Universität Augsburg
Institut für Informatik

Übung zur Vorlesung Informatik 1 WiSe 2022/2023
Fakultät für Angewandte Informatik
Institut für Informatik
Dr. M. Frieb, Dr. J. Mische, V. Schmid, L. Petrak, P. Schalk 02.12.2022

Übungsblatt 7

Abgabe: 12.12.2022 09:00 Uhr

- Dieses Übungsblatt muss im Team abgegeben werden (Einzelabgaben sind nicht erlaubt!).
- 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

Offener Zoom-Arbeitsraum (OZR):
Da zwischenzeitlich ein falscher Link zum offenen Zoom-Raum veröffentlicht war, finden Sie hier den korrekten Link:

https://uni-augsburg.zoom.us/j/97733319417?pwd=RnZjOVhpZktINlVKTlVPcm1uTEEyUT09

Der offene Zoom-Raum findet immer freitags von 14:00 bis 17:00 Uhr statt. Dort sind ein Mitarbeiter und einige Tutoren anwesend, denen Sie Fragen zum aktuellen Stoff und zum aktuellen Übungsblatt stellen können. Außerdem können Sie dort in Breakout-Sessions mit Ihrem Team oder anderen Studierenden gemeinsam am Übungsblatt arbeiten.

Aufgabe 25 *(Eingabe ohne Pufferfehler, ternärer Operator und switch-case, 21 Minuten)*

Erstellen Sie kompilierbare und ausführbare C-Programme mit je einer `main`-Funktion nach unten folgenden Vorgaben. Kompilieren Sie Ihre Programme mit den Compilerschaltern `-ansi -pedantic -Wall -Wextra` und führen Sie sie aus.

a) *(*, 5 Minuten)*
Implementieren Sie eine Funktion `int read_time()`, mit der eine Uhrzeit im 24-Stunden-Format **ohne Berücksichtigung von Pufferfehlern** vom Benutzer eingegeben werden kann. Die Funktion soll dann die Zeit bis zum bzw. seit dem Ladenschluss (in Bayern: 20 Uhr) berechnen und ausgeben.

- Da verschiedene Bundesländer jedoch verschiedene Ladenschlusszeiten haben, legen Sie entsprechende symbolische Konstanten an, um den Ladenschluss später bei Bedarf leicht anpassen zu können.
- Sorgen Sie für verständliche Ausgaben und Eingabeaufforderungen sowie eine Fehlerbehandlung für ungültige Eingaben.
- Verwenden Sie symbolische Konstanten, um mit dem Rückgabewert der Funktion über den (Miss-)Erfolg der Funktion zu informieren.

- Verwenden Sie außerdem den **ternären ?:-Operator**[1] um die Ausgabe der Funktion auf Kommandozeile abhängig von der eingegebenen Zeit zu bestimmen, und **nicht** `if`-Blöcke.
- Bei ungültigen Eingaben soll die Funktion bei Bedarf den Eingabepuffer **ohne Berücksichtigung von Pufferfehlern** leeren – verwenden Sie dazu die Funktion `flush()` aus der Vorlesung.

b) *(**, 10 Minuten)*
Implementieren Sie eine Funktion `int read_date()`, mit der ein Datumsangabe (ohne Jahr) **ohne Berücksichtigung von Pufferfehlern** vom Benutzer eingegeben werden kann. Die Funktion soll angeben, der wievielte Tag des Jahres das eingegebene Datum ist.

- Ignorieren Sie Schaltjahre.
- Verwenden Sie eine switch-case-Fallunterscheidung, um die Eingabe des Tags abhängig vom eingegebenen Monat zu überprüfen.
- Sorgen Sie für verständliche Ausgaben und Eingabeaufforderungen sowie eine Fehlerbehandlung für ungültige Eingaben.
- Verwenden Sie symbolische Konstanten, um mit dem Rückgabewert der Funktion über den (Miss-)Erfolg der Funktion zu informieren.
- Verwenden Sie außerdem **für die Berechnung, um den wievielten Tag es sich handelt, höchstens einen** `if`-Block und sonst nur den **ternären ?:-Operator**.
- Bei ungültigen Eingaben soll die Funktion bei Bedarf den Eingabepuffer **ohne Berücksichtigung von Pufferfehlern** leeren – verwenden Sie dazu die Funktion `flush()` aus der Vorlesung.

