Familienname:

Vorname:

Matrikelnummer:

Aufgabe 1 (2 Punkte):
Aufgabe 2 (2 Punkte):
Aufgabe 3 (1 Punkte):
Aufgabe 4 (3 Punkte):
Aufgabe 5 (2 Punkte):
Aufgabe 6 (4 Punkte):
Aufgabe 7 (3 Punkte):
Aufgabe 8 (3 Punkte):
Aufgabe 9 (10 Punkte):

Gesamtpunktzahl:

Schriftlicher Nachtest zu C (90 Minuten)
VU Einführung ins Programmieren für TM (WS 2009/10)

01. März 2010

Aufgabe 1 (2 Punkte). Schreiben Sie einen Struktur-Datentyp `CDouble` zur Speicherung komplexer Zahlen $z \in \mathbb{C}$. In der Struktur sollen Realteil `real` und Imaginärteil `imag` gespeichert werden. Verwenden Sie diesen Datentyp in allen folgenden Aufgaben.

Lösung zu Aufgabe 1.

Folgende Zugriffsfunktionen stehen für den Datentyp `CDouble` zur Verfügung:

- `CDouble* newCDouble(double real, double imag)`
- `CDouble* delCDouble(CDouble* z)`
- `double getCDoubleReal(CDouble* z)`
- `double getCDoubleImag(CDouble* z)`
- `void setCDoubleReal(CDouble* z, double real)`
- `void setCDoubleImag(CDouble* z, double imag)`

Verwenden Sie diese Signaturen bzw. Funktionen in den folgenden Aufgaben.

Aufgabe 2 (2 Punkte). Schreiben Sie eine Funktion `newCDouble`, die eine neue komplexe Zahl allokiert und initialisiert.

Lösung zu Aufgabe 2.

Aufgabe 3 (1 Punkt). Schreiben Sie eine Funktion `getCDoubleImag`, die den Imaginärteil einer komplexen Zahl zurückgibt.

Lösung zu Aufgabe 3.

Aufgabe 4 (3 Punkte). Schreiben Sie eine Funktion `CMultiply`, die für zwei komplexe Zahlen $x, y \in \mathbb{C}$ das Produkt $x \cdot y$ berechnet und das Ergebnis entsprechend zurückgibt.

Lösung zu Aufgabe 4.

Aufgabe 5 (2 Punkte). Schreiben Sie einen Strukturdatentyp `CPol` zur Speicherung von Polynomen $p(x) = \sum_{j=0}^{n} a_j x^j$ mit komplexwertigen Koeffizienten a_j. In der Struktur sollen neben dem Grad $n \in \mathbb{N}$ auch die $(n+1)$ Koeffizienten $a_j \in \mathbb{C}$ gespeichert werden. Verwenden Sie diesen Datentyp in allen folgenden Aufgaben.

Lösung zu Aufgabe 5.

Folgende Zugriffsfunktionen stehen für den Datentyp `CPol` zur Verfügung:

- `CPol* newCPol(int n)`
- `CPol* delCPol(CPol* p)`
- `int getCPolDegree(CPol* p)`
- `CDouble* getCPolCoeff(CPol* p, int i)`
- `void setCPolCoeff(CPol* p, int i, CDouble* c)`

Verwenden Sie diese Signaturen bzw. Funktionen in den folgenden Aufgaben.

Aufgabe 6 (4 Punkte). Schreiben Sie eine Funktion `newCPol`, die für gegebenen Grad $n \in \mathbb{N}$ ein neues Polynom allokiert und initialisiert.

Lösung zu Aufgabe 6.

Aufgabe 7 (3 Punkte). Schreiben Sie eine Funktion `delCPol`, die den Speicher für ein dynamisch allokiertes Polynom freigibt und den `NULL`-Pointer zurückgibt.

Lösung zu Aufgabe 7.

Aufgabe 8 (3 Punkte). Schreiben Sie eine Funktion `setCPolCoeff`, die für einen gegebenen Index i und eine komplexe Zahl $c \in \mathbb{C}$ dem Koeffizienten des Polynoms den Wert $a_i = c$ durch explizite Zuweisung von Real- und Imaginärteil zuweist.

Lösung zu Aufgabe 8.

Hinweis: In der folgenden Aufgabe dürfen Sie zusätzlich zu der von Ihnen programmierten Funktion

- `CDouble* CMultiply(CDouble* a,CDouble* b)`

folgende Funktion verwenden:

- `CDouble* CAdd(CDouble* a,CDouble* b)`

Aufgabe 9 (10 Punkte). Schreiben Sie eine Funktion `prodCPol`, die für zwei Polynome $p(x) = \sum_{j=0}^{m} a_j x^j$ und $q(x) = \sum_{k=0}^{n} b_k x^k$ das Produktpolynom pq berechnet und zurückgibt. — Beachten Sie, dass p und q verschiedenen Grad haben können. Überlegen Sie sich zunächst, welchen Grad das Polynom pq hat und wie die Koeffizienten c_j von pq berechnet werden können. — Passen Sie auf, dass durch die Verwendung von `CAdd` und `CMultiply` kein toter Speicher entsteht!

Lösung zu Aufgabe 9.