Stefan Kurtz
Andrew Torda

Universität Hamburg
Zentrum für Bioinformatik

Fragen zur Vorbereitung auf die Prüfung im Modul Programmierung für Naturwissenschaften 2 11. Juli 2022

Die folgenden Fragen sollen Ihnen bei der Vorbereitung auf die Abschlussklausur im Modul PFN2 helfen.

1 Aufgaben zur C-Programmierung

1.1. Nennen und beschreiben Sie kurz die einzelnen Schritte der Kompilierung eines C-Programms zu einem ausführbaren Programm, so wie in der Vorlesung dargestellt.

1.2. Identifizieren und korrigieren Sie den syntaktischen Fehler in der folgenden C-Anweisung:

```
for (idx = 1, idx < 5, ++idx) {
  produkt *= idx;
}
```

1.3. Identifizieren und korrigieren Sie den semantischen Fehler in den folgenden C-Anweisungen:

```
if (a < b)
  printf("a␣ist␣kleiner␣als␣b");
else;
  printf("a␣ist␣nicht␣kleiner␣als␣b");
```

1.4. Seien die folgenden syntaktisch korrekten Deklarationen von Variablen gegeben und nehmen Sie an, dass alle Variablen in den darauf folgenden Anweisungen initialisiert werden.

```
void *vptr, *wptr;
char *s;
size_t vsize, ssize;
int cmp;
```

Kreuzen Sie bitte im Folgenden die drei Anweisungen an, die semantisch nicht korrekt sind. Für jedes richtige Kreuz gibt es einen Punkt. Für jedes falsche Kreuz gibt es einen Minuspunkt. Die minimale Punktezahl ist 0.

(a) ☐ `vptr++;`

(b) ☐ `cmp = vptr == wptr;`

(c) ☐ `vsize = sizeof(*vptr);`

(d) ☐ `s = wptr;`

(e) ☐ `s = vptr - 1;`

(f) ☐ `vptr = malloc(ssize * sizeof *s);`

1.5. In C gibt es keinen eigenen Typ für boole'sche Werte. Daher könnte man z.B. den Typ `bool` als Synonym für `int` deklarieren. Geben Sie ein entsprechendes Typsynonym für `bool` an und definieren Sie außerdem die passenden Konstanten `false` und `true`.

1.6. Erläutern Sie die Gemeinsamkeiten und Unterschiede der C-Funktionen `sscanf` und `fscanf`.

1.7. Erläutern Sie den Unterschied zwischen den C-Funktionen `fprintf` und `printf`.

1.8. Wenn man in einem C-Programm Meldungen zu Fehlern ausgeben möchte, dann ist es sinnvoll, den Namen der C-Datei sowie die entsprechende Zeilennummer, die die Fehlermeldung erzeugt, anzugeben. Dazu gibt es zwei Präprozessorvariablen. Wie heißen diese Variablen?

1.9. Beschreiben Sie kurz, was die folgenden Zeilen in einem Makefile bedeuten.

```
CC?=clang
CFLAGS=-g -Wall -Werror -O3

%.o: %.c
        ${CC} $< -c ${CFLAGS} -o $@
```

Nehmen wir an, man führt im aktuellen Verzeichnis den Befehl `ls -l` aus, und es erscheint die folgende Ausgabe:

```
-rw-r-----  1 joe  staff   740 Jul 22 08:47 program.c
-rw-r-----  1 joe  staff  5140 Jul 22 08:52 program.o
-rw-r-----  1 joe  staff  9124 Jul 22 07:27 Makefile
```

Wir nehmen an, dass die Datei `Makefile` den oben gezeigten Inhalt hat. Was passiert, wenn man im Terminal den Befehl `make program.o` ausführt? Begründen Sie Ihre Antwort.

1.10. Schreiben Sie ein Ziel (Goal) in Makefile-Syntax auf, das das folgende leistet: Es soll eine Datei `a.o` aus einer Datei `a.p` erzeugt werden. Dazu soll ein Programm `run.sh` aufgerufen werden, das als einziges Argument `a.p` erhält und sein Ergebnis auf die Standardausgabe schreibt. Sie können davon ausgehen, dass sich das Programm `run.sh` in Ihrer Pfadliste befindet. Kennzeichnen Sie die Stelle im Ziel, an der ein Tabulator eingefügt werden muss.

1.11. Schreiben Sie ein Ziel (Goal) in Makefile-Syntax auf, das das folgende leistet: Es soll eine Datei mit dem Suffix `.o` aus einer Datei mit dem Suffix `.p` erzeugt werden. Dazu

soll ein Programm `compile.sh` aufgerufen werden, das als einziges Argument eine Datei mit dem Suffix `.p` erhält und sein Ergebnis in einer entsprechenden Datei mit dem Suffix `.o` schreibt. D.h. `compile.sh program.p` erzeugt die Datei `program.o`. Sie können davon ausgehen, dass sich das Programm in Ihrer Pfadliste befindet. Kennzeichnen Sie die Stelle im Ziel, an der ein Tabulator eingefügt werden muss.

1.12. Wir betrachten die folgenden Deklarationen und Zuweisungen in einem C-Programm:

```
int a, b, *ptr;
a = 1; b = 16; ptr = (&a) + 1;
```

Nehmen wir an, dass die Variable `a` an der fiktiven Byte-Adresse 16 gespeichert wird. Fertigen Sie eine Skizze an mit einem Kästchen pro Byte, die zeigt, wo die drei Variablen im Speicher abgelegt werden und wieviele Bytes sie jeweils benötigen. Setzen Sie dabei voraus, dass das Programm als 64-Bit Objekt kompiliert wurde. Ignorieren Sie Padding von Bytes auf Wortgrenzen. Geben Sie die Werte der drei Variablen an, wobei Sie eine Adresse als solche kennzeichnen sollen.

1.13. In einem Formatstring für `printf` muss man für die verschiedenen grundlegenden Datentypen unterschiedliche Formatkodierungen angeben. Diese bestehen aus dem Zeichen %, gefolgt von mindestens einem anderen Zeichen. Bei ganzzahligen Werten ≤ 255 hängt die Formatkodierung auch noch davon ab, ob man den Dezimalwert oder das Zeichen für den gegebenen ASCII-Code ausgeben möchte. Ergänzen Sie in der folgenden Tabelle für die angegebenen grundlegenden Typen (Spalte 2) und die gewünschte Form der Ausgabe (Spalte 3) die entsprechende Formatkodierung, die nach dem Zeichen % folgt (Spalte 1).

Formatkodierung	Typ des Wertes	Ausgabe
	`int` ≤ 255	character
	`int`	Dezimalzahl
	`char`	Dezimalzahl
	`short int`	Dezimalzahl
	`unsigned int`	Dezimalzahl ohne Vorzeichen
	`char *`	Folge von Zeichen (\0-terminiert)
	`double`	Fließpunktnotation
	`double`	Exponentennotation mit Zeichen `e`

1.14. Welchen Rückgabetyp haben Funktionen, die keine `return`-Anweisungen enthalten oder bei denen in keiner `return`-Anweisung ein Ausdruck angegeben wird.

