Technische Informatik 2

Scheinbedingungen für den Übungsbetrieb

Wintersemester 2023/24

Diese Scheinbedingungen und Hinweise klären insbesondere die Abgabe und die Bewertung der Übungsblätter. Um spätere Missverständnisse zu vermeiden, schreiben wir dabei auch Details fest, die in anderen Veranstaltungen nur implizit angenommen werden.

Grundsätzlich gilt: Wenn Ihr eine Frage oder ein Problem habt, dann sprecht uns gleich an. Wir suchen dann gemeinsam eine Lösung. Kommt Ihr erst später (z. B. nach einer Abgabefrist) zu uns, dann gibt es möglicherweise keine Lösung mehr für Euer Problem, was schade wäre.

I. Scheinbedingungen

- Erfolgreiche Bearbeitung aller Übungsblätter + Gruppen-Fachgespräch.
- Eine Übungsgruppe besteht aus maximal drei Studierenden.
- Übungsblätter wöchentlich; voraussichtlich neun Übungsblätter. Abgabe fristgerecht und ausschließlich über StudIP.
- Alle Übungsblätter sind zu bearbeiten; als bearbeitet gilt ein Übungsblatt, wenn mindestens zu einer Aufgabe des Übungsblatts ein ernsthafter Lösungsvorschlag vorliegt.
- Insgesamt müssen mindestens 50 Prozent der möglichen Punkte erreicht sein.
- Ca. 10 Punkte pro Übungsblatt, insgesamt ca. 90 Punkte.
- Fachgespräch: individuelle Leistung feststellen; ca. zehn Minuten je Teilnehmer/in.
- Die Vornote für das Fachgespräch ergibt sich aus den im Übungsbetrieb erreichten Punkten (4,0 ab 50 Prozent, darüber in 5-Prozent-Schritten).

- Die Endnote ergibt sich aus der Note aus dem Übungsbetrieb und der Note aus dem Fachgespräch mit einer Gewichtung von 40 % Übung und 60 % Fachgespräch. Beide Teile müssen bestanden sein.
- Fachgespräch: Inhalt der Vorlesung, Inhalt der Übungsblätter, in TI2 angewandte Aspekte von C/C++.
- Es gelten die grundsätzlichen Wiederholungsregeln des AT-BPO.
- Bei Nichtbestehen des Übungsbetriebs kann im Sommersemester als Ersatz eine Klausur geschrieben werden bzw. im folgenden Wintersemester der Übungsbetrieb wiederholt werden.
- Bei Nichtbestehen des Fachgesprächs kann es in Folgesemestern wiederholt werden.

II. Hinweise

1. Leistungsnachweise

Es werden keine unbenoteten Teilnahmescheine ausgestellt. Ausnahmen bilden Gasthörer, die z. B. für eine Einrichtung in ihrem Heimatland eine entsprechende Bestätigung benötigen. Diese melden sich bitte in den ersten Vorlesungswochen bei uns.

2. Übungsbetrieb

Die Übungsblätter zur Veranstaltung werden i. d. R. wöchentlich am Donnerstagabend online bereitgestellt, in den Tutorien in der Woche danach besprochen und sind dann eine weitere Woche später bis 23:59 Uhr am Donnerstag digital über das StudIP-Modul DoIT! zu hinterlegen. Abgaben auf anderem Weg — etwa per E-Mail — werden nicht anerkannt.

Gruppen

Die Übungsblätter sind von Euch in festen Gruppen von drei Teilnehmer/inne/n zu bearbeiten. Einzelne Teilnehmer oder Gruppen aus weniger als drei Personen können durch einen Tutor einer anderen Gruppe zugeschlagen werden, sofern sich so eine neue Gruppe mit bis zu drei Mitgliedern ergibt. Je nach Auslastung der Tutoren können die Abgaben einzelner Gruppen von einem anderen Tutor korrigiert werden.

Auf begründeten Antrag können Einzelabgaben genehmigt werden.

