Universität Augsburg
Institut für Informatik

Übung zur Vorlesung Informatik 1 WS 2020/21
Fakultät für Angewandte Informatik
Lehrprofessur für Informatik
Dr. Martin Frieb, Johannes Metzger 07.12.2020

Übungsblatt 6

Abgabe spätestens bis: 14.12.2020 10:00 Uhr

- Dieses Übungsblatt soll in den in der Übungsgruppe festgelegten Teams abgegeben werden (Einzelabgaben sind erlaubt, falls noch keine Teamzuteilung erfolgt ist).
- Die **Zeitangaben** geben zur Orientierung an, wie viel Zeit für eine Aufgabe später in der Klausur vorgesehen wäre; gehen Sie davon aus, dass Sie zum jetzigen Zeitpunkt wesentlich länger brauchen und die angegebene Zeit erst nach ausreichender Übung erreichen.

* leichte Aufgabe / ** mittelschwere Aufgabe / *** schwere Aufgabe

Allgemeine Hinweise zu den Programmieraufgaben (Erinnerung):

- Achten Sie, wie auf Übungsblatt 1 erwähnt, bei allen Programmieraufgaben auf *Kompilierbarkeit* und *Einhaltung der Coding Conventions*; auch dann, wenn es nicht explizit im Aufgabentext gefordert ist.
- Gehen Sie davon aus, dass alle Programme in der Programmiersprache C (Spezifikation C89) zu erstellen sind. Das bedeutet unter anderem, dass ausschließlich die Funktionen der Standard-Bibliothek C89 verwendet werden dürfen.
- Kompilieren Sie alle Ihre Programme mit den folgenden *Compiler-Schaltern*: `-Wall -Wextra -ansi -pedantic`. Achten Sie darauf, dass trotz Verwendung dieser Schalter keine Fehler-/ Warnmeldungen erzeugt werden.

Aufgabe 21 * *(Wahrheitstafeln, 10 Minuten)*

In jeder Teilaufgabe sollen Sie für eine oder mehrere Bedingungen Wahrheitstafeln erstellen:

- Die Operanden A und B repräsentieren dabei Teilaussagen, die wahr oder falsch sein können.
- Der Operand 1 repräsentiert eine wahre Aussage.
- Der Operand 0 repräsentiert eine falsche Aussage.

Betrachten Sie in jeder Wahrheitstafel jeweils alle Kombinationen von Wahrheitswerten der Operanden A und B und geben Sie jede Teilaussage einer komplexen Bedingung in einer eigenen Spalte an (siehe Beispiele in Kapitel 6).
Kommt ein Operand mehrmals in einer Bedingung vor, so ist jeweils für jedes Vorkommen derselbe Wahrheitswert einzusetzen.

Sie sollen in jeder Teilaufgabe die Wahrheitstafeln benutzen, um zu zeigen, dass zwei Bedingungen **äquivalent** sind. Hierbei heißen zwei Bedingungen **äquivalent**, wenn sie für jede Kombination von Wahrheitswerten von Operanden denselben Wahrheitswert haben.

Beispiel: Die Bedingungen $A \vee A$ und A sind äquivalent, da sie in jeder Zeile der Wahrheitstafel denselben Wert haben:

A	$A \vee A$
0	0
1	1

a) (*, 1 Minute)
Zeigen Sie: Die Bedingungen $\neg(\neg A)$ und A sind äquivalent.

b) (*, 1 Minute)
Zeigen Sie: Die Bedingungen $A \wedge 1$ und A sind äquivalent.

c) (*, 1 Minute)
Zeigen Sie: Die Bedingungen $A \vee 0$ und A sind äquivalent.

d) (*, 1 Minute)
Zeigen Sie: Die Bedingungen $A \wedge A$ und A sind äquivalent.

e) (*, 1 Minute)
Zeigen Sie: Die Bedingungen $A \wedge (\neg A)$ und 0 sind äquivalent.

f) (*, 1 Minute)
Zeigen Sie: Die Bedingungen $A \vee (\neg A)$ und 1 sind äquivalent.

g) (*, 2 Minuten)
Zeigen Sie: Die Bedingungen $\neg(A \wedge B)$ und $(\neg A) \vee (\neg B)$ sind äquivalent.

h) (*, 2 Minuten)
Zeigen Sie: Die Bedingungen $\neg(A \vee B)$ und $(\neg A) \wedge (\neg B)$ sind äquivalent.

