# Rechnen mit Polynomen

### Allgemeine Hinweise:

Die verwendeten Arrays werden mit der konstanten **ARRAY\_SIZE** auf eine fixe Größe von 100 beschränkt.

## Poly\_Print

### Lösungsidee:

In einer FOR-Schleife wird das übergebene Array von 0 bis m durchlaufen. Im ersten Durchlauf (i==0) wird nur die Zahl ausgegeben. Beim zweiten Durchgang (i==1) wird „Zahl \* x“ ausgegeben. Bei den restlichen Durchläufen wird „Zahl \* x ^ i“ ausgegeben. Innerhalb jeder dieser Möglichkeiten gibt es noch die Überprüfung, ob eine Zahl größer oder kleiner 0 ist. Ist die Zahl genau 0, wird nichts ausgegeben.

### Code:

void poly\_print(double const p[], int const m) {

    printf("P(x) = ");

    for (int i = 0; i < m; i++) {

        if (i == 0) {

            if (p[i] != 0)

                printf("%.2f", p[i]);

        }

        else if (i == 1) {

            if (p[i] > 0)

                printf(" + %.2f\*x", p[i]);

            else if (p[i] < 0)

                printf(" - %.2f\*x", p[i] \* - 1);

        } else {

            if (p[i] > 0)

                printf(" + %.2f\*x^%d", p[i], i);

            else if (p[i] < 0)

                printf(" - %.2f\*x^%d", p[i] \* - 1, i);

        }

    }

    printf("\n");

}

### Testfälle:

Testfälle mit den folgenden Arrays:

double a1[] = {1, 1, 3, -4, 0, 0, 0, 0, 0, 0, 0, 0, 0};

double a2[] = {1, 2, -5, -3, 6, 3, 15, 9115, 5, -3654, 20, 11};

double a3[] = {1, 1, 3, 0, 0, 0};

double a4[] = {0, 0, 0, 0, 0, 0};

## Poly\_evaluate

### Lösungsidee:

Das übergebene Polynom wird von 0 bis m in einer FOR-Schleife durchlaufen. In jedem durchlauf wird p[i] mit xi multipliziert und in einer Summenvariable aufsummiert. Diese Variable wird auch zurückgegeben.

### Code:

double poly\_evaluate(double const p[], int const m, double const x) {

    double total = 0;

    for (int i = 0; i < m; i++) {

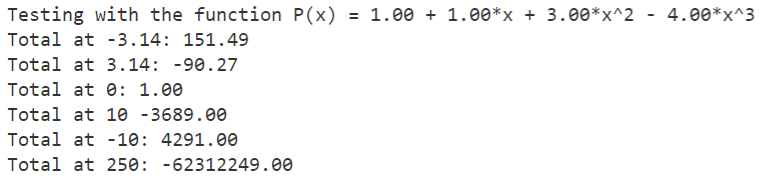
        total += p[i] \* pow(x, i);

    }

    return total;

}

### Testfälle:



# Poly\_add

### Lösungsidee:

Zu Beginn wird die Größe des größeren Arrays in der Variable **max** gespeichert. Eine FOR-Schleife läuft dann von 0 bis **max** und summiert die übergebenen Polynome an der aktuellen Stelle auf. Annahme: Jedes übergebene Array wird bis zur Stelle **max** mit 0 Initialisiert und kann daher keine ungewünschten Werte liefern.

### Code:

int poly\_add(double const p[], int const m, double const q[], int const n,

double r[]) {

    int max = m > n ? m : n;

    for (int i = 0; i < max; i++) {

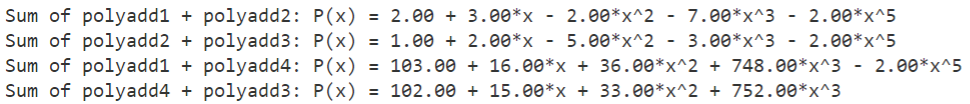
        r[i] = p[i] + q[i];

    }

    return 0;

}

### Testfälle:



Für Testfälle verwendete Polynome:

double polyadd1[] = {1, 1, 3, -4, 0};

double polyadd2[] = {1, 2, -5, -3, 0, -2};

double polyadd3[] = {0, 0, 0, 0, 0, 0, 0, 0};

double polyadd4[] = {102, 15, 33, 752, 0, 0};

Bei jeder Rechnung mit **polyadd3** muss das Ergebnis gleich der anderen Eingabe sein. (Siehe 2. & 4.)

# Poly\_mult

### Lösungsidee:

Das Ausgabearray wird bis zur Stelle m + n mit 0 initialisiert. Zwei verschachtelte FOR-Schleifen laufen dann durch die übergebenen Arrays durch und multiplizieren jede Stelle von p mit jeder Stelle in q. Zurückgegeben wird der Grad des Ergebnispolynoms. Dieser ergibt sich aus m + n. Da diese aber die Größe der Arrays angeben und dieser immer 1 mehr ist als der Grad des Polynoms, muss von dieser Zahl noch 2 subtrahiert werden.