Alle Gruppenmitglieder müssen in StudIP als Gruppe eingetragen sein. **Beim Wechsel einer Gruppe ist die/der Tutor/in zu verständigen,** um die Austragung aus DoIT! vorzunehmen.

Lösungsvorschlag

Euer Lösungsvorschlag muss grundlegenden wissenschaftlichen Ansprüchen gewachsen sein. Neben der eigentlichen „Lösung“, also z. B. einer Tabelle mit Ergebniswerten, einem kurzen Absatz mit der Antwort auf eine Frage oder dem Quelltext der von Euch erstellten Funktionen gehört auch die Herleitung dieser Lösung dazu. **Euer Tutor soll verstehen können, wie Ihr Euren Lösungvorschlag erarbeitet habt.** (Das ist auch deswegen nützlich, weil es manchmal auch dann Punkte für den Lösungsweg gibt, wenn die eigentliche Antwort falsch ist.)

Falls Ihr Literatur (inkl. Websites und andere Arten von Quellen) verwendet: benennt diese. Sie darf gerne hilfreich sein, aber **Ihr müsst Eure Lösung eigenständig erarbeiten**, d. h. es ist nicht zulässig, ganze Abschnitte (Text oder Quellcode) zu übernehmen. Falls Ihr Euren Ansatz mit einer anderen Gruppe diskutiert, so muss diese ebenfalls auf den **Abgaben beider Gruppen genannt** werden (s. u.).

Testen

Von Euch implementierte Funktionen sind immer zu testen. In einigen Fällen geben wir konkrete Testfälle vor. In anderen müsst Ihr diese selbst erstellen und begründen, warum Ihr sie als angemessen erachtet, um die Korrektheit und Vollständigkeit Eures Lösungsvorschlags nachzuweisen. **Testdokumentation und nachvollziehbare Testprotokolle müssen in der Abgabe vorhanden sein.**

Die Referenz-Plattform für den Übungsbetrieb sind die x-Rechner im Rechnerpool des Fachbereichs 3 (`x01` bis `x25`; nicht alle sind in Betrieb). Eure Lösungsvorschläge werden im Zweifel auf diesen Rechnern von uns getestet. Ein Tutor kann Abgaben akzeptieren, wenn sie auf seinem eigenen Linux-System korrekt übersetzt und ausgeführt werden können. Lösungen für andere Betriebssysteme (insbesondere Windows, iOS, Android) werden nicht akzeptiert.

Abgabe

Für die „schriftlichen“ Teile der Abgabe steht in StudIP ein **LaTeX-Template** bereit. Dieses ist **von allen Gruppen zu benutzen**. In der abgegebenen Fassung muss sichergestellt sein, dass Sonderzeichen, Umlaute und sz-Ligatur im erzeugten PDF-Dokument korrekt angezeigt werden. **Aufgabenlösungen mit „verschluckten“ Sonderzeichen (die in LaTeX nicht korrekt maskiert worden sind), werden nicht bewertet.**

Die Abgabe erfolgt ausschließlich über StudIP nach den in der Veranstaltung kommunizierten Abgabekonventionen. Handschriftliche Abgaben — auch eingescannt oder am Tablet erzeugt — werden grundsätzlich nicht akzeptiert und mit null Punkten für die jeweilige Aufgabe gewertet.

Die Abgabe erfolgt als **zip-Archiv** in DoIT. Folgende Dateien müssen darin enthalten sein: unsere jeweiligen Vorgaben, die Quelltexte zu Euren Lösungen und Euer Abgabeblatt als **LaTeX-Quelldatei und als PDF-Dokument**. So sind wir schnell in der Lage, Eure Lösung selbst zu übersetzen und zu testen. Programmcode ist als kommentierter Quelltext abzugeben und **muss zudem in der schriftlichen Abgabe hinreichend erklärt werden (Programmdokumentation)**. Neben von Euch erzeugtem Quellcode müssen auch immer die vorgegebenen Quelldateien und Makefiles in dem Abgabeordner abgelegt sein. Im Anhang dieser Scheinbedingungen findet sich ein Beispiel, wie eine Dokumentation des Programms und der Testfälle aussehen könnte. Für mangelnde Sorgfalt und daraus resultierende Programmierfehler erfolgt in jeder Aufgabe ein automatischer Punktabzug entsprechend der Aufstellung im Anhang.

