SWO 3x Übung zu Softwareentwicklung mit klassischen Sprachen u. Bibliotheken 3 **WS 2014/15, Übung 2**

Abgabetermin: Sa in der KW 41

☐ **Gr. 1,** DI Franz Gruber-Leitner **Name** Roman Lumetsberger **Aufwand in h** 9

☒ **Gr. 2,** Dr. Erik Pitzer

Punkte _____ **Kurzzeichen Tutor / Übungsleiter** _____ / _____

1. Krafttraining (7 Punkte)

Ein Freund von Ihnen beschließt, nachdem er diesen Sommer nicht viel Zeit gehabt hat sich im Freien sportlich zu betätigen, ab jetzt regelmäßig das Fitnessstudio zu besuchen. Dort trainiert er mit einer Langhantel und verschiedenen Gewichten. Es fällt ihm allerdings ziemlich schwer für ein bestimmtes Gewicht, die richtige Kombination an Hantelscheiben zu finden. Da Sie sich vor kurzem mit dem Backtracking-Verfahren auseinandergesetzt haben, schlagen Sie vor ihm dabei zu helfen.

Schreiben Sie ein Programm `weights`, das als Parameter ein bestimmtes Gewicht bekommt und die richtige Kombination an Hantelscheiben ausgibt.

Die Gewichte und die Anzahl jeder verfügbaren Scheibe, sowie das Gewicht der Hantelstange selbst, können dazu in zwei globalen Feldern hartcodiert werden, wobei jeweils die Anzahl von Paaren von Scheiben angegeben wird, z.B.:

```
const double weights[] = { 0.5, 1.25, 2.5, 5, 10, 15, 20, 25 };
const int counts[]     = {   1,    1,   1, 3,  1,  1,  3,  1 };
const double bar = 20;
```

Falls es nicht möglich ist, das gewünschte Gewicht mit den verfügbaren Scheiben zu erreichen soll eine dementsprechende Meldung ausgegeben werden.

2. Mathematik mit Polynomen und Divide&Conquer (2 + 2 + 3 + 3 + 7 Punkte)

In der Mathematik könnte man sagen: „Fast alles ist ein Polynom“. Das beginnt bei den Zahlen eines beliebigen Zahlensystems und geht bis hin zu allen Funktionen, die entweder schon Polynome sind oder durch Interpolation mittels Polynomen näherungsweise dargestellt werden können. Deshalb wird es höchste Zeit, sich auch in der Programmierung mit Polynomen zu beschäftigen.

Ein Polynom p des Grades n hat die Form

$$p(x) = p_0 + p_1 x + p_2 x^2 + p_3 x^3 + \cdots + p_n x^n$$

und kann softwaretechnisch durch ein Feld, das die n+1 Koeffizienten p_0 bis p_n enthält, dargestellt werden.

Entwickeln Sie ein C-Programm, das Berechnungen mit Polynomen durchführt, und dazu folgende Funktionen zur Verfügung stellt:

a) Eine (Hilfs-)Funktion, die ein Polynom p in der üblichen Form (s.o.) anzeigt. Schnittstelle:

```
void printPoly(int n, double p[]);
```

Beispiel: Für $p(x) = 1 + x + 3x^2 - 4x^3$ soll `printPoly` folgendes ausgeben:

```
1 + 1x + 3x^2 - 4x^3
```

Tipp: Für die kompakte Formatierung von Fließkommazahlen können sie das Format `"%g"` verwenden.

b) Eine Funktion, die ein Polynom p an der Stelle x auswertet, also $p(x)$ berechnet und dafür natürlich das Horner-Schema verwendet. Schnittstelle:

```
double evalPoly(int n, double p[], double x);
```

c) Eine Funktion, die die Summe zweier Polynome p und q bildet, indem deren Koeffizienten paarweise addiert werden. Schnittstelle:

```
int polySum(int np, double p[], int nq, double q[], double r[]);
```

Diese Funktion liefert im Ausgabeparameter r die Summe von p und q. Das Funktionsergebnis ist der Grad von r, also das Maximum der Grade von p und q, also $\max(np, nq)$, z.B.:

$$p(x) = 1 + x + 3x^2 - 4x^3$$
$$q(x) = 1 + 2x - 5x^2 - 3x^3$$
$$p(x) + q(x) = r(x) = 2 + 3x - 2x^2 - 7x^3$$

d) Eine Funktion, die das Produkt zweier Polynome p und q bildet, indem deren Koeffizienten so miteinander multipliziert werden, dass jeder Koeffizient des zweiten Polynoms mit jedem des ersten multipliziert wird. Sind beide vom Grad n, so erfordert dies $(n+1)^2$ Multiplikationen, führt also zu einem $O(n^2)$-Verfahren (bezogen auf die Anzahl der Multiplikationen):

```
int polyProd(int np, double p[], int nq, double q[], double r[]);
```

