Lehrstuhl für Praktische Informatik IV
Prof. Dr. Wolfgang Effelsberg
A5, Raum B223
68131 Mannheim
Telefon: (0621) 181–2600
Email: effelsberg@informatik.uni-mannheim.de

Marcel Busse
A5, Raum B221
68131 Mannheim
Telefon: (0621) 181–2616
Email: busse@informatik.uni-mannheim.de

UNIVERSITÄT MANNHEIM

Programmierkurs in C für Bachelor IMI — Klausur
Herbstsemester 2006 — 25. Januar 2007

Hinweise

1. Überprüfen Sie bitte Ihr Klausurexemplar auf Vollständigkeit (9 Seiten).
2. Bearbeiten Sie die Aufgaben *ausschließlich* auf dem Aufgabenblatt der jeweiligen Aufgabe.
3. Schreiben Sie auf jedes Blatt, das bewertet werden soll, oben Ihren Namen und Ihre Matrikelnummer.
4. Verwenden Sie nur dokumentenechte Stifte (z. B. keinen Bleistift) und keine roten Stifte.
5. Es sind keine Hilfsmittel zugelassen.
6. Bearbeitungszeit: 33 Minuten.

Korrekturzeile — Bitte *nicht* ausfüllen!

Aufgabe	1	2	3	Summe
Max. Punktzahl	10	13	10	33
Erreichte Punktzahl				

Name: Matrikelnummer:

Aufgabe 1 — 10 Punkte

Aufgabe 1 a) — 4 Punkte

Wandeln Sie die folgende rekursive Funktion in eine äquivalente iterative Funktion unter Verwendung einer `while`-Schleife um.

```
int fibonacci(int n) {
  if (n == 0) return 0;
  if (n == 1) return 1;
  return (fibonacci(n−1) + fibonacci(n−2));
}
```

Lösung:

```
int fibonacci(int n) {
  int a = 0, b = 1, tmp, i = 2;
  if (n == 0) return 0;
  while (i <= n) {
    tmp = b;
    b = a + b;
    a = tmp;
    i++;
  }
  return b;
}
```

Aufgabe 1 b) — 2 Punkte

Vergleichen Sie die rekursive Funktion aus der vorherigen Aufgabe mit der iterativen Variante hinsichtlich ihrer Laufzeit.

Lösung:

Die rekursive Version hat eine Laufzeit von $\Theta(2^n)$, die iterative Version benötigt nur $\Theta(n)$.

Name: Matrikelnummer:

Aufgabe 1 c) 4 Punkte

Geben Sie eine rekursive Funktion zur Berechnung der Fibonacci-Folge ähnlich zu der in Aufgabe 1a an, die eine verbesserte Laufzeit aufweist. Die Signatur darf dabei beliebig verändert werden.

Lösung:

```
int fibonacci(int n, int &tmp) {
  if (n == 0) { tmp = 0; return 0; }
  if (n == 1) { tmp = 0; return 1; }
  int f = fibonacci(n−1, tmp);
  int _tmp = tmp;
  tmp = f;
  return (f + _tmp);
}
```

Aufgabe 2 13 Punkte

Prioritätswarteschlangen werden in vielen elementaren Algorithmen verwendet. Eine solche Warteschlange hat die Eigenschaft, dass sie das Einfügen eines neuen Elementes und das Löschen des größten Elementes unterstützt.

Eine einfache Methode zur Organisation einer Prioritätswarteschlange ist eine ungeordnete Liste, wobei alle Elemente in einem `int`-Feld `a[1..n]` aufbewahrt werden ($n \leq$ `n_max`). Dabei beschreibt `n` die Anzahl an derzeit in der Liste vorhandenen Elementen und `n_max` die maximal mögliche Anzahl an Elementen.

Aufgabe 2 a) 1 Punkte

Implementieren Sie die Funktion

```
void insert(int v);
```

zum Einfügen eines neuen Elementes.

Lösung:

```
void insert(int v) {
  if (n < n_max) {
```

```
    a[n++] = v;
  }
}
```

Aufgabe 2 b) 4 Punkte

Implementieren Sie die Funktion

```
int remove();
```

zum Entfernen des größten Elementes in der Prioritätswarteschlange. Dabei soll das entsprechende Element dem Rückgabewert der Funktion entsprechen.

Lösung:

```
int remove() {
  int tmp, i, max_i = 0;
  for (i = 1; i < n; i++) {
    if (a[i] > a[max_i]) max_i = i;
  }
  tmp = a[max_i];
  for (i = max_i+1; i < n; i++) a[i−1] = a[i];
  n−−;
  return tmp;
}
```

Aufgabe 2 c) 8 Punkte

Durch eine geschickte Organisation der Liste als Baum (Heap) kann die Laufzeit der Operationen verbessert werden. Ein Element mit Index i hat dabei zwei Söhne auf den Positionen $2i$ und $2i + 1$. Als zusätzliche Eigenschaft gilt dabei, dass beide Söhne kleinere Elemente speichern. Das größte Element befindet sich somit ständig an der Wurzel des Baumes auf Position 1.