Aufgabe 22 *(Zeichenketten, 34 Minuten)*

Erstellen Sie ein kompilierbares und ausführbares C-Programm mit einer `main`-Funktion und weiteren Funktionen nach folgenden Vorgaben. Kompilieren Sie Ihr Programm mit den Compilerschaltern `-ansi -pedantic -Wall -Wextra` und führen Sie es aus.

a) (**, 6 Minuten)
Implementieren Sie ohne Benutzung von Bibliotheksfunktionen oder Funktionen aus der Vorlesung eine Funktion

```
int string_compare(char v[], char w[])
```

zum Vergleich der Zeichenketten `v` und `w`, die sich genau wie die Bibliotheksfunktion `strcmp` verhält.

b) (**, 6 Minuten)
Implementieren Sie ohne Benutzung von Bibliotheksfunktionen oder Funktionen aus der Vorlesung eine Funktion

```
char *string_cat(char v[], char w[])
```

zum Anhängen der Zeichenkette `w` an die Zeichenkette `v`, die sich genau wie die Bibliotheksfunktion `strcat` verhält.

c) (***, 6 Minuten)
Implementieren Sie ohne Benutzung von Bibliotheksfunktionen oder Funktionen aus der Vorlesung eine Funktion

```
char *string_ncopy(char v[], char w[], int size)
```

zum sicheren Kopieren der Zeichenkette `w` in die Zeichenkette `v`, die sich genau wie die Bibliotheksfunktion `strncpy` verhält.

d) (***, 6 Minuten)
Implementieren Sie ohne Benutzung von Bibliotheksfunktionen oder Funktionen aus der Vorlesung eine Funktion

```
int string_suffix(char v[], char w[])
```

die einen von `0` verschiedenen Wert liefert, falls `w` ein Suffix von `v` ist, und andernfalls `0` liefert.

Beispiel: Der Rückgabewert von `string_suffix("Informatik", "matik")` *ist von* `0` *verschieden.*

e) (*, 10 Minuten)
Legen Sie eine systemunabhängige symbolische Konstante `SMAX` für die maximale Länge von Zeichenketten (inkl. binärer Null) in Ihrem Programm an. Deklarieren Sie in der `main`-Funktion einige Zeichenketten und testen Sie mit diesen die obigen Funktionen. Berücksichtigen Sie dabei insbesondere folgende Fälle:

- Zeichenketten vergleichen: Zeichenketten gleicher und ungleicher Länge, die leere Zeichenkette, gleiche und ungleiche Zeichenketten.
- Zeichenketten kopieren: Die leere Zeichenkette, zu lange und nicht zu lange Zeichenketten.
- Zeichenketten aneinanderhängen: Die leere Zeichenkette, zu lange und nicht zu lange Zeichenketten.
- Suffix-Test: Zeichenketten gleicher und ungleicher Länge, die leere Zeichenkette, lange und kurze Suffixe.

Vergessen Sie nicht, die notwendigen Bibliotheken einzubinden.

Aufgabe 23 *(**, Programme mit Kommandozeilenparametern, 24 Minuten)*

Erstellen Sie ein kompilierbares und ausführbares C-Programm mit einer `main`-Funktion und weiteren Funktionen nach folgenden Vorgaben. Kompilieren Sie Ihr Programm mit den Compilerschaltern `-ansi -pedantic -Wall -Wextra` und führen Sie es aus.

