Dirk Praetorius,
Gregor Gantner

Wintersemester 2018/19
13./14.12.2018

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 7

Aufgabe 7.1. Jede Integerzahl $x \in \mathbb{N}_0$ lässt sich in der Form $x = \sum_{k=0}^{N-1} a_k 2^k$ darstellen mit Bits $a_k \in \{0,1\}$, wobei $N \in \mathbb{N}$ die kleinste Zahl ist mit $x < 2^N$. Schreiben Sie eine Funktion `integer2bin`, die eine Integer Zahl x als Input nimmt und als Output einen (dynamischen) String der Bit-Kodierung $a_0 a_1 a_2 a_3 \dots$ zurückgibt. Beispielsweise gilt

```
0 für 0
1 für 1
01 für 2
11 für 3
001 für 4
101 für 5
011 für 6
111 für 7
etc.
```

Schreiben Sie ferner eine Funktion `bin2integer`, welche die Bit-Kodierung einer Integer Zahl $x \in \mathbb{N}_0$ als (dynamischen) String übernimmt und den entsprechenden Integer-Wert $x \in \mathbb{N}_0$ zurückgibt. Testen Sie Ihren Code entsprechend! Speichern Sie den Source-Code unter `integerVSbin.c` in das Verzeichnis `serie07`.

Aufgabe 7.2. Schreiben Sie eine Bibliothek zur Verwaltung von *spaltenweise* gespeicherten $m \times n$-Matrizen. Implementieren Sie die folgenden Funktionen

- `double* mallocmatrix(int m, int n)`
  Allokieren von Speicher für eine spaltenweise gespeicherte $m \times n$- Matrix.
- `double* freematrix(double* matrix)`
  Freigeben des allokierten Speichers einer Matrix.
- `double* reallocmatrix(double* matrix, int m, int n, int mNew, int nNew)`
  Reallokieren des Speichers der Matrix, gegebenenfalls Beibehalten von vorhandenen Einträgen und Initialisieren von neuen Einträgen mit 0.

Speichern Sie die Funktionssignaturen in das Header-File `dynamicmatrix.h`. Schreiben Sie auch entsprechende Kommentare zu den Funktionen in das Header-File. In die Datei `dynamicmatrix.c` kommt dann die Implementierung der Funktionen. Verwenden Sie dynamische Arrays. Wie haben Sie Ihren Code auf Korrektheit getestet?

Aufgabe 7.3. Erweitern Sie die Bibliothek aus Aufgabe 7.2 um folgende Funktionalitäten

- `void printmatrix(double* matrix, int m, int n)`
  Gibt eine spaltenweise gespeicherte $m \times n$-Matrix als Matrix am Bildschirm aus. Die 2×3-Matrix `double matrix[6]={1,2,3,4,5,6}` soll wie folgt ausgegeben werden:

  ```
  1 3 5
  2 4 6
  ```

- `double* scanmatrix(int m, int n)`
  Allokiert Speicher für eine Matrix und liest die Koeffizienten der Matrix von der Tastatur ein.

- `double* cutOffRowJ(double* matrix, int m, int n, int j)`
  Schneidet die j-te Zeile aus einer $m \times n$-Matrix heraus.
- `double* cutOffColK(double* matrix, int m, int n, int k)`
  Schneidet die k-te Spalte aus einer $m \times n$-Matrix heraus.

Verwenden Sie dynamische Arrays. Wie haben Sie Ihren Code auf Korrektheit getestet?

Aufgabe 7.4. Schreiben Sie eine Funktion `matrixvector` zur Berechnung des Matrix-Vektor-Produkts $Ax \in \mathbb{R}^{m\times 1}$, wobei $x \in \mathbb{R}^{n\times 1}$ ist und die Matrix $A \in \mathbb{R}^{m\times n}$ spaltenweise gespeichert ist. Nutzen Sie zum anlegen und freigeben des Speichers ihre Matrix-Bibliothek aus Aufgabe 7.2. Speichern Sie den Source-Code unter `matrixvector.c` in das Verzeichnis `serie07`.