1.15. Wenn man in C einen Datentyp implementiert, dann ist es sinnvoll, die Schnittstelle von der Implementierung zu trennen. Der Benutzer oder die Benutzerin des Datentyps sollte nur die Schnittstelle kennen, die in der Header-Datei spezifiziert ist, aber nichts über die Implementierung selbst wissen müssen. Man spricht in diesen Zusammenhang von *Data hiding*.

Nehmen wir an, es soll eine einfach verkettete Liste implementiert werden, und zwar in den Dateien `list.c` (für die Implementierung) und `list.h` (für die Schnittstelle).

Dazu soll ein Typname `List` eingeführt werden, der ein Synonym für eine Struktur ist. Diese Struktur soll jeweils einen Zeiger auf das nächste Listenelement und einen Wert `data` vom Typ `int` enthalten. Geben Sie entsprechende C-Deklarationen des Typsynonyms `List` und der genannten Struktur an. Geben Sie, unter Berücksichtigung des *Data Hiding*-Prinzips, an, in welchen Dateien diese Deklarationen erfolgen und welche `include`-Anweisung in der C-Datei vorkommen müssen.

1.16. Schreiben Sie eine C-Funktion `size_t quersumme(size_t n)`, die die Quersumme (d.h. die Summe der Ziffern) einer ganzen Zahl `n` vom Typ `size_t` liefert. Ihre Lösung darf nicht den Wert von `n` zunächst in einen String verwandeln, um auf die Ziffern zugreifen zu können. Stattdessen müssen Sie durch geeignete arithmetische Operationen nacheinander die Ziffern extrahieren.

1.17. Geben Sie die `include`-Anweisung für den C-Präprozessor an, die in einem C-Programm vorhanden sein muss, damit mathematische Funktionen wie `sqrt` oder `pow` verwendet werden können? Mit welcher Option muss in einem solchen Fall der Linker aufgerufen werden?

1.18. Definieren Sie ein C-Präprozessor Makro `MAX(X,Y)`, das das Maximum von zwei Ausdrücken `X` und `Y` liefert. D.h. `MAX(X,Y)` ist selbst ein Ausdruck.

1.19. Nehmen wir an, dass ein C-Programm Betriebssystem-abhängige Teile enthält, deren Übersetzung durch eine Präprozessorvariable `OS` gesteuert wird, wie im Beispiel unten. Wenn man nun diesen Programmcode auf unterschiedlichen Platformen compiliert, dann möchte man natürlich nicht den Wert der Präprozessorvariable `OS` in der Datei festlegen, in dem sich die Zeilen unten befinden. Wie kann man den Wert von `OS` beim Aufruf des Compilers definieren, so dass `OS` z.B. im Makefile definiert wird. Wie kann man `OS` sogar automatisch durch den Aufruf eines entsprechenden Shell-Befehls definieren.

```
#if OS == Linux
  /* Linux-spezifischer Programmcode */
  ...
#else
  #if OS == Darwin
    /* Mac OSX spezifischer Programmcode */
    ...
  #else
    /* Programmcode fuer alle anderen OS */
    ...
  #endif
#endif
```

1.20. Schreiben Sie ein C-Programm, das über das `argv`-Array Ausdrücke der Form `i + j` für zwei ganze Zahlen `i` und `j` erhält und den Wert des Ausdrucks ausgibt. Falls der übergebene Ausdruck nicht die erwartete Form hat, soll eine Fehlermeldung ausgegeben werden. Beispiel: sei `mini_add.x` der Name des ausführbaren Programms. Dann soll der Aufruf von `./mini_add.x '18 + 24'` die folgende Zeile ausgeben:

```
18 + 24 = 42
```

1.21. Nennen Sie mindestens zwei verschiedene Formen von zusammengesetzten Datentypen in C. Geben Sie jeweils ein Beispiel mit einer Variablen-Deklaration für den zusammengesetzten Datentypen.

1.22. Schreiben Sie ein Programm, dass den Benutzer oder die Benutzerin auffordert, zwei positive ganze Zahlen einzugeben, diese einliest und dann Summe, Produkt, positive Differenz und Quotient ≥ 1 (als Fließkommawert) der beiden Zahlen ausgibt. Beachten Sie, dass für die Ausgabe der positiven Differenz die kleinere von der größeren Zahl subtrahiert wird. Bei der Ausgabe des Quotienten ≥ 1 muss die größere durch die kleinere Zahl geteilt werden. Verwenden Sie korrekte C- oder C++-Syntax.

1.23. Gegeben sei die Zuweisung $y = ax^3 + 7$. Welche der folgenden Ausdrücke sind korrekte C-Anweisungen für diese Zuweisung? Es sind mehrere Antworten möglich.

(a) ☐ `y = a * x * x * x + 7;`

(b) ☐ `y = a * x * x * (x + 7);`

(c) ☐ `y = (a * x) * x * (x + 7);`

(d) ☐ `y = (a * x) * x * x + 7;`

(e) ☐ `y = a * (x * x * x) + 7;`

(f) ☐ `y = a * x * (x * x + 7);`

1.24. Wir betrachten die folgende Deklaration eines initialisierten Arrays in C mit allen Schlüsselworten der Programmiersprache C.

```
char *ckeywords[]
 = {"auto", "double", "int", "struct", "break", "else", "long",
    "switch", "case", "enum", "register", "typedef", "char",
    "extern", "return", "union", "const", "float", "short",
    "unsigned", "continue", "for", "signed", "void", "default",
    "goto", "sizeof", "volatile", "do", "if", "static",
    "while"};
```

Schreiben Sie C-Programmcode, der die Anzahl der Schlüsselworte sowie die Summe der Längen aller Schlüsselworte berechnet und ausgibt. Sie können nicht davon ausgehen, dass das Schlüsselwort `while` das letzte Element des Arrays ist. Sie müssen daher die Anzahl der Elemente in `ckeywords` mit Hilfe des `sizeof`-Operators bestimmen.

1.25. Implementieren Sie eine C-Funktion `int alpha_num(int cc)` die für alle alphabetischen Zeichen, also Groß- und Kleinbuchstaben, die Nummer dieses Buchstabens im Wertebereich von 1 bis 26 entsprechend der Ordnung im Alphabet als Return-Wert liefert. So soll z.B. `alpha_num('a')` den Wert 1 liefern, `alpha_num('e')` den Wert 5 und `alpha_num('z')` den Wert 26. Entsprechendes gilt für die Großbuchstaben. Benutzen Sie in Ihrer Implementierung keine `case`-Anweisung und geben Sie den Funktionsrumpf vollständig an. Sie dürfen Funktionen, deren Deklaration in `ctype.h` zu finden ist, verwenden.

1.26. Gegeben sei die folgende Struktur, die es erlaubt, einen Zeitpunkt minutengenau zu speichern.

```
typedef struct
{
  unsigned char day;
  unsigned char month;
  unsigned int year;
  unsigned char hours;
  unsigned char minutes;
} Time;
```

Die Struktur benötigt auf einem 32-bit System 12 Bytes. Wenn man viele solche Strukturen speichern möchte, dann lohnt es sich, die Größe zu optimieren. Modifizieren Sie die Struktur so, dass die gleichen Werte gespeichert werden können und nur 8 Bytes zur Speicherung benötigt werden. Begründen Sie Ihre Modifikation.