Das von Euch abgegebene PDF-Dokument trägt einen Namen, der sich aus der Nummer des Übungsblatts, Eurer Gruppenbezeichnung und den Nachnamen der beteiligten Gruppenmitglieder nach folgendem Muster zusammensetzt: `n_gruppe_name1_name2_name3.pdf`. Für das erste Übungsblatt wäre dies z. B. `1_H04_meyer-hansel_klotzhuber_muench.pdf` für Gruppe 4 im Tutorium H, bestehend aus den Gruppenmitgliedern mit den Nachnamen Meyer-Hansel, Klotzhuber und Münch. Die Namen von Euch abgegebenen Dateien dürfen nur die ASCII-Zeichen [a-zA-Z0-9._-] enthalten. **Insbesondere sind keine Leerzeichen oder Umlaute in Dateinamen erlaubt und werden bei der Korrektur nicht beachtet.**

Die abzugebende Archivdatei folgt ebenfalls dieser Namenskonvention, aber die Endung muss entsprechend `.zip` sein.

Die Gruppennummern werden zu Beginn des Übungsbetriebs ausschließlich von Eurem Tutor vergeben.

3. Fragen und Antworten

Rückfragen zur Veranstaltung können zunächst in der Vorlesung selbst und in den Tutorien gestellt werden. Fragen in Online-Meetings sollten möglichst über Mikrofon gestellt werden, da Textchat unnötig aufhält. Die Veranstalter und Tutoren ignorieren Textkommunikation während der Sitzung, wenn diese damit unnötig in die Länge gezogen würde.

Des weiteren sind der Mattermost-Kanal und das Forum in der StudIP-Präsenz zu TI2 die primären Orte für Fragen, Antworten und Anmerkungen. **Ankündigungen erfolgen**

mündlich in Vorlesungsterminen, in Online-Live-Sitzungen, in Tutorien und bei Bedarf über die StudIP-Seite zu dieser Veranstaltung. Es obliegt den an der Veranstaltung Teilnehmenden, sich notwendige Informationen selbständig einzuholen.

Natürlich stehen Euch alle Tutoren auch außerhalb der Tutorien gerne zur Verfügung, aber schaut bitte zuerst in Mattermost, bzw. StudIP nach, ob Eure Frage möglicherweise schon beantwortet wurde — das spart uns allen Wiederholungen und wir können unsere Zeit für „neue“ Probleme nutzen.

Wir verwenden zur Kontaktaufnahme stets Eure Universitäts-E-Mail-Adresse und bitten Euch daher, Euch ggf. eine Weiterleitung einzurichten: https://onlinetools.zfn.uni-bremen.de/.

4. Täuschungsversuch und Störung der Prüfung

Ihr befindet Euch während der gesamten Veranstaltung in einer Prüfungssituation. Das heißt, dass sowohl ein Täuschungsversuch selbst als auch die Störung der Prüfung zum Ausschluss von der Prüfung führen kann: „Kein Schein“.

Zur Verdeutlichung der Begriffe „Täuschungsversuch“ und „Störung der Prüfung“ aus dem „Allgemeinen Teil der Bachelor-Prüfungsordnungen der Universität Bremen“ (Paragraph 18, Abs. 2 und 3) legen wir fest:

1. Abschreiben gilt als Täuschungsversuch.
2. Wir begrüßen die Zusammenarbeit von Gruppen bzw. deren Mitgliedern während der Bearbeitung! Dies darf jedoch nicht zu gleichen Abgaben führen. Die individuelle Leistung der Gruppen muss erkennbar sein.
   Falls mehrere Gruppen signifikant zusammenarbeiten (also z. B. ein Übungsblatt gemeinsam besprechen und dann einen ersten Lösungsansatz entwickeln), so müsst Ihr die jeweils anderen Gruppen in Eurer Abgabe nennen. Dies gilt auch für Wiederholer, die Unterlagen aus einer früheren Veranstaltung wiederverwenden wollen.