### Code:

int poly\_mult(double const p[], int const m, double const q[], int const n,

double r[]) {

    int i;

    for (i = 0; i < m + n; i++)

        r[i] = 0;

    for (i = 0; i < m; i++) {

        for (int j = 0; j < n; j++)

            r[i+j] += p[i] \* q[j];

    }

    return m + n - 2;

}

### Testfälle:

Für die Testfälle verwendete Arrays:

double polymult1[] = {1, 1, 3, -4, 0, 0};

double polymult2[] = {1, 2, -5, -3, 0, -2};

double polymult3[] = {1, 0, 0, 0, 0, 0, 0};

double polymult4[] = {102, 15, 33, 752, 314};

Bei Multiplikationen mit **polymult3** wird das ursprüngliche Array ausgegeben.

# Poly\_mult\_fast

### Lösungsidee:

**Anmerkungen**: Diese Funktion funktioniert nur bei *m % 2 == 0*, also Polynomen mit ungeradem Grad. Weiters müssen die Grade der Polynome gleich sein, daher verwendet die Funktion nur **m.**

Diese Funktion wurde nach den Erklärungen in der Angabe rekursiv gelöst. Sind die in den Anmerkungen definierten Bedingungen nicht erfüllt, gibt die Funktion sofort -1 zurück. Zu Beginn wird die Hälfte von m bestimmt. Alles von 0 bis **half - 1** wird in die Variablen **Pl** und **Ql** gespeichert. Alles ab **half** bis **m** wird in **Pr** und **Qr** gespeichert. Mit der Funktion poly\_mult\_fast() werden dann die Hilfsvariablem **Hl** und **Hr** berechnet. Um die Hilfsvariable **Hm** zu berechnen werden zuerst **Pl** und **Pr** addiert und danach mit der Summe von **Ql** und **Qr** multipliziert. Entsprechend der Formel wird dann **Hm** mithilfe der neuen Funktion poly\_sub() (Eine Kopie von poly\_add() nur mit subtraktion) minus **Hl** und **Hr** gerechnet. Das Ergebnis dieser Rechnung wird dann mit **x^half** multipliziert. Dafür werden die Werte im Array um **half** stellen verschoben. Dieselbe Lösung wird auch für das Multiplizieren von **Hr** mit **x^m** verwendet. Die Ergebnisse dieser beiden Nebenrechnungen werden dann mit **Hl** addiert und der Grad des resultierenden Polynoms zurückgegeben.

Als Abbruchbedingung für die Rekursion wird zu Beginn der Funktion **m == 1** geprüft. Ist diese Bedingung erfüllt, berechnet die Funktion nur das Produkt der beiden übergebenen Funktionen an der Stelle 0.

### Code:

int poly\_mult\_fast(double const p[], int const m, double const q[], int const n, double r[]) {

    if (m != n)

        return -1;

    if (m != 1) {

        if (m % 2 != 0)

            return -1;

        double pl[ARRAY\_SIZE], pr[ARRAY\_SIZE], ql[ARRAY\_SIZE], qr[ARRAY\_SIZE],

hl[ARRAY\_SIZE], hr[ARRAY\_SIZE], hm[ARRAY\_SIZE], hm1[ARRAY\_SIZE],

hm2[ARRAY\_SIZE];

        int half = m / 2;

        for (int i = 0; i < half; i++) {

            pl[i] = p[i];

            ql[i] = q[i];

        }

        for (int i = half; i < m; i++) {

            pr[i - half] = p[i];

            qr[i - half] = q[i];

        }

        //Calculate helping variables hl, hr, hm

        poly\_mult\_fast(pl, half, ql, half, hl);

        poly\_mult\_fast(pr, m - half, qr, m - half, hr);

        poly\_add(pl, half, pr, m - half, hm1);

        poly\_add(ql, half, qr, m - half, hm2);

        poly\_mult\_fast(hm1, m - half, hm2, m - half, hm);

        //Calculate result

        double min[ARRAY\_SIZE];

        double hrx[ARRAY\_SIZE];

        for (int i = 0; i < m \* m; i++) {

            min[i] = 0;

            hrx[i] = 0;

        }

        poly\_sub(hm, m-half+1, hl, half+1, min);

        poly\_sub(min, m-half+1, hr, m-half+1, min);

        int mult = half;

        for (int j = m + mult; j >= mult; j--) {

            min[j] = min[j - mult];

        }

        for (int j = mult - 1; j >= 0; j--) {

            min[j] = 0;

        }

        int mul = m;

        for (int j = m + mul; j >= mul; j--)

            hrx[j] = hr[j - mul];

        for (int j = mul - 1; j >= 0; j--)

            hrx[j] = 0;

        poly\_add(hl, half, min, m + mult, r);

        poly\_add(r, m + mult, hrx, m + mul, r);

        return m + n - 2;

    } else {

        r[0] = p[0] \* q[0];

        return 1;

    }

}

### Testfälle:

Für die Testfälle verwendete Arrays:

double polymultfast1[] = {1, 1, 3, -4, 0, 0};

double polymultfast2[] = {1, 2, -5, -3, 0, 0};

double polymultfast3[] = {1, 0, 0, 0, 0, 0, 0};

double polymultfast4[] = {102, 15, 33, 752};

Beim letzten Testfall wurde polymultfast mit den Größen 1 und 4 aufgerufen. Deshalb wird eine Fehlermeldung ausgegeben und das Programm beendet.