1.27. Erklären Sie den Unterschied zwischen einer `union` und einem `struct` in der Programmiersprache C.

1.28. Implementieren Sie eine C-Funktion

```
void reverse_array(int *arr, size_t len)
```

die die Reihenfolge der Elemente eines Arrays über dem Basistyp `int` umkehrt. Das Array wird über den Zeiger `arr` referenziert und hat genau `len` Elemente. Die Anzahl der Schritte der Funktion muss proportional zur Länge des Arrays sein. Es darf keine Kopie des Arrays angelegt werden.

1.29. Implementieren Sie eine C-Funktion

```
void rev_compl_DNA(char *seq, size_t len)
```

die das reverse Komplement einer DNA-Sequenz im gleichen Speicherbereich liefert, in der auch die Eingabe steht. Dazu muss man die Zeichen in der Sequenz umkehren und die folgende zeichenweise Ersetzung vornehmen: `a` $\mapsto$ `t`, `t` $\mapsto$ `a`, `c` $\mapsto$ `g`, `g` $\mapsto$ `c`. Verwenden Sie für Letzteres eine `case`-Anweisung. Die Sequenz wird durch den Zeiger `seq` referenziert und enthält genau `len` Zeichen aus dem Alphabet {`a`, `c`, `g`, `t`}. Die Anzahl der Schritte der Funktion muss proportional zur Länge des Arrays sein. Es darf keine Kopie des Arrays angelegt werden. Achten Sie insbesondere auch darauf, dass Sequenzen von ungerader Länge korrekt prozessiert werden.

1.30. Implementieren Sie eine C-Funktion

```
void reverse_lines(FILE *fpin)
```

die genau ein Argument hat, nämlich einen Datei-Zeiger (File-pointer) `fpin` zum Lesen einer Datei. Unter Verwendung von `fpin` (der auf eine bereits geöffnete Datei zeigt) soll diese Datei zeilenweise mit der Funktion `fgets` (siehe unten) gelesen werden. Jede eingelesene Zeile soll bis zum letzten Zeichen vor \n in umgekehrter Reihenfolge nach `stdout` ausgegeben werden. Die Zeilenlänge ist auf 1024 Zeichen begrenzt. Zur Umkehrung eines Strings `str` der Länge `len` verwenden Sie die Funk-

tion `void reverse_string(char *str,size_t len)`, die Sie nicht implementieren müssen.

Beispiel: Falls der Dateiinhalt links von `reverse_lines` eingelesen wird, erscheinen in der Ausgabe die Zeilen rechts:

`Alf, John,`	`,nhoJ ,flA`
`Bea,`	`,aeB`
`Mike,`	`,ekiM`
`Nadja`	`ajdaN`

Hier zur Erinnerung der Funktionskopf von `fgets`:

```
char *fgets(char *str, int size, FILE *stream);
```

`fgets` liest höchstens `size-1` Zeichen von `stream` und speichert diese jeweils im Speicherbereich auf den `str` zeigt. Der Lesevorgang eines Aufrufs von `fgets` endet mit dem ersten Vorkommen des Zeichens \n oder dem Erkennen des Dateiendes (end-of-file). Das Zeichen \n bleibt in `str` erhalten. Falls mindestens ein Zeichen gelesen wurde, wird ein \0 angehängt. Falls das Dateiende-Zeichen gelesen wurde, liefert die Funktion `NULL` zurück.

1.31. Mit welcher C-Funktion muss man einen Datei-Zeiger (Filepointer) schließen, den man mit der C-Funktion `fopen` geöffnet hat?

1.32. Schreiben Sie eine C-Funktion

```
long sum_of_lines(const char *progname,const char *filename)
```

die eine Textdatei öffnet, diese zeilenweise liest, dabei die ganzen Zahlen extrahiert und diese aufsummiert. Sie können davon ausgehen, dass in jeder Zeile genau 3 ganze Zahlen stehen, die durch beliebig viele Leerzeichen getrennt sind und in einer Variable vom Typ `long` gespeichert werden können. Sie brauchen also beim Lesen keine Fehlerbehandlung durchzuführen. Der Name der Datei wird über den Zeiger `filename` übergeben. Der Name des Programms, das diese Funktion aufruft, wird durch den Zeiger `progname` übergeben. Falls die Datei nicht geöffnet werden kann, soll eine Fehlermeldung mit dem Formatstring

```
"%s:␣cannot␣open␣file␣%s\n"
```

ausgegeben werden und das Programm soll mit dem Fehlercode `EXIT_FAILURE` terminieren. Falls die Datei geöffnet werden kann, soll mit einer `return`-Anweisung die Summe aller Zahlen in der eingelesenen Datei zurückgegeben werden.

Beispiel: Wir nehmen an, dass eine Datei mit dem Namen `tmp.csv` den folgenden Inhalt hat:

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

Dann soll der Aufruf der Funktion für einen Zeiger `filename`, der auf den String `"tmp.csv"` zeigt, den Wert 5 zurückliefern.

1.33. Um einen C-String der Länge n zu speichern, muß man mindestens $n + 1$ Bytes allokieren. Wozu dient das zusätzliche Byte?

1.34. Schreiben Sie eine C-Funktion

```
size_t *chardist_file(const char *filename)
```

die als Parameter den Namen einer Datei erhält. Die Funktion soll die Verteilung der Zeichen in der Datei bestimmen. Dazu soll diese geöffnet und zeichenweise gelesen werden. Als Ergebnis soll die Verteilung als Array über dem Basistyp `size_t` zurückgeliefert werden. Beachten Sie, dass die Größe des gelieferten Arrays sich aus dem maximalen Wert ergibt, der in einer Variablen vom Typ `unsigned char` gespeichert werden kann. Vergessen Sie nicht, innerhalb der Funktion die Verteilung zu initialisieren. Beachten Sie, dass beliebige Dateien verarbeitet werden sollen, nicht nur DNA-Sequenzen.

1.35. Die C-Standardbibliothek enthält zwei Funktionen `strcpy` und `strdup`, die wie folgt deklariert sind.

```
char *strcpy(char *dest,const char *src);
char *strdup(char *src);
```

Erläutern Sie kurz, was diese beiden Funktionen machen und worin der wesentliche Unterschied besteht.

1.36. Begründen Sie, warum die folgenden syntaktisch korrekten Anweisungen aus einem C-Programm zu einem Programmabbruch führen.

```
char *s = "this␣is␣a␣string";
s[0] = 'T';
```

1.37. Gegeben seien die folgenden Funktionsköpfe in C:

```
double func1();
double *func2();
double (*func3)();
double *(*func4)();
```

Ergänzen Sie bitte am Anfang der folgenden Sätze jeweils einen der Funktionsnamen `func1`, `func2`, `func3` und `func4`, so dass wahre Aussagen entstehen. Jeder der vier Funktionsnamen muss genau einmal verwendet werden.

