SWO3	**Übung zu Softwareentwicklung mit klassischen Sprachen und Bibliotheken 3**	**WS 2021/22, ÜZ 2**

☐ Gruppe M. Hava

☒ Gruppe J. Heinzelreiter Name: Angelos Angelis Aufwand [h]: 10

☐ Gruppe P. Kulczycki Peer-Review von: _____

Beispiel	Lösungsidee (max. 100%)	Implement. (max. 100%)	Testen (max. 100%)
1 (100 P)	100%	100%	100%

Beispiel 1: Rechnen mit Polynomen (`src/poly/`)

Ein Polynom $P(x)$ mit $x \in \mathbb{R}$ vom Grad $m \in \mathbb{N}_0$ hat die allgemeine Form

$$P(x) = p_0 \cdot x^0 + p_1 \cdot x^1 + p_2 \cdot x^2 + \cdots + p_m \cdot x^m = \sum_{i=0}^{m} p_i \cdot x^i$$

wobei $p_i \in \mathbb{R}$ mit $0 \leq i \leq m$ die jeweiligen Koeffizienten der Potenzen von x sind. Ein Polynom vom Grad m kann als C-Array der Länge $m + 1$, welches die Werte der Koeffizienten aufnimmt, dargestellt werden.

Implementieren Sie ein C11-Programm `polynomial`, das die folgenden Funktionalitäten enthält:

a) Schreiben Sie eine Funktion `poly_print`, die ein Polynom auf der Konsole (am Bildschirm) ausgibt. Die Schnittstelle von `poly_print` muss wie folgt aussehen:

```
void poly_print (double const p [], int const m);
```

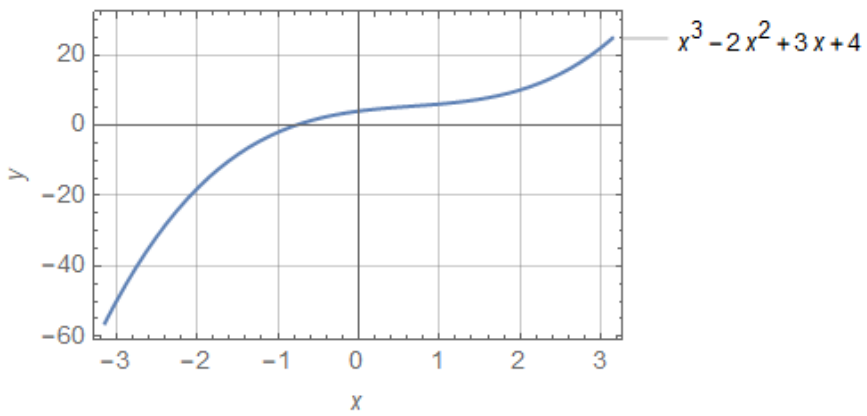
Ein Beispiel: Für das Polynom $P(x) = 4 + 3 \cdot x - 2 \cdot x^2 + x^3$ muss poly_print folgendes ausgeben:

```
4 + 3*x - 2*x^2 + x^3
```

b) Schreiben Sie eine Funktion `poly_evaluate`, die ein Polynom an einer gegebenen Stelle $x \in \mathbb{R}$ ausrechnet. Die Schnittstelle von `poly_evaluate` muss wie folgt aussehen:

```
double poly_evaluate (double const p [], int const m, double const x);
```

Ein Beispiel: Für obiges Polynom ergibt sich ein ungefährer Wert von -56.1703, wenn man es an der Stelle $-\pi$ auswertet.



Aufgabe 1)

Lösungsidee:

Poly_print: Man geht mit einer FOR-Schleife alle Zahlen im Array durch, indem man die aktuelle Zahl in einer temporären Variablen („currentNumber“) speichert.
Um ein Polynom auszugeben, müssen 4 Bedingungen geprüft werden:

1. Ist die Zahl negativ?
    1. Falls ja, gib ein „–“ aus und multipliziere currentNumber mit „-1“
    2. Falls nein gib ein „+“ aus
2. Ist der Koeffizient gleich 1?
    1. Falls ja, gib ein „x^exponent“ aus
    2. Falls nein gib die Zahl und „* x^exponent“ aus
3. Ist der Exponent gleich 1?
    1. Falls ja, gib einfach ein „x“ aus
    2. Falls nein gib ein „x^exponent“ aus
4. Ist der Koeffizient = 0?
    1. Falls ja, gib eine „0“ aus (Man kann auch gar nichts ausgeben aber ich finde es übersichtlicher, wenn man die 0er ausgibt).