Implementieren Sie jetzt die Funktion zum Einfügen eines neuen Elementes unter Verwendung der Heap-Struktur.

```
void insert(int v);
```

Hinweis: Fügen Sie das neue Element auf der Position `n+1` in das Feld `a` ein. Anschließend vergleichen Sie es mit seinem Vater und, falls nötig, vertauschen beide Elemente. Wiederholen Sie diesen Schritt solange, bis die Heap-Eigenschaft wieder hergestellt wurde.

Name: Matrikelnummer:

Lösung:

```
void insert(int v) {
  int i = n+1;
  if (i < n_max) {
    n++;
    a[i] = v;
    while ((i != 1) && (a[i/2] > a[i])) {
      tmp = a[i/2];
      a[i/2] = a[i];
      a[i] = tmp;
      i /= 2;
    }
  }
}
```

Aufgabe 3 — 10 Punkte

Schreiben Sie ein Programm zur Multiplikation zweier `int`-Matrizen A und B. Gehen Sie davon aus, dass beide Matrizen bereits eingelesen wurden und im Speicher liegen. Matrix A sei dabei durch das `int`-Feld der Größe `[0..l-1, 0..m-1]` und Matrix B durch das Feld der Größe `[0..m-1, 0..n-1]` bestimmt. Das Ergebnis der Multiplikation ist eine Matrix C der Größe `[0..l-1, 0..n-1]`.

$$\begin{pmatrix} a_{0,0} & \cdots & a_{0,m-1} \\ \vdots & \ddots & \vdots \\ a_{l-1,0} & \cdots & a_{l-1,m-1} \end{pmatrix} \begin{pmatrix} b_{0,0} & \cdots & b_{0,n-1} \\ & & \\ \vdots & \ddots & \vdots \\ & & \\ b_{m-1,0} & \cdots & b_{m-1,n-1} \end{pmatrix} = \begin{pmatrix} c_{0,0} & \cdots & c_{0,n-1} \\ \vdots & \ddots & \vdots \\ c_{l-1,0} & \cdots & c_{l-1,n-1} \end{pmatrix}$$

Die Multiplikation der Matrizen A und B kann mittels der Produktsummenformel berechnet werden:

$$c_{i,j} = \sum_{k=0}^{m-1} a_{i,k} \cdot b_{k,j} \qquad i = 0 \ldots l-1, \quad j = 0 \ldots n-1.$$

Aufgabe 3 a) — 3 Punkte

Implementieren Sie zunächst eine Funktion zum dynamischen Erzeugen der Ergebnismatrix C. Die Matrix soll mittels eines Zeigers als Rückgabewert zurückgeliefert werden. Verwenden Sie dafür folgende Funktionsdeklaration

Name: Matrikelnummer:

```
int** create(int l, int n);
```

Lösung:

```
int** create(int l, int n) {
  int i;
  int** C = (int**)malloc(l * sizeof(int*));
  if (C == NULL) return NULL;
  for (i = 0; i < l; i++) {
    C[i] = (int*)malloc (n * sizeof (int*));
    if (C[i] == NULL){
      int j;
      for (j = 0; j < i; j++) free(C[j]);
      free (C);
      return NULL;
    }
  }
  return C;
}
```

Aufgabe 3 b) 5 Punkte

Implementieren Sie jetzt die Funktion zum Multiplizieren der Matrizen A und B.

```
int** mult(int** A, int** B, int l, int m, int n);
```

Beide Matrizen werden mittels call-by-reference übergeben. Als Rückgabewert wird ein Zeiger auf die Ergebnismatrix C erwartet.

Lösung:

```
int** mult(int** A, int** B, int l, int m, int n) {
  int i, j, k;
  int** C;
  if ((A == NULL) || (B == NULL)) return NULL;
  C = create(l, n);
  if (C == NULL) return NULL;
  for (i = 0; i < l; i++) {
    for (j = 0; j < n; j++) {
      C[i][j] = 0;
      for (k = 0; i < m; k++) {
        C[i][j] += A[i][k] * B[k][j];
      }
```

Name: Matrikelnummer:

```
      }
    }
    return C;
  }
```

Aufgabe 3 c) 2 Punkte

Abschließend geben Sie eine Funktion an, die den zuvor dynamisch allokierten Speicher wieder frei gibt.

```
void destroy(int** C, int l, int n);
```

Lösung:

```
void destroy(int** C, int l, int n) {
  int i;
  if (C == NULL) return;
  for (i = 0; i < l; i++) free(C[i]);
  free(C);
}
```