_____ ist ein Zeiger auf eine Funktion, die einen `double`-Wert liefert.

_____ ist eine Funktion, die einen Zeiger auf einen `double`-Wert liefert.

_____ ist ein Zeiger auf eine Funktion, die einen Zeiger auf einen `double`-Wert liefert.

_____ ist eine Funktion, die einen `double`-Wert liefert.

1.38. Gegeben sei die folgende Funktion `strlen_idx` zur Berechnung der Länge eines \0-terminierten C-Strings.

```
size_t strlen_idx(const char *s)
{
  size_t idx;
```

```
  for (idx = 0; s[idx] != '\0'; idx++)
     /* Nothing */ ;
  return idx;
}
```

Implementieren Sie eine Funktion

```
size_t strlen_ptr(const char *s)
```

zur Berechnung der Länge eines \0-terminierten C-Strings, die keinen indexbasierten Zugriff auf `s` sondern nur eine einzige lokale Variable vom Typ `const char *` verwendet. Eine Funktion aus der C-Standardbibliothek dürfen Sie nicht benutzen.

1.39. Gegeben sei die folgende Deklaration einer Struktur zur Speicherung von drei `size_t`-Werten.

```
typedef struct
{
  size_t digit_count, lower_count, upper_count;
} Charclass_counts;
```

Schreiben Sie eine C-Funktion

```
Charclass_counts *char_classify(const char *s)
```

die im \0-terminierten String `s` die Anzahl der Ziffern, der Kleinbuchstaben und der Grossbuchstaben zählt. Die Funktion soll diese Werte in einer Struktur vom Typ `Charclass_counts` speichern, die über einen Zeiger als `return`-Wert der obigen Funktion geliefert wird. Für die Klassifikation der Zeichen in den drei Kategorien dürfen nur die entsprechenden Funktionen `isdigit`, `islower` und `isupper`, deklariert in der C-Header Datei `ctype.h`, verwendet werden.

1.40. Gegeben sei Integer-Array a der Länge n. Für alle $k, 0 \leq k \leq n-1$ ist $p(k) = \sum_{i=0}^{k} a[i]$ die k-te Partialsumme. Schreiben Sie eine C-Funktion

```
static void partialsums(int *a,size_t n)
```

die im Array a die Partialsummen von a berechnet, d.h. die Werte in a werden durch ihre Partialsummen überschrieben.

1.41. Die Fibobacci-Folge besteht aus den Zahlen $\mathit{fib}(0), \mathit{fib}(1), \mathit{fib}(2), \ldots$ mit

$$\begin{aligned} \mathit{fib}(0) &= 0 \\ \mathit{fib}(1) &= 1 \\ \mathit{fib}(n) &= \mathit{fib}(n-2) + \mathit{fib}(n-1) \text{ für alle } n \geq 2 \end{aligned}$$

Implementieren Sie eine C-Funktion

```
size_t *fibonacci_sequence(size_t max_index)
```

die ein Array `fib` der Länge `max_index+1` liefert, so dass `fib`$[i] = \mathit{fib}(i)$ für alle i, $0 \leq i \leq$ `max_index`. Die Laufzeit soll proportional zu `max_index+1` sein.

1.42. Ein *Run* in einer Sequenz ist eine Folge von gleichen aufeinanderfolgenden Zeichen. Implementieren Sie eine C-Funktion

```
size_t longest_run(const char *seq,size_t len)
```

die für eine Sequenz der Länge `len`, referenziert durch den Zeiger `seq`, die Länge des längsten Runs in `seq` liefert. Die Sequenz darf nur einmal durchlaufen werden, so dass die Anzahl der Schritte der Funktion proportional zu Länge der Sequenz ist. In der folgenden Tabelle finden Sie einige Beispiele für den zu berechnenden Wert (siehe Spalte 3) für verschiedene Sequenzen:

Sequenz	längster Run r	$\|r\|$
ε (leere Sequenz)	ε	0
`agacgatagatatcg`	`a`	1
`aag`	`aa`	2
`agagaaa`	`aaa`	3
`aggggagaaa`	`gggg`	4
`ggagaac`	`gg`	2

1.43. Implementieren Sie, basierend auf der Funktion `qsort` aus der C-Standardbibliothek, eine C-Funktion

```
void sort_int_ulong_pairs(IntUlongPair *tab, size_t len)
```

zum Sortieren eines Arrays über dem Basistyp `IntUlongPair`, der wie folgt deklariert ist:

```
typedef struct
{
  int first;
  unsigned long second;
} IntUlongPair;
```

Dabei ist `len` die Länge des Arrays, das durch `tab` referenziert wird. Dabei sollen `IntUlongPair`-Strukturen aufsteigend nach der ersten Komponente `first` sortiert werden. Falls die erste Komponente von zwei zu vergleichenden `IntUlongPair`-Strukturen gleich ist, soll absteigend nach der zweiten Komponente `second` sortiert werden.

Zur Erinnerung geben wir hier nochmal den Typ der Funktion `qsort` an:

```
void qsort(void *base, size_t num_elems, size_t size_elem,
           int (*compare)(const void *, const void *));
```

1.44. Gegeben sei ein aufsteigend sortiertes Array von ganzen Zahlen vom Typ `size_t`. Seien `leftptr` und `rightptr` Zeiger auf das erste bzw. letzte Element des sortierten Arrays.

Schreiben Sie eine C-Funktion

die mit Hilfe einer binären Suche einen Zeiger auf das kleinste Element, das $\geq$ dem Schlüssel `key` ist, liefert. Falls es von diesem Element mehrere Vorkommen im Array gibt, soll die Funktion einen Zeiger auf das Vorkommen des Elements mit dem minimalen Index im Array liefern. Die Anzahl der Vergleiche des Schlüssels mit einem Wert

im Array soll höchstens $1 + \lfloor \log_2(n) \rfloor$ sein, wobei n die Anzahl der Elemente des Arrays ist. Falls es im Array kein Element gibt, das $\geq$ dem Schlüssel ist, dann liefert die Funktion den `NULL`-Zeiger.

Als Ausgangspunkt Ihrer Entwicklung können Sie die folgende C-Funktion entsprechend erweitern. Diese Funktion liefert die Adresse eines Elements, das identisch ist mit dem Schlüssel, bzw. `NULL`, wenn das Element nicht vorhanden ist:

```
const size_t *binarysearch(const size_t *leftptr,
                           const size_t *rightptr,
                           size_t key) {
  while (leftptr <= rightptr) {
    const size_t *midptr
      = leftptr + (size_t) (rightptr-leftptr)/2;
    if (key < *midptr)
      rightptr = midptr - 1;
    else
      if (key > *midptr)
        leftptr = midptr + 1;
      else
        return midptr;
  }
  return NULL;
}
```

Beispiel: Sei das folgende sortierte Array gegeben:

19 27 33 39 55 76 78 79 84 91

Die folgende Tabelle zeigt für verschiedene Schlüssel die gelieferten Adressen (als Index relativ zum Start des Arrays) und die Werte an den Adressen (falls diese nicht `NULL` sind):

`key`	Adresse	Wert
18	0	19
20	1	27
26	1	27
27	1	27
32	2	33
33	2	33
34	3	39
38	3	39
92	`NULL`	

1.45. Gegeben sei ein aufsteigend sortiertes Array von ganzen Zahlen vom Typ `size_t`. Das Array enthält keine Duplikate. Seien `leftptr` und `rightptr` Zeiger auf das erste bzw. letzte Element des sortierten Arrays. Schreiben Sie eine C-Funktion

```
const size_t *linearsearch_sm_geq(const size_t *leftptr,
                                  const size_t *rightptr,
                                  size_t key)
```

Diese Funktion liefert im gegebenen Array einen Zeiger auf das kleinste Element, das $\geq$ dem Schlüssel `key` ist. Falls es im Array kein Element gibt, das $\geq$ dem Schlüssel ist, dann liefert die Funktion den `NULL`-Zeiger. Verwenden Sie eine lineare Suche, die im schlechtesten Fall für ein Array mit n Elementen höchstens n Vergleiche des Schlüssels mit dem Array benötigt.