Poly_evaluate: Man geht mittels einer FOR-Schleife alle Zahlen vom Polynom durch und macht Folgendes:

1. Berechne x^exponent und speicher das Zwischenergebnis in einer Variable powerRes(power result).
2. Multipliziere den Koeffizient mit powerRes und addiere das Ergebnis in der Variable temp.
3. Gib nach der FOR-Schleife temp als Ergebnis der Funktion aus;

Poly_add: Zu Beginn wird r mit den Werten des größten Polynoms initialisiert. Mit einer FOR-Schleife die bis zur Länge des kleinsten Polynoms läuft werden die Zahlen an der Stelle i von den zwei Polynomen in dem Array r an der Stelle i gespeichert.
Poly_mult: Man multipliziert zwei Polynome mithilfe zwei FOR-Schleifen wobei die eine in der anderen verschachtelt ist. Die erste läuft bis m+1 erreicht wurde und die andere bis n+1 erreicht wurde. An der Stelle i+j von r speichert man dann das Ergebnis von der Multiplikation p[i] * q[j].
Poly_mult_fast: Man braucht 13 Arrays um Zwischenergebnisse zu speichern.

- Zu Beginn werden die zwei Arrays in jeweils zwei Hälften aufgeteilt
- Dann multipliziert man pl und ql um hl zu errechnen und pr und qr um hr zu errechnen
- Hm wird errechnet, indem man pl+pr und ql+qr rechnet und die Ergebnisse jeweils miteinander multipliziert.
- Dann wird (*Hm(x)-hl(x)-hr(x))* errechnet mithilfe einer FOR-Schleife gerechnet
- Dann wird hmlr*x^(m/2)+1 ebenfalls mithilfe einer FOR-Schleife gerechnet
- Dann wird *hr*x^m+1* ebenfalls mithilfe einer FOR-Schleife gerechnet

- Dann werden die letzten drei Ergebnisse addiert, um das Ergebnis der Multiplikation zu berechnen. Dieser wird dann in r gespeichert und 2*m wird als Ergebnis der Funktion ausgegeben.

Code:

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

#define M_PI 3.14159265358979323846
#define MAX_ARRAY_LENGTH 20

void init_array(double arr[]){
   for (int i = 0; i < MAX_ARRAY_LENGTH; i++)
   {
      arr[i]=0;
   }
}

void poly_print(double const p [],int const m){
   //printf("%.2f + ",p[0]);
   int i;
   double currentNumber;
   currentNumber = p[0]; //print first number
   printf("%.2f",currentNumber);
   for (i = 1; i < m+1; i++)
   {
      currentNumber = p[i];
      if (currentNumber < 0) // check for negative number
      {
         printf(" - ");
         currentNumber = currentNumber * -1;
      }
      else // Positive number
      {
         printf(" + ");
      }
      if (currentNumber == 1) // coefficient = 1
      {
         if (i == 1) //exponent = 1
         {
            printf("x");
         }
         else //exponent > 1
```

```
         {
            printf("x^%d",i);
         }
      }
      else if (currentNumber > 1)// coefficient > 1
      {
         if (i == 1) //exponent = 1
         {
            printf("%.2f*x",currentNumber);
         }
         else // exponent > 1
         {
            printf("%.2f*x^%d",currentNumber,i);
         }
      }
      else
      {
         printf("0.00"); // coeffizient = 0; ALSO REMOVE THIS IF YOU DONT WANT
COEFFICIENTS   THAT ARE 0 TO BE PRINTED
      }