   Zuwiderhandlung gilt als Täuschungsversuch.
3. Ein Täuschungsversuch kann zum Scheinverlust führen.
4. Eine Hilfestellung (siehe Ziffer 2) darf nicht dazu führen, dass Gruppen zum Täuschungsversuch ermuntert werden. Die Veranstalter behalten sich das Recht vor, die Betreffenden ggf. aufzufordern, zukünftig für Abhilfe zu sorgen. Offensichtliche Zuwiderhandlung, z. B. das öffentliche Bereitstellen des eigenen Lösungsvorschlags, kann als Störung der Prüfung interpretiert werden.

5. Eigene Lösungen oder Lösungsansätze müssen zu jeder Zeit gegen Einsicht durch Unbefugte geschützt werden. Grob fahrlässiges Verhalten (etwa Verlassen des eigenen Rechners ohne aktivierten Schutz vor unbefugtem Zugriff, Screen-Sharing usw.) wird als öffentliche Bereitstellen des eigenen Lösungsvorschlags im Sinne von Ziffer 4 behandelt. Die gilt auch bei Verwendung von Online-Plattformen für die Zusammenarbeit. Auf Online-Plattformen (z. B. discord) muss aktiv sichergestellt werden, dass die eigene Lösung nicht von anderen einsehbar ist, mit denen keine Zusammenarbeit dokumentiert wurde. Satz 1 gilt ausdrücklich auch nach dem Ablauf der Abgabefrist.
6. Verwendete Quellen (mit Ausnahme der Folien und Videos aus der Vorlesung und den Tutorien zu der aktuellen Veranstaltung) jeglicher Form sind zu benennen, und wörtliche oder sinngemäß übernommene Textstellen sowohl im Textteil des Lösungsvorschlags als auch im Programmcode sind ausdrücklich zu kennzeichnen. Diese Kennzeichnung muss wissenschaftlichen Standards für das Zitieren fremder Werke genügen.
7. Wenn Verdacht auf einen Täuschungsversuch besteht, empfiehlt es sich, die Lösungsvorschläge für die Übungsblätter weiter abzugeben, bis die Sachlage eindeutig geklärt ist (alle Übungsblätter müssen bearbeitet sein, siehe Scheinbedingungen).

Anhang

Punktabzug für mangelnde Sorgfalt

In jeder Aufgabe werden automatisch die folgenden Punktabzüge wegen mangelnder Sorgfalt vorgenommen:

- Unvollständige oder falsche Quellenangaben (insbesondere fehlende oder fehlerhafte Angabe von Autor, Titel, Zeitpunkt der Veröffentlichung, Fundstelle) werden für das jeweilige Übungsblatt pauschal mit einem Abzug von einem Punkt bewertet.
- Programmcode, der nicht kompiliert werden kann, wird als nicht abgegeben bewertet (Ausnahme: es wurde Pseudocode oder nur ein Programmfragment verlangt)
- bis zu 50 % Abzug, wenn Tests und Dokumentation nicht ausreichen
- Programmabsturz zur Laufzeit wird mit -1 P bewertet, sofern es sich nicht um eine ausdrücklich dokumentierte Einschränkung handelt (z. B. komplexe Corner-Cases).