1.46. Implementieren Sie eine nicht-rekursive C-Funktion

```
void insert_sorted_array(int *arr,size_t len,int elem)
```

die einen Integer-Wert `elem` in ein aufsteigend sortiertes Array mit `len` Elementen, referenziert durch den Zeiger `arr` sortiert einfügt. Das Array darf nur einmal durchlaufen werden, so dass die Anzahl der Schritte proportional zur Länge des Arrays ist. Nehmen Sie an, dass der Speicherbereich, auf den `arr` verweist gross genug ist, um ein weiteres Element aufzunehmen. Nach dem Einfügen von `elem` soll das entstehende Array mit `len+1` Elementen aufsteigend sortiert sein.

1.47. Sei die folgende Deklaration einer Struktur zum Speichern eines Datums gegeben:

```
typedef struct
{
  int day, month, year;
} Date;
```

Welche der beiden folgenden Implementierungen der C-Funktion `date_new` zur Erzeugung einer neuen Struktur vom Typ `Date` (referenziert durch einen Zeiger) ist korrekt? Welche Implementierung ist falsch? Begründen Sie Ihre Antwort.

```
Date *date_new(int day,int month,int year)
{
  Date date;
  date.day = day;
  date.month = month;
  date.year = year;
  return &date;
}
```

```
Date *date_new(int day,int month,int year)
{
  Date *date = malloc(sizeof(*date));
  assert(date != NULL);
  date->day = day;
  date->month = month;
  date->year = year;
  return date;
}
```

1.48. Schreiben Sie ein C-Programm `reverse_each_arg.c`, das beliebig viele Argumente über das Array `argv` erhält. Es sollen nacheinander alle Argumente in den gleichen dynamisch allokierten Speicherbereich kopiert und dann jeweils umgekehrt ausgegeben werden, ohne eine weitere Kopie zu erstellen. Das Programm soll nur eine Iteration über

`argv` durchführen. Falls das ausführbare Programm den Namen `reverse_each_arg.x` hat, dann soll der Aufruf

```
./reverse_each_arg.x abcde 123 x xyz
```

die folgende Ausgabe liefern:

```
x.gra-hcae-esrever
edcba
321
x
zyx
```

1.49. Das folgende Programm soll Zahlen von der Standard-Eingabe einlesen und sie in einem Array `arr` speichern. Falls mindestens eine negative Zahl eingelesen wurde, soll der maximale Index einer negativen Zahl in `arr` und die Zahl selbst ausgegeben werden.

```
#include "stdio.h"
#include "stdlib.h"
int main(void) {
  size_t allocated = 5, nextfree = 0;
  long read_long, *last_negative = NULL,
       *arr = malloc(allocated * sizeof *arr);
  while (scanf("%ld",&read_long) == 1)
  {
    if (nextfree >= allocated)
    {
      allocated += (allocated * 0.2) + 128;
      arr = realloc(arr,sizeof *arr * allocated);
    }
    if (read_long < 0)
      last_negative = arr + nextfree;
    arr[nextfree++] = read_long;
  }
  if (last_negative != NULL)
    printf("last␣negative␣value␣%ld␣was␣at␣index␣%lu\n",
                 *last_negative,
                 (size_t) (last_negative - arr));
  free(arr);
  return EXIT_SUCCESS;
}
```

Das Programm funktioniert jedoch nicht immer korrekt. Wenn z.B. die Zahlenfolge 0, 1, 2, -3, 5, 0, 1 zeilenweise eingelesen wird, tritt sehr wahrscheinlich ein Fehler auf. Erläutern Sie am Beispiel dieser Zahlenfolge, warum ein Fehler auftritt und beschreiben Sie, was verändert werden muss, um den Fehler zu korrigieren.

1.50. Wenn man beim n-Queens Problem nicht nur die Anzahl der Lösungen zählen will, sondern auch noch die Lösungen selbst aufzählen möchte, dann muss man jede Platzierung einer Königin an Position (i, j) protokollieren. Wie kann man diese Positionen auf einfache Weise protokollieren, ohne Paare von Positionen zu speichern? Wie kann man dabei Zeilen erkennen, in denen noch keine Königin platziert wurde?

1.51. Seien `a` und `b` Zeiger auf aufsteigend sortierte `int`-Arrays der Längen `alen` und `blen`. Implementieren Sie eine C-Funktion

```
size_t count_match_pairs(const int *a,size_t alen,
                         const int *b,size_t blen)
```

die die Anzahl der Paare `(i,j)`, $0 \leq$`i`$\leq$`alen-1`, $0 \leq$`j`$\leq$`blen-1` zählt, so dass `a[i]==b[j]` gilt. Die Laufzeit soll proportional zur `alen+blen` sein.

2 Aufgaben zu numerischen und kombinatorischen Problemen

2.1. Nehmen Sie an, Sie wollen ein Spiel implementieren, in dem zwei sechsseitige Würfel verwendet werden, und zwar ein gezinkter Würfel und ein fairer Würfel:

- beim gezinkten Würfel wird bei jedem Wurf mit einer Wahrscheinlichkeit von 0.5 eine 6 gewürfelt und jeweils mit einer Wahrscheinlichkeit von 0.1 die anderen 5 Zahlen.
- beim fairen Würfel wird jede Zahl mit der gleichen Wahrscheinlichkeit von $\frac{1}{6}$ gewürfelt.

Implementieren Sie zwei Funktionen

```
int gezinkter_wuerfel(void)

int fairer_wuerfel(void)
```

die das obige Verhalten implementieren, indem Sie die entsprechenden Zahlen mit den angegebenen Wahrscheinlichkeiten liefern. Beide Funktionen sollen mit der C-Funktion `double drand48(void)` Pseudozufallszahlen berechnen. In den zu implementierenden Funktionen darf `drand48()` jeweils nur genau einmal aufgerufen werden. Um die Initialisierung des Seeds für den Zufallsgenerator brauchen Sie sich nicht zu kümmern.

2.2. Schreiben Sie eine C-Funktion

```
char *random_sequence_prob(const char *alphabet,const double *prob,size_t n)
```

die eine Zufallssequenz der Länge `n` über dem Alphabet `alphabet` in einer Laufzeit, die proportional zu $n \log_2 k$ ist, berechnet. k ist die Größe des Alphabets, d.h. die Länge des `\0`-terminierten Strings `alphabet`, der keine Duplikate enthält. k ist wesentlich kleiner als n. `prob` ist ein Zeiger auf einen Speicherbereich mit genau `k` `double`-Werten, deren Summe 1 ist. Für das i-te Zeichen des Alphabets mit $0 \leq i \leq k-1$ und für alle Positionen ist die Wahrscheinlichkeit des Vorkommens `prob[i]`.