Aufgabe 7.5. Schreiben Sie Funktionen `int countValueInRow(int** matrix, int m, int n, int val, int row)` und `int countValueInColumn(int** matrix, int m, int n, int val, int col)`, die zu einer gegebenen $m \times n$-Matrix die Anzahl der Einträge in der gegebenen Zeile/Spalte zurückliefern, die mit `val` übereinstimmen. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `countValueInRowCol.c` in das Verzeichnis `serie07`.

Aufgabe 7.6. Schreiben Sie ein Programm, welches das bekannte Spiel *Tic Tac Toe* realisiert. Ihr Hauptprogramm könnte z.B. so aussehen:

```
int player=1;
int winner;
int** playboard = newPlayboard();  // Verwende 3x3 Matrix als Spielfeld.
resetPlayboard(playboard);
while(!isGameFinished(playboard)){ // Solange noch kein Sieger klar & Felder frei.
    makeMove(playboard,player);    // Lese Spielzug von der Tastatur ein
                                   // und speichere ihn ins Spielfeld.
    printPlayboard(playboard);     // Gib die neue Spielsituation am Bildschirm aus.
    player=changePlayer(player);   // Wechsle den Spieler.
}
winner=getWinner(playboard);
printWinnerMessage(winner);
delPlayboard(playboard);
```

Die Funktion `makeMove` sollte dabei den Spieler solange nach einem Spielzug fragen, bis dieser gültig ist. (Felder dürfen nicht überschrieben werden!) Für die Funktion `getWinner` können Sie unter anderem Aufgabe 7.5 verwenden. Die Funktion `isGameFinished` kann wiederum die Funktion `getWinner` verwenden. Wie haben Sie ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `tictactoe.c` in das Verzeichnis `serie07`.

Aufgabe 7.7. Schreiben Sie einen Strukturdatentyp `cDouble`, in dem Realteil $a \in \mathbb{R}$ und Imaginärteil $b \in \mathbb{R}$ einer komplexen Zahl $z = a + bi \in \mathbb{C}$ jeweils als `double` gespeichert werden. Die imaginäre Einheit i erfüllt die Eigenschaft $i^2 = -1$, siehe

`https://de.wikipedia.org/wiki/Komplexe_Zahl`.

Schreiben Sie Funktionen

- `cDouble* newCDouble(double a, double b)`,
- `cDouble* delCDouble(cDouble* z)`

sowie die vier Zugriffsfunktionen

- `void setCDoubleReal(cDouble* z, double a)`,
- `double getCDoubleReal(cDouble* z)`,
- `void setCDoubleImag(cDouble* z, double b)`,

- sowie `double getCDoubleImag(cDouble* z)`.

Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code, aufgeteilt in Header-Datei `cdouble.h` und `cdouble.c`, in das Verzeichnis `serie07`.

Aufgabe 7.8. Schreiben Sie Funktionen

- `cDouble* cAdd(cDouble* z, cDouble* w)`,
- `cDouble* cSub(cDouble* z, cDouble* w)`,
- `cDouble* cMult(cDouble* z, cDouble* w)`,
- `cDouble* cDiv(cDouble* z, cDouble* w)`,

die die Addition, die Subtraktion, die Multiplikation und die Division für komplexe Zahlen realisieren. Weiters schreiben Sie

- eine Funktion `double cNorm(cDouble* z)`, die den Betrag $|z| = \sqrt{a^2 + b^2}$ von $z = a + ib \in \mathbb{C}$ berechnet und zurückgibt,
- eine Funktion `cDouble* cConj(cDouble* z)`, die die Konjugierte $\overline{z} = a - ib \in \mathbb{C}$ von $z = a + ib \in \mathbb{C}$ berechnet und zurückgibt.

Verwenden Sie zur Speicherung die Struktur `cDouble` aus Aufgabe 7.7, 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 $|w|$, $|z|$, $w + z$, $w - z$, wz sowie w/z (falls $z \neq 0$) ausgegeben werden. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `carithmetik.c` in das Verzeichnis `serie07`.