Dirk Praetorius, Samuel Ferraz-Leite
Christoph Überhuber

Sommersemester 2010
04. Mai 2010

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 6

Die Aufgaben mit Stern (*) sind bis zur Übung in der kommenden Woche vorzubereiten. Kopieren Sie die Source-Codes vor der Übung auf Ihren Account auf der `lva.student.tuwien.ac.at` *und überprüfen Sie, ob diese mit dem* `gcc` *compiliert werden können. In den folgenden Übungsaufgaben sollen* ***Strukturen*** *und* ***weitere Sprachkonzepte*** *geübt werden.*

Aufgabe 6.1*. Schreiben Sie einen Strukturdatentyp `cdouble`, in dem Realteil und Imaginärteil einer Zahl $a + bi \in \mathbb{C}$ jeweils als `double` gespeichert werden. Schreiben Sie Funktionen `newCDouble`, `delCDouble` sowie die vier Zugriffsfunktionen `setCDoubleReal`, `getCDoubleReal`, `setCDoubleImag` sowie `getCDoubleImag`. Speichern Sie den Source-Code im Verzeichnis `serie06` unter dem Namen `cdouble.c`.

Aufgabe 6.2*. Schreiben Sie Funktionen, die die Addition, die Subtraktion, die Multiplikation und die Division für komplexe Zahlen $a + bi \in \mathbb{C}$ realisieren. Verwenden Sie zur Speicherung die Struktur aus Aufgabe 6.1, und benutzen Sie beim Strukturzugriff nur die entsprechenden Zugriffsfunktionen. Schreiben Sie ein aufrufendes Hauptprogramm, in dem zwei komplexe Zahlen $w, z \in \mathbb{C}$ eingelesen werden und $w + z$, $w - z$, $w \cdot z$ sowie w/z ausgegeben werden. Binden Sie den Code aus Aufgabe 6.1 mittels `#include "cdouble.c"` ein. Speichern Sie den Source-Code im Verzeichnis `serie06` unter dem Namen `carithmetik.c`.

Aufgabe 6.3*. Der C-Preprozessor bietet die Möglichkeit, mit der Definition `#define TEXT1 text2` den Ausdruck `TEXT1` im Quelltext durch den Ausdruck `text2` zu ersetzen. Mithilfe von `#define` lassen sich zusätzlich zu Konstanten auch Makros definieren: Der Audruck `TEXT1` kann durch eine Parameterliste in runden Klammern `TEXT1(A,B,...)` ergänzt werden. Die Parameter können anschließend in `text2` verwendet werden. Man kann beispielsweise das Berechnen des Quadrats einer Zahl als Makro definieren:

```
#define SQUARE(X) X * X
```

Beachten Sie, dass hierbei lediglich Textersatz durchgeführt wird. Beispielsweise wird im Programmcode die Befehlszeile

```
double a = SQUARE(3.7);
```

vom Preprozessor durch die Zeile

```
double a = 3.7 * 3.7;
```

ersetzt. Welche Vor- und Nachteile hat ein Makro verglichen mit einer Funktion? Überlegen Sie, wann die Verwendung des hier definierten Makros zu Schwierigkeiten führt. Schreiben Sie ein Hauptprogramm, in dem Sie das Makro `SQUARE` verwenden und unerwartete Ergebnisse erhalten. (Hinweis: Was passiert, wenn Sie SQUARE(a+b) berechnen wollen, und warum?). Schreiben Sie ein Makro `GOODSQUARE`, dass stets das Quadrat des eingesetzen Ausdrucks korrekt berechnet. Sepeichern Sie den Source-Code mit exemplarischen Tests im Verzeichnis `serie06` unter dem Namen `testmakros.c`.

Aufgabe 6.4*. Die Main-Funktion ermöglicht es, Parameter vom System zu übernehmen:

```
main(int argc, char* argv[])
```

Im Array von Strings `argv` werden Parameter aus der Kommandozeile übergeben. Das erste Array-Element ist stets der Funktionsname selbst. Die folgenden Array-Elemente sind die vom Benutzer bei Funktionsaufruf aus der Shell übergebenen Parameter. Die Variable `argc` gibt an wieviele Parameter übergeben wurden. Ein Minimalbeispiel für eine Funktion, die alle übergebenen Parameter ausgibt lautet wie folgt:

```
/* File: testarg.c */
#include <stdio.h>

main(int argc, char* argv[])
{
  int i=0;

  for(i=1;i<argc;++i)
    printf("%s ",argv[i]);
  printf("\n");
}
```