Der Rückgabewert der Funktion soll ein Zeiger auf einen Speicherbereich mit der Zufallssequenz sein. Die Sequenz ist nicht `\0`-terminiert. Benutzen Sie die Funktion `drand48()` zum Generieren von Zufallszahlen. Sie können davon ausgehen, dass `drand48()` konstante Zeit benötigt. Um die Initialisierung des Seeds für den Zufallsgenerator brauchen Sie sich nicht zu kümmern.

Beispiel: Falls `alphabet` auf den String `"acgt"` zeigt und `prob` auf das Array mit den Werten `0.1`, `0.2`, `0.3`, `0.4`, dann liefert `random_sequence_prob(alphabet,prob,n)` eine Zufallssequenz der Länge n in der `a` etwa $0.1n$ mal vorkommt, `c` etwa $0.2n$ mal vorkommt, `g` etwa $0.3n$ mal vorkommt, und `t` etwa $0.4n$ mal vorkommt.

2.3. Sei $n \geq 0$ und p eine Funktion, die für alle i, $0 \leq i \leq n-1$ eine nicht negative Zahl $p(i)$ liefert. p kann man als Verteilung interpretieren. Eine typische Fragestellung besteht darin zu ermitteln, ob eine beobachtete Verteilung q sich von einer gegebenen bekannten Verteilung p unterscheidet. Dazu kann man den Chi-Square Test verwenden, der für p und q die folgende Summe auswertet:

$$\sum_{i=0}^{n-1} \frac{(p(i)-q(i))^2}{p(i)}$$

Sei p durch ein Array `p_arr` der Länge n über dem Basistyp `size_t` gegeben, so dass `p_arr[i]`$= p(i)$ für alle i, $0 \leq i \leq n-1$. Analoges gilt für q und ein Array `q_arr`.

Implementieren Sie eine C-Funktion

```
double chi_square(const size_t *p_arr,
                  const size_t *q_arr,size_t n)
```

die obige Summe berechnet und zurückliefert. Dabei zeigen `p_arr` und `q_arr` auf die beiden oben genannten Arrays.

2.4. Durch die begrenzte Anzahl von Bytes zur Repräsentation von ganzen Zahlen können bei der Addition und Multiplikation Berechnungsfehler auftreten. Das passiert insbesondere dann, wenn die zu addierenden bzw. multiplizierenden Werte zu groß werden, so dass ein Überlauf (overflow) entsteht. Schreiben Sie eine C-Funktion `safe_add`, die das Ergebnis der Addition von zwei `unsigned long`-Werten `a` und `b` liefert, falls kein Überlauf auftritt. Falls bei der Addition von `a` und `b` ein Überlauf auftreten würde, soll die Funktion eine sinnvolle Fehlermeldung generieren und mit `exit` abbrechen. Verwenden Sie für die Fallunterscheidung die in der Vorlesung vorgestellte Technik.

2.5. Zur Berechnung des Binomialkoeffizienten $\binom{n}{k}$ kann die folgende Rekurrenz verwendet werden:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \text{ für } n > k > 0, \quad \binom{n}{0} = \binom{n}{n} = 1 \text{ für } n \geq 0, \quad \binom{0}{k} = 0 \text{ für } k > 0$$

Die direkte Implementierung dieser Rekurrenz durch eine rekursive Funktion würde zu einer sehr großen Anzahl von Funktionsaufrufen führen. Daher verwendet man die Technik der Dynamischen Programmierung mit einer Tabelle `tab`, in der die Ergebnisse von Funktionsaufrufen gespeichert werden, sobald sie das erste Mal berechnet wurden. `tab` enthält Werte vom Typ `Definedvalue`, der wie folgt deklariert ist.

```
typedef struct {
  size_t value;
  bool defined;
} Definedvalue;
```

Implementieren Sie in einer rekursiven C-Funktion

```
size_t evalbinom_memo(Definedvalue **tab,
                      size_t n, size_t k)
```

die obige Rekurrenz effizient mit Hilfe der Tabelle `tab`. Gehen Sie davon aus, dass `tab` auf bereits allokierten Speicherplatz für ein zweidimensionales Array passender Größe zeigt, in dem zu Anfang alle `defined`-Werte mit `false` initialisiert sind.

2.6. Die Funktion Γ ist für alle positiven ganzen Zahlen p definiert durch $\Gamma(p) = \prod_{i=1}^{p-1} i = (p-1)!$. Die C-Funktion `lgamma` berechnet für alle positiven ganzen Zahlen p den Wert von $\ln \Gamma(p)$. Wie kann man mit Hilfe der C-Funktionen `lgamma` und `exp` aus der mathematischen C-Bibliothek den Wert des Ausdrucks $\binom{n}{k}$ für $k \leq n$ näherungsweise ohne Multiplikation und ohne Division berechnen? Geben Sie ein entsprechendes C-Programmfragment an.

Zur Erinnerung: Es gilt $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ für alle $n \geq k \geq 0$.

2.7. In der Vorlesung wurden unimodale Funktionen betrachtet. Sei $f : \mathbb{R} \to \mathbb{R}$ eine Funktion und $[\ell, r]$, mit $\ell, r \in \mathbb{R}$ und $\ell \leq r$ ein Intervall. Geben Sie die zwei Bedingungen an, die f erfüllen muss, damit es eine unimodale Funktion mit Minimum $x_{\min}$, $\ell \leq x_{\min} \leq r$ ist.

2.8. Beschreiben Sie den in der Vorlesung vorgestellten Algorithmus zur Suche eines globalen Minimums einer unimodalen Funktion in einem gegebenen Wertebereich. Welche Rolle spielt dabei der „goldene Schnitt“? Sie müssen hier keinen Pseudocode oder Programmcode angeben.

2.9. Definieren Sie den Begriff *Maschinen-Genauigkeit* in der Fließkomma-Arithmetik, wie er in der Vorlesung eingeführt wurde. Geben Sie ein C-Programmfragment an, das die Maschinen-Genauigkeit ϵ für Werte vom Typ `double` berechnet.

2.10. In der Vorlesung wurde das Divide und Conquer-Verfahren zur Multiplikation einer $m \times n$-Matrix A mit einer $n \times \ell$-Matrix B im Detail erläutert. Beschreiben Sie dieses Verfahren, gerne auch durch eine Skizze. Pseudocode ist nicht notwendig, aber die Teilungssschritte sollen dabei verständlich erläutert werden.

3 Aufgaben zur C++-Programmierung

3.1. Identifizieren und korrigieren Sie den syntaktischen Fehler in der folgenden C++-Anweisung:

```
std::cin << wert;
```

3.2. Identifizieren und korrigieren Sie den logischen Fehler in der folgenden C++-Anweisung:

```
if (wert == 1)
  std::cout << "wahr" << std::endl;
else;
  std::cout << "falsch" << std::endl;
```

3.3. Zu welchem Zweck wird in C++ ein `namespace` verwendet?

3.4. Nehmen Sie an, dass das letzte Argument `int n` einer C++-Funktion in fast allen Aufrufen den gleichen Wert `1` hat. Welche Form der Deklaration von Parametern erscheint hier sinnvoll und wie sollte in diesem Fall der Funktionskopf deklariert werden?

