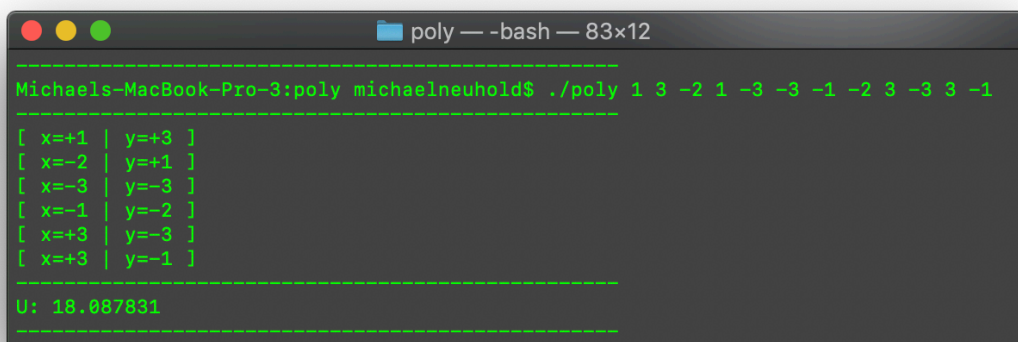
Dieser Testfall zeigt das Verhalten des Programms bei der Eingabe von 4 Koordinatenpaaren. Es wird eine Ausgabe generiert, die nochmals die Parameter formatiert ausgibt und zeitgleich den Umfang berechnet und repräsentiert.



```
poly — -bash — 80x9
[Michaels-MacBook-Pro-3:poly michaelneuhold$ ./poly 1 1 -1 1 -1 1 -1]
-----
[ x=+1 | y=+1 ]
[ x=-1 | y=+1 ]
[ x=-1 | y=-1 ]
[ x=+1 | y=-1 ]
-----
U: 8.000000
-----
```

2.3.3. Gültige Eingabe 2

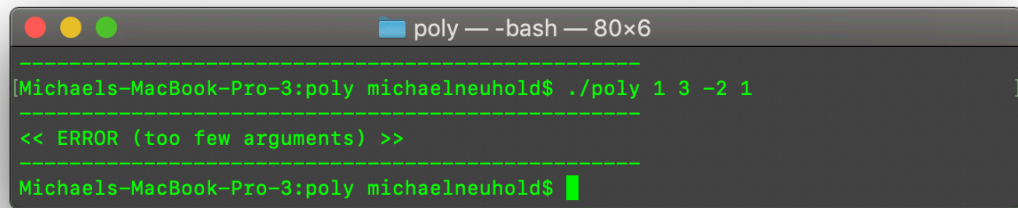
Dieser Testfall zeigt eine Benutzereingabe mit mehr Koordinatenpaaren.



```
poly — -bash — 83x12
-----
Michaels-MacBook-Pro-3:poly michaelneuhold$ ./poly 1 3 -2 1 -3 -3 -1 -2 3 -3 3 -1
-----
[ x=+1 | y=+3 ]
[ x=-2 | y=+1 ]
[ x=-3 | y=-3 ]
[ x=-1 | y=-2 ]
[ x=+3 | y=-3 ]
[ x=+3 | y=-1 ]
-----
U: 18.087831
-----
```

2.3.4. Fehlerbehandlung 1

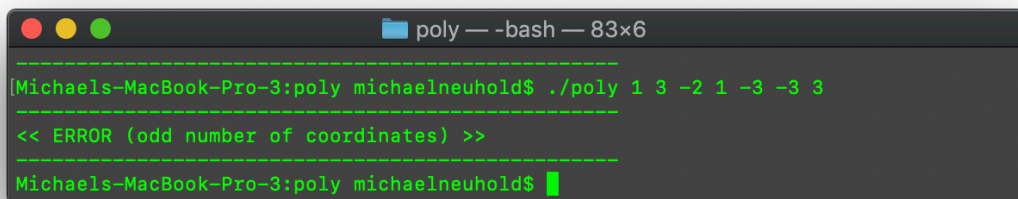
Sollte der Benutzer zu wenige Argumente dem Programm übergeben, so wird folgende Fehlermeldung ausgegeben.



```
poly — -bash — 80x6
[Michaels-MacBook-Pro-3:poly michaelneuhold$ ./poly 1 3 -2 1 ]
<< ERROR (too few arguments) >>
Michaels-MacBook-Pro-3:poly michaelneuhold$
```

2.3.5. Fehlerbehandlung 2

Sollte der Benutzer eine ungerade Zahl an Parameter (in case ($\text{argc} > 6$)) dem Programm übergeben, so wird folgende Fehlermeldung ausgegeben.



```
poly — -bash — 83x6
[Michaels-MacBook-Pro-3:poly michaelneuhold$ ./poly 1 3 -2 1 -3 -3 3 ]
<< ERROR (odd number of coordinates) >>
Michaels-MacBook-Pro-3:poly michaelneuhold$
```