Implementieren Sie dieses Beispiel und rufen Sie das Programm mit

```
>> testarg parameter1 text2 test3
```

aus der Konsole auf. Schreiben Sie ein Programm `testCDouble`, das die Real und Imaginärteile zweier komplexer Zahlen als Parameter von der Konsole übernimmt und die Ergebnisse der Funktionen aus Aufgabe 6.2 am Schrim ausgibt. Hinweis: Mit der Funktion `double atof(char* c)` aus der Standardbibliothek können Sie den `double`-Wert auslesen, den ein String repräsentiert. Speichern Sie den Source-Code im Verzeichnis `serie06` unter dem Namen `testCDouble.c`.

Aufgabe 6.5. Schreiben Sie eine Struktur `CPoly` zur Speicherung von Polynomen mit komplexwertigen Koeffizienten, die bezüglich der Monombasis dargestellt sind, d.h. $p(x) = \sum_{j=0}^{n} a_j x^j$. Es sind also der Grad $n \in \mathbb{N}_0$ sowie der Koeffizientenvektor $(a_0, \ldots, a_n) \in \mathbb{C}^{n+1}$ zu speichern. Verwenden Sie für die Darstellung der komplexwertigen Koeffizienten den Strukturdatentyp aus Aufgabe 5.1. Schreiben Sie ferner die nötigen Zugriffsfunktionen `newCPoly`, `delCPoly`, `getCPolyDegree`, `getCPolyCoefficient` und `setCPolyCoefficient`.

Aufgabe 6.6. Schreiben Sie eine Funktion `addCpolynomials`, die die Summe $r = p + q$ zweier komplexer Polynome p und q (auch unterschiedlichen Grades) berechnet und zurückgibt. Verwenden Sie zur Speicherung die Struktur aus Aufgabe 6.5. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem zwei Polynome p, q eingelesen und die Summe $r = p + q$ ausgegeben werden.

Aufgabe 6.7. Man schreibe eine Struktur `CVector`, zur Speicherung von Vektoren mit komplexwertigen Koeffizienten. Benutzen Sie hierzu den Strukturdatentyp `cdouble` aus Aufgabe 5.1. Ferner schreibe man die zugehörigen Funktionen `newCVector`, `delCVector`, `getCVectorLength`, `getCVectorEntry`, `setCVectorEntry`.

Aufgabe 6.8. Schreiben Sie eine Funktion `cvectorvector`, die das Skalarprodukt $x \cdot y := \sum_{j=1}^{n} x_j \overline{y_j}$ zweier komplexwertiger Vektoren $x, y \in \mathbb{C}^n$ berechnet. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem die Vektoren x, y eingelesen und der Wert $x \cdot y \in \mathbb{C}$ ausgegeben werden. Speichern Sie den Source-Code unter `cvectorvector.c` in das Verzeichnis `serie06`.

Aufgabe 6.9. Schreiben Sie eine Struktur `CMatrix`, zur Speicherung von $(m \times n)$-Matrizen $A \in \mathbb{C}^{m \times n}$ mit komplexwertigen Koeffizienten. Schreiben Sie ferner die zugehörigen Funktionen `newCMatrix`, `delCMatrix`, `getCMatrixM`, `getCMatrixN`, `getCMatrixCoeff`, `setCMatrixCoeff`.

Aufgabe 6.10. In der Standardbibliothek `time.h` wird die Funktion

```
double clock()
```

bereitgestellt. Um die Laufzeit eines Programms zu messen geht man wie folgt vor:

1. `t=clock();`
2. Programmcode dessen Laufzeit gemessen werden soll
3. `t=clock()-t;`
4. `t=t/CLOCKS_PER_SEC //liefert Laufzeit in Sekunden`

Implementieren Sie die Matrix-Vektor-Multiplikation für Komplexe Matrizen und Vektoren. Bestimmen Sie den arithmetischen Aufwand Ihrer Funktion. Schreiben Sie ein Hauptprogramm, das für zunehmende Dimension (z.B. $N = 50, 100, 200, 400, 800, 1600$) die Laufzeit des Skalarproduktes und der Matrix-Vektor-Mulitplikation misst. Geben Sie die Laufzeiten tabellarisch aus. Was erwarten Sie und was beobachten Sie?