3.5. Deklarieren Sie in C++-Syntax die Schnittstelle einer Klasse `Protein` mit zwei privaten Membervariablen `name` (für den Namen des Proteins als Zeiger auf ein Array mit Basistyp `char`)) und `length` (für die Länge als Wert vom Typ `size_t`) sowie vier `public`-Methoden, nämlich zwei zum Setzen und zwei zum Lesen dieser Membervariablen. Verwenden Sie für diese Methoden jeweils als Präfix den Namen der Membervariablen und als Suffix `_get` bzw. `_set`. Benutzen Sie keine `inline`-Funktionen, d.h. die Klassendefinition enthält nur die Funktionsköpfe. Eine Implementierung der Methoden selbst ist nicht erforderlich. Wie heißen die default-Konstruktoren bzw. -Destruktoren für diese Klasse?

3.6. Der Absolutwert einer komplexen Zahl $x = a + b\mathtt{i}$ ist $|x| = \sqrt{a^2 + b^2}$. Für zwei komplexe Zahlen $x = a + b\mathtt{i}$ und $y = c + d\mathtt{i}$ ist das Produkt xy definiert durch $xy = ac - bd + (bc + ad)\mathtt{i}$. Seien komplexe Zahlen durch die Klasse `Complex` in C++ implementiert:

```
class Complex {
  private: int real, imag;
  public:
    Complex(int r,int i) {
      real = r;
      imag = i;
    }
    std::string to_string() const {
      return std::string("(") + std::to_string(real) + std::string("+") +
             std::to_string(imag) + std::string("i)");
    }

};
```

`std::to_string` liefert für eine Zahl die String-Darstellung in Form einer Instanz der Klasse `std::string`. Implementieren Sie innerhalb dieser Klasse zwei inline-Methoden

```
double absolute(void) const
Complex operator* (const Complex &other) const
```

zur Berechnung des Absolutwertes einer komplexen Zahl und zur Multiplikation zweier komplexer Zahlen. Da diese Methoden innerhalb der Klasse implementiert werden, können Sie auf die privaten Member-Variablen zugreifen. Die Methoden sollen so implementiert werden, dass das folgende Programm die daneben stehende Ausgabe liefert:

```
Complex a = Complex(2,4);
std::cout << a.absolute() << std::endl;
Complex b = Complex(3,1);
std::cout << a.to_string() << "*" << b.to_string()
          << "=" << (a * b).to_string() << std::endl;
```

```
4.47214
(2+4i)*(3+1i)=(2+14i)
```

3.7. Was ist ein C++-Funktionstemplate, wie wird es verwendet und worin liegen die Vorteile der Verwendung von Templates? Hier ist es nicht notwendig, Programmcode anzugeben.

3.8. Schreiben Sie in C++ ein Funktionstemplate `getmin` für die Berechnung des Minimums von zwei Werten, deren Typ als Template-Parameter deklariert wird.

3.9. Schreiben Sie in C++ ein Funktionstemplate `swap` für das Vertauschen der Werte von zwei Variablen, deren Typ als Template-Parameter deklariert wird. Verwenden Sie Referenzen für die Parameter, aber keine Zeiger.

3.10. Implementieren Sie in C++ eine template-basierte Funktion `distance_template`, die vier Parameter x_1, y_1, x_2, y_2 eines unbekannten aber gleichen numerischen Typs erhält. Diese repräsentieren zwei Punkte (x_1, y_1) und (x_2, y_2) im Euklidschen Raum. Die genannte Funktion soll die Distanz der beiden Punkte als `double`-Wert liefern.

3.11. Implementieren Sie in C++ eine template-basierte Funktion `rechteck_flaeche`, die zwei Parameter `l` und `r` eines unbekannten numerischen Typs T erhält. Die beiden Werte repräsentieren die Seitenlängen eines Rechtecks und die genannte Funktion soll die Fläche dieses Rechtecks als Wert vom Typ T liefern.

3.12. Nennen Sie drei in der Vorlesung genauer besprochenen Container-Klassen der Standard-Template Library in C++ und erläutern Sie, was sie repräsentieren.

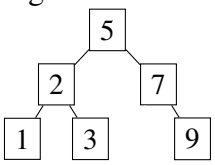
3.13. Wie zählt man typischerweise die Elemente eines Containers aus der C++- Standard-Template Library auf? Geben Sie entsprechenden C++-Programmcode an, der das Vorgehen beispielhaft zeigt.

3.14. In den Übungen haben Sie Techniken zur Implementierung binärer Bäume in C kennengelernt. Hier sehen Sie den Anfang der Deklaration einer C++-Klasse, die auf den gleichen Techniken basiert. Aus Platzgründen zeigen wir nur den Konstruktor, die private rekursive Methode `print_rec` und ihren Aufruf in der public-Methode `print` zur Ausgabe des Binärbaums.

```
template<typename T>
class Bintree
{
  struct Node {
    Node *left, *right;
    size_t count;
    T data;
    Node(const T &_data)
      : left(nullptr), right(nullptr), count(1), data(_data)
     {}
  };
  void print_rec(size_t d,const Node *node) const {
    if (node != nullptr) {
      print_rec(d+1,node->right);
      for (size_t i = 0; i < 3*d; i++) std::cout << "␣";
      std::cout << node->data << "␣" << node->count << std::endl;
      print_rec(d+1,node->left);
    }
  }

  Node *root;
  public:
  Bintree(void) : root(nullptr) {}
  void print(void) const { print_rec(0,root); }
```

Darstellung eines Binärbaums mit Werten vom Typ `T=int`. Die `count`-Werte sind nicht dargestellt.



Implementieren Sie in dieser Klasse eine Methode `size()`, die mit einer `return`-Anweisung die Anzahl der Knoten des durch `root` repräsentierten Binärbaums liefert. Z.B. soll für den oben rechts dargestellten Baum der Rückgabewert 6 sein. Da die Methode keinen Parameter hat, kann sie selbst nicht rekursiv sein, aber sie kann eine rekursive private Methode aufrufen, die Sie implementieren müssen.

4 Aufgaben zu Shared Memory Multithreading

4.1. Definieren Sie den Begriff *Thread*, wie er in der Informatik verwendet wird.

4.2. Wir betrachten die folgende rekursive Funktion `f` und eine Funktion `eval_show_f`, die `f` aufruft und den berechneten Wert ausgibt.

```
size_t f(size_t n) { return n<=1 ? 1 : (f(n-1) + f(n-2)); }

void eval_show_f(size_t n) {
  std::cout << "f(" << n << ")=" << f(n) << std::endl;
}
```

Implementieren Sie in C++ unter Verwendung der Klasse `std::thread` eine Funktion

```
void eval_show_f_multithreaded(const size_t *values,size_t k)
```

die `eval_show_f` in `k` Threads gleichzeitig aufruft. `k` ist auch die Anzahl der Werte im Speicherbereich, auf den `values` zeigt und für alle `i`, $0 \leq \texttt{i} \leq \texttt{k} - 1$ wird ein eigener Thread, der `eval_show_f(values[i])` ausführt, gestartet.

