Modellierung und Programmierung 2

Hinweise Abgabe: https://moodle2.uni-leipzig.de/course/view.php?id=3907

Übungsserie 2

Programmierung mit C

Aufgabe 5 Schreiben Sie ein C-Programm `binom_it.c`, das den Binomialkoeffizienten $\binom{a}{b}$ zweier natürlicher Zahlen a und b *iterativ* berechnet. Aufgrund des begrenzten Darstellungsbereichs der verschiedenen Datentypen wird es jedoch selbst für kleine Werte von a und b schnell zu einem Überlauf und damit Rechenfehlern kommen, wenn Sie diese Formel verwenden:

$$\binom{a}{b} = \frac{a!}{b! \cdot (a-b)!}$$

Verbessern Sie die Funktion, so dass Sie einen möglichst großen Wertebereich an Eingabedaten abdecken. Dabei soll die Rechenzeit $O(a)$ nicht überschritten werden.

Vermerken Sie das größtmögliche a, das für das Ihren Algorithmus für alle $0 \leq b \leq a$ noch korrekte Ergebnisse liefert, in einem Kommentar über der entsprechenden Funktionsdeklaration. Dokumentieren Sie des Weiteren Ihre Überlegungen zum Algorithmus selbst mit Kommentaren an geeigneten Stellen im Funktionsrumpf.

Hinweis: $\binom{a}{b} = 1$ falls: $a = 0$ oder $b = 0$ oder $a = b$ ist.

Aufgabe 6 Gegeben ist folgendes Code-Fragment:

```
1  char txt[] = "abcdefghij";
2  char *zeiger;
3  zeiger = &txt[1];
4  printf("%p\n", txt);
5  printf("%p\n", &zeiger[4]);
6  printf("%c\n", *zeiger);
7  printf("%p\n", zeiger+3);
8  printf("%p\n", ++zeiger);
9  printf("%c\n", *++zeiger);
10 printf("%p\n", zeiger);
11 printf("%c\n", *zeiger++);
12 printf("%c\n", --(*zeiger));
13 printf("%c\n", *zeiger++);
14 printf("%c\n", *(zeiger-3));
15 printf("%c\n", *zeiger);
16 printf("%c\n", zeiger[-1]);
17 printf("%ld\n", zeiger-txt);
```

Schreiben Sie für jede `printf`–Anweisung, was diese unter der Voraussetzung, dass das Feld `txt` an der Position `0xABCDEF90` im Speicher steht, ausgeben würde und begründen Sie dies. Hinweis: “Springt an Stelle von ’i’ und gibt ’i’ aus” zählt nicht als Begründung.

Aufgabe 7 Schreiben Sie in einem C-Programm `strings.c` Funktionen

- zum *Aneinanderhängen* zweier Zeichenketten,

- zum *Generieren einer Teil- Zeichenkette* aus einer gegebenen Zeichenkette (inklusive der Start- und Endposition), und
- zum *Umkehren einer Zeichenkette*.

Verwenden Sie keine Zeichenkettenfunktionen der Standardbibliothek oder anderer Bibliotheken, sondern implementieren Sie die Funktionalität unter Verwendung von Zeigerarithmetik selbst. Schreiben Sie ebenfalls ein kleines Hauptprogramm, welches die Funktionen kurz testet.

Aufgabe 8 Leider bietet C keine direkte Möglichkeit, die Grenzen von dynamischen Feldern einzuhalten. Schreiben Sie ein C-Modul `intarray.h` und `intarray.c` in dem Sie eine Struktur `IntArray` definieren, die zusätzlich zum eigentlichen Feld, bestehend aus Integer-Ganzzahlen, ebenfalls Information über die Länge des Feldes speichert sowie folgende Operationen:

- Erzeugen eines solchen `IntArrays` bei einer gegebenen Länge,
- Zerstören eines solchen `IntArrays`,
- Lesen und Setzen von Feldelementen an einer gegebenen Position n

Im Falle von Zugriffen außerhalb des Feldbereiches, soll dies entweder ignoriert werden oder das Programm mit einer entsprechenden Fehlermeldung abgebrochen werden. Dokumentieren Sie das von Ihnen gewählte Verhalten an geeigneten Stellen. Implementieren Sie weiterhin ein kleines Testprogramm namens `arraymain.c`, in welchem Sie die Funktionalität Ihres Moduls testen.

Zusatzaufgabe 2 Schreiben Sie in einem C-Modul `bitstream.h` und `bitstream.c` eine Struktur für einen Bitstream. Dieser besteht aus einem Zeiger auf ein Feld von Bytes (verwenden Sie `unsigned char`), einer Längenangabe des Feldes, sowie einer Positionsangabe, an welchem Bit man sich gerade im Feld befindet.

Implementieren Sie Funktionen zum Erzeugen und Löschen eines solchen Bitstreams. Nach der Generierung eines Bitfeldes soll das Feld anfangs nur mit Nullen gefüllt sein und der Positionszeiger auf den Anfang des Bitfeldes zeigen. Schreiben Sie weiterhin eine Funktion, die in einen gegebenen Stream eine gegebene Zahl von Bits schreibt, die als vorzeichenlose Ganzzahl codiert sind (`unsigned int`). Dabei soll gelten, dass das Bit niedrigsten Position zuerst geschrieben wird.

Schreiben Sie in einem Hauptprogramm `bitmain.c` einen Bitstream Ihrer Wahl mit Anweisungen ähnlich dem gleich folgendem Beispiel und geben Sie anschließend das Feld von Zeichen als Folge von Ganzzahlen aus. Um hierbei die Modularisierung nicht zu zerstören, ist es notwendig, eine entsprechende Funktion dem Modul `bitstream` hinzuzufügen.

Hinweis: Bitverschiebungen können Sie sowohl über arithmetische Ausdrücke, d.h. mittels Multiplikation, Division und Rest bezüglich Zweierpotenzen, oder aber durch direkte

Bitverschiebeoperationen >> und << umsetzen.

Beispiel: Angenommen, Sie hätten die folgende Folge von Schreiboperationen auf einem anfangs leeren Bitstream ausgeführt:

```
/* Bitstream mit 10*8 Bits erzeugen */
struct Bitstream *bs = bs_create( 10 );

/* schreibe ein Bit mit dem Wert 0 */
bs_write( bs, 1, 0 );

/* schreibe ein Bit mit dem Wert 1 */
bs_write( bs, 1, 1 );

/* schreibe fuenf Bits, die den Wert 27 ergeben */
bs_write( bs, 5, 27 );

/* schreibe neun Bits, die den Wert 510 ergeben */
bs_write( bs, 9, 510 );

/* schreibe drei Bits, die den Wert 3 ergeben */
bs_write( bs, 3, 3 );
```

Dann wäre der Inhalt des Bitstreams:

Byte 0								Byte 1								Byte 2			
0			...				7	0			...				7	0			...
0	1	1	1	0	1	1	0	1	1	1	1	1	1	1	1	1	1	0	...
0	1	27						510								3			...