   }
   printf("\n");
}

double poly_evaluate(double const p[], int const m, double const x){
   double temp;
   double powerRes;
   temp = p[0];
   for (int i = 1; i < m+1; i++)
   {
      powerRes = x;
      for (int j = 1; j < i; j++) // Calculate the Exponent of X
      {
         powerRes = powerRes * x;
      }
      temp = temp + (p[i])*powerRes;
   }
   return temp;
}

int poly_add(double const p [], int const m,// Polynom P(x) vom Grad m
             double const q [], int const n,// Polynom Q(x) vom Grad n
             double       r []              // Polynom R(x) vom Grad max (m, n)
){
```

```
   int min = 0;
   int max = 0;
   if (n >= m) //Initialize r with the numbers of
   {           //the biggest polynomial

      for (int i = 0; i < n+1; i++)
      {
         r[i] = q[i];
      }
      max = n;
      min = m;
   }
   else
   {
      for (int i = 0; i < n+1; i++)
      {
         r[i] = p[i];
      }
      max = m;
      min = n;
   }
   for (int i = 0; i < min+1; i++) // add the polynomials until the Length of the
smallest polynomial
   {
      r[i] = p[i] + q[i];
   }
   return max;
}

int poly_mult(double const p [], int const m,   // Polynom P(x) vom Grad m
              double const q [], int const n,   // Polynom Q(x) vom Grad n
              double       r []                 // Polynom R(x) vom Grad m + n
)
{

   for (int i = 0; i < m+1; i++)
   {
      for (int j = 0; j < n+1; j++)
      {
         r[i+j]= r[i+j]+(p[i] * q[j]);
      }
   }
   return m+n;
}
```

```
int poly_mult_fast(double const p [], int const m,   // Polynom P(x) vom Grad m
                   double const q [], int const n,   // Polynom Q(x) vom Grad n==
m
                   double       r []                 // Polynom R(x) vom Grad 2 *
m
)
{
   if (m != n)
   {
      printf("Error occured. Pls check polynomial size (ERRORCODE: 1)\n");
      return EXIT_FAILURE;
   }
   if ((m+1)%2 != 0)
   {
      printf("Error occured. Pls check polynomial size (ERRORCODE: 2)\n");
      return EXIT_FAILURE;
   }
   int const grad = ((m+1)/2)-1;

   double pl[MAX_ARRAY_LENGTH];//the divided polynomials
   double pr[MAX_ARRAY_LENGTH];
   double ql[MAX_ARRAY_LENGTH];
   double qr[MAX_ARRAY_LENGTH];

   double hl[MAX_ARRAY_LENGTH];//auxiliary variables
   double hr[MAX_ARRAY_LENGTH];
   double hm[MAX_ARRAY_LENGTH];

   double hm1[MAX_ARRAY_LENGTH];// with the help of hm1 and hm2 we can calculate
hm
   double hm2[MAX_ARRAY_LENGTH];

   double hmlr[MAX_ARRAY_LENGTH];// (Hm(x)-hl(x)-hr(x))
   double hmlr2[MAX_ARRAY_LENGTH];// hmlr*x^(m/2)+1

   double hr2[MAX_ARRAY_LENGTH];// hr*x^m+1
   double end1[MAX_ARRAY_LENGTH];// hm + hmlr2

   for (int v = 0; v < MAX_ARRAY_LENGTH; v++)// init all arrays
   {
      pl[v]=0;
      pr[v]=0;
      ql[v]=0;
      qr[v]=0;
      hl[v]=0;
```

```
      hr[v]=0;
      hm[v]=0;
      hm1[v]=0;
      hm2[v]=0;
      hmlr[v]=0;
      hmlr2[v]=0;
      hr2[v]=0;
      end1[v]=0;x
   }

   for (int i = 0; i < grad+1; i++)//divide the 2 arrays into 2 arrays each
   {
      pl[i] = p[i];
      pr[i] = p[grad+1+i];
      ql[i] = q[i];
      qr[i] = q[grad+1+i];
   }

   int hlgrad = poly_mult(pl,grad,ql,grad,hl);// we need hlgrad if we want to
display intermediate results
   int hrgrad = poly_mult(pr,grad,qr,grad,hr);

   poly_add(pl,grad,pr,grad,hm1);
   poly_add(ql,grad,qr,grad,hm2);
   int hmgrad = poly_mult(hm1,grad,hm2,grad,hm);