Beispiel: Durch die folgenden Programmzeilen

```
size_t values[] = {39,42,41};
size_t k = sizeof values/sizeof values[0];
```

```
eval_show_f_multithreaded(values,k);
```

sollen die Funktionsaufrufe `eval_show_f(39)`, `eval_show_f(42)`, `eval_show_f(41)` in drei Threads gleichzeitig erfolgen.

4.3. Die rekursive Funktion

```
size_t f(size_t n) { return n<=1 ? 1 : (f(n-1) + f(n-2)); }
```

soll auf alle Werte in einer Tabelle `table` von `num_tasks` ganzen Zahlen vom Typ `size_t` angewendet werden, und zwar so, dass an der i-ten Stelle in der Tabelle nach Ausführung von `f` der Wert `f(table[i])` steht. Wenn man nur einen Thread verwendet, kann man z.B. die folgende Schleife implementieren:

```
for (size_t i = 0; i < num_tasks; i++) table[i] = f(table[i]);
```

Hier geht es aber darum, die Werte in `table` mit mehreren Threads gleichzeitig zu berechnen. In der Vorlesung wurde gezeigt, wie man eine Funktion

```
void pfn_run_threaded(size_t k, size_t num_tasks,
                      PfNThreadFunc thread_proc, void *thread_data);
```

implementiert, die eine Funktion `thread_proc` mit `k` Threads auf Werte zwischen 0 und `num_tasks-1` anwendet. Jeder dieser Werte kann als Nummer einer Aufgabe interpretiert werden. Der Typ `PfNThreadFunc` ist dabei wie folgt definiert:

```
typedef void (*PfNThreadFunc)(size_t thread_id, size_t task_num,
                              void *thread_data);
```

Es wird nun jede Aufgabennummer als Index in `table` interpretiert, an dem zunächst das Argument `n` für die Funktion `f` steht und dann der Funktionswert `f(n)` gespeichert wird.

Implementieren Sie nun eine Funktion `eval_f`, so dass

```
pfn_run_threaded(k, num_tasks, eval_f, (void *) table);
```

die Funktion `f` mit `k` Threads auf `num_tasks` Werte in `table` anwendet. Es soll das gleiche Ergebnis entstehen, wie bei der Berechnung mit einem Thread.

4.4. Durch welche in der Vorlesung besprochene Technik kann man Threads synchronisieren, wenn sie auf gemeinsame Variablen zugreifen und mindestens ein Thread den Wert der Variablen ändert?

5 Aufgaben zur R-Programmierung

5.1. Welchen Datentyp hat die Variable `a` nach der folgenden Wertzuweisung in R?

```
a = c(1.0, 2.0, 3.0)
```

5.2. Welchen Datentyp hat die Variable `a` nach der folgenden Wertzuweisung in R?

```
a = c(F, T, F)
```

5.3. Geben Sie den Inhalt der Variablen `c` nach den folgenden Wertzuweisungen an.

```
a = c(1, 2, 3)
b = c(4, 5, 6)
c = a + b
```

5.4. Sei `a` ein Vektor mit numerischen Werten. Das folgende R-Script berechnet die Summe der Elemente in `a`.

```
s <- 0
for (x in a) {
     s <- s + x
}
```

Schreiben Sie eine einzelne Anweisung in R, die das gleiche Ergebnis berechnet.

5.5. Gegeben sei die folgende Funktionsdefinition in R:

```
sumsq <- function (x) {
    s <- 0
    for (a in x) {
         s = s + a * a
    }
    return (s)
}
```

Für einen Vektor `numbers` kann man die Funktion z.B. in der folgenden Anweisung verwenden:

```
cat("Ergebnis:␣",sumsq(numbers))
```

Die Funktion `sumsq()` ist nicht effizient. Schreiben Sie eine effizientere Variante der Funktion, ohne eine explizite Iteration zu verwenden.

5.6. Wozu kann man in R die folgende Anweisung verwenden?

```
png (file='x.png')
```

5.7. Was bedeutet das Symbol λ in einer exponentiellen Verteilung der Form $\lambda e^{-\lambda t}$?

5.8. Nennen Sie eine von zwei Eigenschaften von Ereignissen, deren Verteilung einer Poisson-Verteilung entspricht.

5.9. Wir betrachten das folgende Fragment eines R-Skriptes:

```
a <- c(1, 4, 16)
b <- sqrt(a)
```

Welchen Typ und welchen Wert hat `b`?

5.10. Worin besteht in R der Unterschied zwischen einer Liste und einem Vektor?

5.11. Wir betrachten das folgende Fragment eines R-Skriptes:

```
f <- function (x) {
```

```
    z <- 0
    for (y in x) {
        z <- z + y
    }
    z/length(x)
}

k <- f(c(1, 2, 3, 4, 5))
```

Welchen Wert hat `k`?

5.12. Sei `a` ein Vektor mit n numerischen Werten. Schreiben Sie einen mathematischen Ausdruck auf, der nach Auswertung, den Wert der Variable m liefert, der durch die folgende R-Anweisung berechnet wird. Sie dürfen in Ihrem Ausdruck nicht die Logarithmus-Funktion verwenden.

```
m = exp(sum(log(a)))
```

5.13. Wir betrachten das folgende Fragment eines R-Skriptes:

```
team1_result <- rpois(1, mu)
```

Was bedeutet `mu` in diesem Zusammenhang?

5.14. Wir betrachten eine Bushaltestelle, an der in unregelmäßigen Zeitabständen Busse halten. Der Zeitraum zwischen zwei aufeinanderfolgenden Haltevorgängen der Busse soll mit einer Exponential-Verteilung modelliert werden. Begründen Sie, warum das kein gutes Modell für diese Aufgabe ist, selbst wenn die Zeitpunkte seltene Ereignisse sind.

5.15. Was ist der Typ und Inhalt von `a` nach der folgenden Anweisung?

```
a <- rep (c(1, 0), c(20, 40))
```

5.16. Ich möchte die Anzahl der Fische in einem See schätzen. Ich fange 100 Fische und markiere sie. Ich werfe sie zurück in den See und fange einen Tag später 100 Fische. Von ihnen sind 5 markiert. Ich schätze, es gibt 2 000 Fische im See. Ich benutze einen Bayes-Ansatz und die statistische Modellierung, um meine Fehler abzuschätzen.

Ein alter Fischer sagt, er habe die Fische kürzlich gezählt und 3 000 mit einer Standardabweichung von 100 gefunden. Wie würden Sie diese zusätzliche Information in einer Stichprobensimulation im Bayes-Kontext kombinieren?

Thema	Anzahl Fragen
Aufgaben zur C-Programmierung	51
Aufgaben zu numerischen und kombinatorischen Problemen	10
Aufgaben zur C++-Programmierung	14
Aufgaben zu Shared Memory Multithreading	4
Aufgaben zur R-Programmierung	16
	$\sum$ 95