Diese Funktion liefert im Ausgabeparameter r das Produkt von p und q. Das Funktionsergebnis ist der Grad von r, also das die Summe der Grade von p und q, also $np + nq$, z.B.:

$$p(x) = 1 + x + 3x^2 - 4x^3$$
$$q(x) = 1 + 2x - 5x^2 - 3x^3$$
$$p(x) \cdot q(x) = r(x) = 1 + 3x - 6x^3 - 26x^4 + 11x^5 + 12x^6$$

e) Die Krönung Ihrer Implementierung ist eine Funktion zur Bildung des Produkts, die durch Anwendung des Divide&Conquer-Prinzips nun aber ein $O(n^{1.58})$-Verfahren implementiert:

```
int polyProdFast(int np, double p[], int nq, double q[], double r[]);
```

Ohne Beschränkung der Allgemeinheit können Sie annehmen (*und dürfen es auch so implementieren!*), dass

- beide Polynome (p und q) vom selben Grad n sind ($n = n_p = n_q$),
- und dass $n+1$ eine Zweierpotenz ist ($n + 1 = 2k$ mit $k > 1$).

Idee des Verfahrens: Beide Polynome werden in der Mitte in ein niederwertiges (*low*, *l*) und ein höherwertiges (*high*, *h*) Polynom zerlegt, also z.B. für $p(x)$

$$p_l(x) = p_0 + p_1 x + \cdots + p_{n/2} x^{n/2}$$
$$p_h(x) = p_{n/2+1} + p_{n/2+2} x + \cdots + p_n x^{n/2}$$

Damit kann das Produkt auch wie folgt berechnet werden:

$$p \cdot q = p_l \cdot q_l + (p_l \cdot q_h + q_l \cdot p_h) \cdot x^{n/2+1} + p_h \cdot q_h \cdot x^{x+1}$$

Das bringt allerdings noch keinen Vorteil, denn es sind nun vier Multiplikationen von Polynomen halber Länge notwendig, und weil $4 \cdot \left(\frac{n+1}{2}\right)^2 = (n+1)^2$ ist, handelt es sich immer noch um ein quadratisches Verfahren.

Eine Verbesserung ergibt sich erst, wenn man ganz genau hinschaut und erkennt, dass man, wenn man drei Hilfspolynome r_l, r_h und r_m definiert, das Produkt, auch folgendermaßen berechnen kann:

$$r_l = p_l \cdot q_l$$
$$r_h = p_h \cdot q_h$$
$$r_m = (p_l + p_h) \cdot (q_l + q_h)$$
$$p \cdot q = r_l + (r_m - r_l - r_h) \cdot x^{n/2+1} + r_h \cdot x^{n+1}$$

Damit sind „nur“ noch drei Polynommultiplikationen (je eine für r_l, r_h und r_m) notwendig. Das „Zusammenbauen“ zum Schluss erfolgt einfach durch Zuweisung der Elemente von r_l, r_h und r_m an die richtigen Indizes von r.

Beispiel

$$p(x) = 1 + x + 3x^2 - 4x^3$$
$$q(x) = 1 + 2x - 5x^2 - 3x^3$$

Diese Polynome lassen sich nun folgendermaßen zerlegen:

$$p_l(x) = 1 + x \qquad q_l(x) = 1 + 2x$$
$$p_h(x) = 3 - 4x \qquad q_h(x) = -5 - 3x$$

Daraus ergibt sich als Zwischenrechnungen:

$$r_l(x) = (1 + x) \cdot (1 + 2x) = 1 + 3x + 2x^2$$
$$r_h(x) = (3 - 4x) \cdot (-5 - 3x) = -15 + 11x + 12x^2$$
$$r_m(x) = (4 - 3x) \cdot (-4 - x) = -16 + 8x + 3x^2$$
$$r_m(x) - r_l(x) - r_h(x) = -2 - 6x - 11x^2$$

Schließlich kommen wir wieder zu demselben Produkt wie schon im Beispiel zu Punkt d):

$$r(x) = 1 + 3x - 6x^3 - 26x^4 + 11x^5 + 12x^6$$

Hinweise:

1. Für die Implementierung dieser Aufgabe ist *viel weniger Mathematik* notwendig als auf den ersten Blick ersichtlich. Sie müssen lediglich mit Feldern und Indizes jonglieren.
2. Sie können diese Übung komplett ohne dynamischen Speicher implementieren, indem Sie die Felder für das Resultat groß genug wählen und bereits am Stack anlegen. Falls Sie dennoch bereits mit dynamisch allokiertem Speicher arbeiten wollen (nicht empfohlen!), überlegen und dokumentieren Sie genau, wer für das Reservieren und Freigeben verantwortlich ist und stellen Sie sicher, dass keine Speicherlecks entstehen.

Allgemeine Hinweise

1. Geben Sie für alle Ihre Lösungen immer eine **Lösungsidee** an.
2. **Kommentieren** und **testen** Sie Ihre Programme **ausführlich**.