- Legen Sie eine systemunabhängige symbolische Konstante `VMAX` für die maximale Länge von `int`-Feldern in Ihrem Programm an.
- Das Programm soll genau einen Kommandozeilenparameter erwarten. Übergibt der Benutzer weniger oder mehr Kommandozeilenparameter, so soll das Programm eine passende Fehlermeldung auf der Kommandozeile ausgeben und mit einem Fehlerwert ungleich Null abbrechen.
- Entspricht der übergebene Parameter einer der Zeichenketten `-h` oder `--help`, so soll ein hilfreicher Text, der die korrekte Benutzung und die Funktionsweise des Programms beschreibt, auf der Kommandozeile ausgegeben werden.
- Entspricht der übergebene Parameter einer nicht-leeren Zeichenkette, die nur aus Ziffern besteht und nicht mit `0` beginnt, so soll der Parameter in eine ganze Zahl `n` umgewandelt werden:

  - Ist `n` größer als `VMAX`, so soll das Programm eine passende Fehlermeldung auf der Kommandozeile ausgeben und mit einem Fehlerwert ungleich Null abbrechen.
  - Ansonsten sollen `n` ganze Zufallszahlen zwischen einschließlich `0` und `999` in einem `int`-Feld der Länge `VMAX` gespeichert werden.
  - Dann soll der Durchschnitt dieser Zahlen berechnet und ausgegeben werden.
- Hat der übergebene Parameter ein anderes als die beschriebenen Formate, so soll das Programm eine passende Fehlermeldung auf der Kommandozeile ausgeben und mit einem Fehlerwert ungleich Null abbrechen.
- Verwenden Sie wo möglich passende Bibliotheksfunktionen, und legen Sie sinnvolle eigene Funktionen für verschiedene Teilaufgaben an.

Aufgabe 24 *(Eingaben ohne Pufferfehler, 25 Minuten)*

Erstellen Sie ein kompilierbares und ausführbares C-Programm mit einer `main`-Funktion und weiteren Funktionen nach folgenden Vorgaben. Kompilieren Sie Ihr Programm mit den Compilerschaltern `-ansi -pedantic -Wall -Wextra` und führen Sie es aus.

a) (**, 5 Minuten)
Implementieren Sie ohne Berücksichtigung von Pufferfehlern (`EOF`) eine Einlesefunktion

```
int read_command(void)
```

die bei Eingabe eines einzelnen Zeichens `'n'`, `'p'`, `'s'` oder `'q'` den ASCII-Code des eingelesenen Zeichens und bei sonstigen (ungültigen) Eingaben bei Bedarf den Eingabepuffer leert und den Wert `-1` zurückgibt.

b) (**, 5 Minuten)
Implementieren Sie ohne Berücksichtigung von Pufferfehlern (`EOF`) eine Einlesefunktion

```
double read_probability(void)
```

die vom Benutzer eine Dezimalzahl zwischen einschließlich `0.0` und `1.0` einliest. Bei gültigen Eingaben soll die Funktion die eingelesene Zahl zurückgeben. Bei ungültigen Eingaben soll die Funktion bei Bedarf den Eingabepuffer leeren und den Wert `-1.0` zurückgeben.

c) (***, 10 Minuten)
Legen Sie eine systemunabhängige symbolische Konstante `SMAX` für die maximale Länge von Zeichenketten (inkl. binärer Null) in Ihrem Programm an. Implementieren Sie ohne Berücksichtigung von Pufferfehlern (`EOF`) eine Einlesefunktion

```
int read_int(char input[])
```

die vom Benutzer eine Zeichenfolge einliest und in `input` als Zeichenkette speichert. Die Eingabe soll gültig sein, wenn Sie

- nur aus Ziffern besteht
- minimal aus einem und maximal aus `SMAX` Zeichen besteht
- nicht mit `0` beginnt, wenn sie aus zwei oder mehr Zeichen besteht

Bei gültigen Eingaben soll die Funktion `1` zurückgeben. Bei ungültigen Eingaben soll die Funktion bei Bedarf den Eingabepuffer leeren und den Wert `0` zurückgeben.

d) (*, 5 Minuten)
Testen Sie in der `main`-Funktion die obigen Einlesefunktionen, indem Sie jeweils eine Benutzereingabe einlesen und auf der Kommandozeile ausgeben, ob die Eingabe gültig war.