   for (int j = 0; j < hmgrad+1; j++)// (Hm(x)-hl(x)-hr(x))
   {
      hmlr[j] = hm[j] - hl[j] - hr[j];
   }

   int k = 0;
   for (int z = (m+1)/2; z < hmgrad+((m+1)/2)+1; z++)// hmlr*x^(m/2)+1
   {
      hmlr2[z] = hmlr[k];
      k++;
   }
   k = 0;
   for (int y = (2*m)-hrgrad; y < 2*m+1; y++)// hr*x^m+1
   {
      hr2[y] = hr[k];
      k++;
   }

   int end1grad = poly_add(hl,hlgrad,hmlr2,hmgrad+((m+1)/2),end1);
```

```
    poly_add(end1,end1grad,hr2,2*m,r);
    return 2*m;
}

void test_print(){
    printf("Test exponent = 1\n");
    double test1[] = {-2,-10};
    poly_print(test1,1);

    printf("Test if it can print negative numbers\n");
    double test2[] = {-2,-10,20,-32,-1,2};
    poly_print(test2,5);

    printf("Test coefficient = 1\n");
    double test3[] = {1,-1,1-1,1,-1,1,-1,1,-1};
    poly_print(test3,8);

    printf("Test coefficient = 0\n");
    double test4[] = {0,0,0,0,0,0,0,0};
    poly_print(test4,7);
}

void test_evaluate(){
    printf("Test coefficient = 0\n");
    double test1[] = {0,0,0,0};
    double test1res = poly_evaluate(test1,4,3);
    printf("%.2f\n",test1res);

    printf("Test x=0\n");
    double test2[] = {-2,-10,20,-32,-1,2};
    double test2res = poly_evaluate(test1,5,0);
    printf("%.2f\n",test2res);

    printf("Test x=pi\n");
    double test3[] = {4,3,-2,1};
    double test3res = poly_evaluate(test3,3,-M_PI);
    printf("%.2f\n",test3res);
}

void test_add(){
    double result[MAX_ARRAY_LENGTH];
    init_array(result);
    printf("Test pol1 degree > pol2 degree\n");
    double pol1[] = {1,1,3,-4};
    double pol2[] = {1,2,-5,-3,0,-2};
```

```
    int degree = poly_add(pol1,3,pol2,5,result);
    poly_print(result,degree);

    printf("Test pol2 degree > pol1 degree\n");
    init_array(result);
    degree = poly_add(pol2,5,pol1,3,result);
    poly_print(result,degree);

    printf("Test adding 0\n");
    double pol3[] = {1,1,0,0};
    double pol4[] = {0,0,0,0,0,0};
    init_array(result);
    degree = poly_add(pol3,3,pol4,5,result);
    poly_print(result,degree);
}

void test_mult(){
    double result[MAX_ARRAY_LENGTH];
    init_array(result);
    printf("Test given example\n");
    double pol1[] = {1,1,3,-4};
    double pol2[] = {1,2,-5,-3,0,-2};
    int degree = poly_mult(pol1,3,pol2,5,result);
    poly_print(result,degree);

    printf("Given example but q*p instead of p*q \n");
    init_array(result);// set everything to 0 again
    degree = poly_mult(pol2,5,pol1,3,result);
    poly_print(result,degree);

    printf("Test multiplication by 0\n");
    double pol3[] = {0,0,0,0};
    double pol4[] = {1,2,-5,-3,0,-2};
    double result2[MAX_ARRAY_LENGTH];
    init_array(result2);
    degree = poly_mult(pol4,5,pol3,3,result2);
    poly_print(result2,degree);
}

void test_mult_fast(){
    double result[MAX_ARRAY_LENGTH];
    init_array(result);
    printf("Test given example\n");
    double pol1[] = {1,1,3,-4};
    double pol2[] = {1,2,-5,-3};
```