c) *(*, 6 Minuten)*
Implementieren Sie ein Hauptprogramm, das den Benutzer auffordert, entweder `'t'` oder `'d'` einzugeben. Abhängig von dieser Eingabe soll dann mithilfe der Funktionen aus den vorherigen Teilaufgaben entweder die Zeit bis zum Ladenschluss (für die Eingabe `'t'`) ausgegeben werden, oder der wievielte Tag das eingegebe Datum im Jahr ist (für die Eingabe `'d'`) . Sorgen Sie für verständliche Ausgaben und Eingabeaufforderungen sowie eine Fehlerbehandlung für ungültige Eingaben. Leeren Sie bei ungültigen Eingaben bei Bedarf den Eingabepuffer **ohne Berücksichtigung von Pufferfehlern**.

Aufgabe 26 *(Eingabe mit Pufferfehlern, 22 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) *(**, 10 Minuten)*
Legen Sie eine systemunabhängige symbolische Konstante `DIM` für die Anzahl der Bits einer Binärcodierung an. Implemetieren Sie **mit** Berücksichtigung von Pufferfehlern (`EOF`) eine Funktion

```
int read_binary(int b[])
```

die eine vom Benutzer eingegebene Binärcodierung exakt der Länge DIM einliest und in `b` speichert. Dabei soll das zuletzt eingegebene Bit im letzten Eintrag von `b` gespeichert werden. Bei ungültigen Eingaben soll die Funktion bei Bedarf den Eingabepuffer leeren – verwenden Sie dazu die Funktion `flush_buff` aus der Vorlesung.
Unterscheiden Sie über den Rückgabewert der Funktion mögliche Fehlerfälle vom Erfolgsfall und legen Sie für alle möglichen Rückgabewerte geeignete symbolische Konstanten an.

[1] Z.B. gibt `printf("\%s", a ? "true" : "false")` für $a \neq 0$ "true" aus und sonst "false". Siehe auch Skript Definition 6.4.

b) *(**, 12 Minuten)*
Implementieren Sie **mit** Berücksichtigung von Puffer-Fehlern eine Funktion

`int read_and_decode_fk()`

die eine vom Benutzer eingegebene ganze Zahl k als Anzahl der Nachkommastellen einer FK-Codierung in `DIM` Bits interpretiert. Benutzen Sie dann die Funktionen `read_binary` aus Teilaufgabe a) und `decode_fk()` von Blatt 5, um die eingegebene Binärcodierung als FK-Codierung in `DIM` Bits mit k Nachkommastellen zu decodieren und auszugeben. Bei ungültigen Eingaben soll die Funktion bei Bedarf den Eingabepuffer leeren – verwenden Sie dazu die Funktion `flush_buff` aus der Vorlesung.
Unterscheiden Sie über den Rückgabewert der Funktion mögliche Fehlerfälle vom Erfolgsfall und legen Sie für alle möglichen Rückgabewerte geeignete symbolische Konstanten an. Testen Sie Ihre Funktionen mit einer kurzen `main()`-Methode und geeigneten Benutzereingaben.

Hinweis: In dieser Aufgabe haben Sie einen Baustein der Semesteraufgabe zur Gleitkommacodierung implementiert. Die zugehörige und weitere Systembeschreibungen finden Sie ab dem 7.12.22 im Digicampus unter Dateien im Ordner Semesteraufgabe. Sie können als freiwillige Zusatzaufgabe zuhause eine der Systembeschreibungen implementieren. Die Aufgabe wird durch die in den Übungsgruppen vorhandenen Teams bearbeitet. Sie erhalten für die vollständige und korrekte Bearbeitung bis zu 5 Übungspunkte. Die Bewertung erfolgt durch eine Abnahme. Die Teams mit dem besten Programm zu einer Aufgabe erhalten eine Urkunde. Eine Teilnahme mit gutem Erfolg ist empfohlen, wenn man Tutor für Informatik 1 werden möchte.