- in jeder Teilaufgabe pauschal mindestens 0.5 Punkte für die folgenden Programmierfehler, im Wiederholungsfall bis zu 50 % der Punkte pro Aufgabe:
  - falsche Datentypen benutzt (z. B. `int` statt `off_t` für `st_size` in `struct stat`)
  - Rückgabewert von Systemaufrufen nicht überprüft, insbesondere `NULL`
  - Speicher nicht freigegeben (z. B. fehlendes `delete` nach `new`)
  - Dateien nicht geschlossen
  - falsch initialisierte Variablen
  - fehlende Rückgabewerte für Funktionen (außer main())
  - Arraygrenzen werden nicht überprüft
  - „Magic Numbers“ statt aussagekräftiger Namen für Konstanten (hängt stark von der Aufgabenstellung ab)
  - Nebenläufigkeit (außer in Aufgaben zu Nebenläufigkeit, dort stärker geahndet): kritische Abschnitte nicht geschützt

Beispiel: Dokumentation einer Programmabgabe

Eine Dokumentation zu einem als kommentiertem Quellcode abgegebenen Programm könnte in dem Haupt-PDF-Dokument folgendermaßen aussehen:

Wir haben ein Programm `fib.cc` geschrieben, das wie gewünscht eine rekursive Berechnung der Fibonacci-Folge durchführt. Das Programm liest nacheinander alle auf der Kommandozeile übergebenen Argumente ein und wandelt sie in Dezimalzahlen um. Falsche Eingaben werden nicht erkannt.

Die rekursive Berechnung der Fibonacci-Folge geschieht in der Funktion `fib()`, die einen Wert vom Typ `unsigned long` als Argument erhält und einen `unsigned long` als Rückgabewert liefert. Als Rekursionsende werden die Fälle `n = 0` und `n = 1` separat behandelt. Wir definieren dafür fib(0) := 0. Für `n > 1` erfolgt die Berechnung durch zweimaligen rekursiven Aufruf.

Schwächen

Die Rückgabe unserer Funktion `fib()` ist ein unsigned long. Auf einer LP64-Plattform ist der Zahlenbereich somit auf $2^{64} = 18446744073709551616$ begrenzt[1]. Das bedeutet, dass das übergebene Argument n dort höchstens 93 betragen darf, da

[1] Auf ILP32-Systemen liegt die Grenze entsprechend niedriger.

$$
\begin{aligned}
fib(93) &= 12200160415121876738 \\
fib(92) &= 7540113804746346429 \\
\\
\Rightarrow fib(94) &= fib(93) + fib(92) \\
&= 12200160415121876738 + 7540113804746346429 \\
&= 19740274219868223167 \\
&\geq 18446744073709551616
\end{aligned}
$$

Unser Programm erkennt den Integer-Überlauf nicht und würde daher ein fehlerhaftes Ergebnis liefern.

Für zu große Eingaben kommt es sogar zu einem Programmabbruch mit einem *Segmentation fault*. Ein Blick auf den Stack-Backtrace im Debugger legt nahe, dass dies mit einer zu hohen Rekursionstiefe zusammenhängt.

Die auf der Kommandozeile übergebenen Argumente werden mit `strtoul()` in einen `unsigned long int` umgewandelt. Wir haben auf eine Fehlererkennung verzichtet, so dass die Umwandlung beim ersten nicht-numerischen Anteil der Eingabe abgebrochen wird. Beispielsweise würde aus der Eingabe `123abc` die Zahl `123`. Wegen der Behandlung als vorzeichenlose Zahlen führt die Eingabe negativer Zahlen zu einem Integer-Unterlauf, was für kleine Absolutwerte wiederum zu sehr hohen `n` und damit den oben angesprochenen Laufzeitfehler führt (siehe Tests).

Außerdem haben wir während unserer Tests einen deutlichen Geschwindigkeitsunterschied zu Referenzimplementierungen im Web[2] wahrgenommen. Wir vermuten, dass dies an der sehr ineffizienten rekursiven Implementierung liegt.

Tests

In der Literatur[3] finden wir die ersten 30 Elemente für die Fibonacci-Folge beginnend bei 1:

1,1,2,3,5,8,13,21,…,317811,514229,832040

Wir zeigen mit unseren Tests die normale Funktionsweise auf und decken die identifizierten Grenzfälle ab.

[2] Wir haben insbesondere mit dem Werkzeug …, verfügbar unter `https://...`, getestet (letzter Zugriff am …).