```
    int degree = poly_mult_fast(pol1,3,pol2,3,result);
    poly_print(result,degree);

    printf("Given example but q*p instead of p*q\n");
    init_array(result);// set everything to 0 again
    degree = poly_mult_fast(pol2,3,pol1,3,result);
    poly_print(result,degree);

    printf("Test different polynomial degrees\n");
    init_array(result);// set everything to 0 again
    degree = poly_mult_fast(pol2,5,pol1,3,result);

    printf("Test odd m+1\n");
    init_array(result);// set everything to 0 again
    degree = poly_mult_fast(pol2,6,pol1,6,result);

    printf("Test multiplication by 0\n");
    double pol3[] = {0,0,0,0};
    double pol4[] = {1,2,-5,-3};
    double result2[MAX_ARRAY_LENGTH];
    init_array(result2);
    degree = poly_mult_fast(pol4,3,pol3,3,result2);
    poly_print(result2,degree);
}

int main(){
    printf("-----------TESTING poly_print FUNCTION-----------\n");
    test_print();
    printf("\n");
    printf("-----------TESTING poly_evaluate FUNCTION-----------\n");
    test_evaluate();
    printf("\n");
    printf("-----------TESTING poly_add FUNCTION-----------\n");
    test_add();
    printf("\n");
    printf("-----------TESTING poly_mult FUNCTION-----------\n");
    test_mult();
    printf("\n");
    printf("-----------TESTING poly_mult_fast FUNCTION-----------\n");
    test_mult_fast();
}
```

Tests:

Poly_print: wird mittels der test_print Funktion getestet.

```
----------TESTING poly_print FUNCTION----------
Test exponent = 1
-2.00 - 10.00*x
Test if it can print negative numbers
-2.00 - 10.00*x + 20.00*x^2 - 32.00*x^3 - x^4 + 2.00*x^5
Test coefficient = 1
1.00 - x + 0.00 + x^3 - x^4 + x^5 - x^6 + x^7 - x^8
Test coefficient = 0
0.00 + 0.00 + 0.00 + 0.00 + 0.00 + 0.00 + 0.00 + 0.00
```

Poly_evaluate: wird mittels der test_evaluate Funktion getestet:

```
----------TESTING poly_evaluate FUNCTION----------
Test coefficient = 0
0.00
Test x=0
0.00
Test x=pi
-56.17
```

Poly_add: wird mittels der test_add Funktion getestet:

```
----------TESTING poly_add FUNCTION----------
Test pol1 degree > pol2 degree
2.00 + 3.00*x - 2.00*x^2 - 7.00*x^3 + 0.00 - 2.00*x^5
Test pol2 degree > pol1 degree
2.00 + 3.00*x - 2.00*x^2 - 7.00*x^3 + 0.00 + 0.00
Test adding 0
1.00 + x + 0.00 + 0.00 + 0.00 + 0.00
```

Poly_mult: wird mittels der test_mult Funktion getestet:

```
----------TESTING poly_mult FUNCTION----------
Test given example
1.00 + 3.00*x + 0.00 - 6.00*x^3 - 26.00*x^4 + 9.00*x^5 + 10.00*x^6 - 6.00*x^7 + 8.00*x^8
Given example but q*p instead of p*q
1.00 + 3.00*x + 0.00 - 6.00*x^3 - 26.00*x^4 + 9.00*x^5 + 10.00*x^6 - 6.00*x^7 + 8.00*x^8
Test multiplication by 0
0.00 + 0.00 + 0.00 + 0.00 + 0.00 + 0.00 + 0.00 + 0.00 + 0.00
```

Poly_mult_fast: wird mittels der test_mult_fast Funktion getestet:

```
-----------TESTING poly_mult_fast FUNCTION-----------
Test given example
1.00 + 3.00*x + 0.00 - 6.00*x^3 - 26.00*x^4 + 11.00*x^5 + 12.00*x^6
Given example but q*p instead of p*q
1.00 + 3.00*x + 0.00 - 6.00*x^3 - 26.00*x^4 + 11.00*x^5 + 12.00*x^6
Test different polynomial degrees
Error occured. Pls check polynomial size (ERRORCODE: 1)
Test odd m+1
Error occured. Pls check polynomial size (ERRORCODE: 2)
Test multiplication by 0
0.00 + 0.00 + 0.00 + 0.00 + 0.00 + 0.00 + 0.00
```