[3] Vergleiche N. J. A. Sloane: A000045 — Fibonacci Numbers. In: The On-line Encyclopedia of Integer Sequences. 1964. Online verfügbar unter `https://oeis.org/A000045`, zuletzt abgerufen am…

Test 1

Eingabe: 0
erwartetes Ergebnis: `fib(0) = 0`

Testlauf:

```
$ ./fib 0
fib(0) = 0
```

Ergebnis: OK

Test 2

Eingabe: 1
erwartetes Ergebnis: `fib(1) = 1`

```
$ ./fib 1
fib(1) = 1
```

Ergebnis: OK

Test 3

Eingabe: 6
erwartetes Ergebnis: `fib(6) = 8`

```
$ ./fib 6
fib(6) = 8
```

Ergebnis: OK

Test 4

Eingabe: 8
erwartetes Ergebnis: `fib(8) = 21`

```
$ ./fib 8
fib(8) = 21
```

Ergebnis: OK

Test 5

Eingabe: 30
erwartetes Ergebnis: `fib(30) = 832040`

```
$ ./fib 30
fib(30) = 832040
```

Ergebnis: OK

Test 6

Auf einem LP64-System sollte unser Programm für den Eingabewert `93` noch das korrekte Ergebnis 12200160415121876738 liefern.

Eingabe: 93
erwartetes Ergebnis: `fib(93) = 12200160415121876738`

```
$ ./fib 93
fib(93) = 12200160415121876738
```

Ergebnis: OK

Test 7

Für die Eingabe `94` müsste eine korrekte Implementierung das Ergebnis 19740274219868223167 liefern. Wie zuvor beschrieben, wird dieser Test mit unserem Programm auf einem LP64-System fehlschlagen.

Eingabe: 94
erwartetes Ergebnis: `fib(94) = 19740274219868223167`

```
$ ./fib 94
fib(94) = 1293530146158671551
```

Ergebnis: FAIL

Das Fehlschlagen dieses Tests wurde im Rahmen der zuvor beschriebenen Einschränkungen erwartet.

Test 8

Die Eingabe `12abc 8 xyz` führt auf unserem System zu drei Ausgaben, nämlich

Eingabe: 12abc 8 xyz
erwartetes Ergebnis:

```
fib(12) = 144
fib(8) = 21
fib(0) = 0
```

```
$ ./fib 12abc 8 xyz
fib(12) = 144
fib(8) = 21
fib(0) = 0
```

Ergebnis: OK

Test 9

Die Eingabe `-2` sollte zu einer Fehlermeldung führen.

Eingabe: -2
erwartetes Ergebnis: *Fehlerhafte Eingabe*

```
$ ./fib -2
Segmentation fault
```

Ergebnis: FAIL

Das Fehlschlagen dieses Tests wurde im Rahmen der zuvor beschriebenen Einschränkungen erwartet.

Kommentierter Quellcode

Der Quellcode ist in unserer Abgabe zu finden unter `src/fib.cc`.

```
#include <iostream>
#include <cstdlib>

using namespace std;

/* Funktion liefert den Fibonacci-Wert für das übergebene Argument n.
 * Überläufe für n > 93 werden nicht erkannt. */
unsigned long fib(unsigned long n) {
  switch (n) {
  case 0: return 0;
  case 1: return 1;
  default:
    /* rekursiver Aufruf zum Berechnen von fib(n): */
    return fib(n-2) + fib(n-1);
  }
}

int main(int argc, char **argv) {
  /* Iterieren über die Argumente auf der Kommandozeile.
   * argv[0] überspringen (Name der Programmdatei). */
  for (--argc, ++argv; argc > 0; --argc, ++argv) {
    /* Umwandeln in positive ganze Zahl, Fehler werden
     * ignoriert. */
    unsigned long n = strtoul(*argv, nullptr, 10);

    /* Ermitteln der Fibonacci-Zahl und Ausgabe */
    cout << "fib(" << n << ") = " << fib(n) << endl;
  }
  return 0;
}
```