Aufgabe 27/28 *(Einlesen und Modifizieren von Zeichenketten, 40 Minuten)*

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

a) (**, 3 Minuten)
Betrachten Sie folgendes kurzes Programm

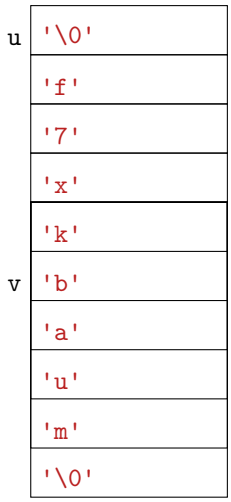
```c
#include <stdio.h>
#define N_VAL 5

void unsafe_read(void)
{
    char u[N_VAL];
    char v[] = "baum";

    scanf("%s",u);
    printf("%s\n", v);
}

int main(void)
{
    unsafe_read();
    return 0;
}
```

sowie folgende Speichersituation:



Geben Sie die Ausgabe des Programms für die Eingabe der Zeichenkette `"Ahorn"` an und begründen Sie Ihre Antwort. Geben Sie außerdem eine Eingabe an, für die das Verhalten des Programms auch in der beschriebenen Speichersituation nicht definiert ist.

b) (**, 6 Minuten)
Legen Sie eine systemunabhängige symbolische Konstante `MAX_STRING` an und schreiben Sie **mit** Berücksichtigung von Pufferfehlern eine Einlesefunktion

```
int read_string(char str[])
```

die eine Zeichenkette mit maximaler Länge `MAX_STRING` (inkl. binärer 0) vom Benutzer einliest und in `str` speichert. Legen Sie geeignete symbolische Konstanten an, um bei zu langen Eingaben oder Puffer-Fehlern Fehlerwerte zurückzugeben.
Hinweis: Unsicheres Einlesen von Zeichenfolgen, wie in Aufgabenteil a) gezeigt, und Einlesen von Zeichenketten durch scanf mit scanf(%s) ist in allen Programmieraufgaben grundsätzlich nicht erlaubt.

c) (*, 3 Minuten)
Schreiben Sie eine Funktion `int check_string(const char str[])`, die überprüft, ob die übergebene Zeichenkette lediglich aus lateinischen Klein- und Großbuchstaben, Satzzeichen und Leerzeichen besteht. Im Erfolgsfall soll 1 zurückgegeben werden, sonst 0. Verwenden Sie wieder symbolische Konstanten für die Rückgabewerte.

d) (**, 10 Minuten)
Schreiben Sie zwei Funktionen

```
void caesar_encode(int n, char str[])
```

und

```
void caesar_decode(int n, char str[])
```

die die Caesar-Codierung[2] anwenden, um die übergebene Zeichenkette entweder zu codieren oder zu decodieren (sprich, die Buchstaben zyklisch um n verschieben). Zeichen, die keine lateinischen Klein- oder Großbuchstaben sind, sollen dabei unverändert bleiben. Gehen Sie davon aus, dass für n nur Werte zwischen 0 und 25 (einschließlich) übergeben werden.

[2]Dabei werden alle Zeichen zyklisch um einen konstanten Wert im Alphabet verschoben: Für den Wert 5 wird also `'A'` zu `'F'`, `'B'` zu `'G'` und `'V'` wird zu `'A'`, `'W'` zu `'B'` usw. Siehe Beispiel 4.27 im Skript oder https://de.wikipedia.org/wiki/Caesar-Verschl%C3%BCsselung

e) (**, 10 Minuten)
Schreiben Sie eine Funktion `int crack_caesar(char str[])`, die zu erraten versucht, mit welcher Zahl die übergebene Zeichenkette nach der Caesar-Codierung verschlüsselt wurde. Zählen Sie dazu (unabhängig von Groß- und Kleinschreibung) die Häufigkeiten der vorkommenden lateinischen Buchstaben und identifizieren Sie den Buchstaben, der am häufigsten vorkommt, als `'E'`[3].

f) (**, 8 Minuten)
Schreiben Sie eine `main`-Funktion, die Funktionen der vorherigen Teilaufgaben verwendet, um

- eine Zeichenkette vom Benutzer einzulesen
- eine Zahl zwischen 0 und 25 vom Benutzer einzulesen
- die eingegebene Zeichenkette nach der Caesar-Codierung zu verschlüsseln
- zu versuchen, die Codierung mit `crack_caesar` zu knacken
- und die mit dieser geratenen Zahl decodierte Zeichenkette auszugeben.

Sorgen Sie für verständliche Eingabeaufforderungen und geeignete Ausgaben bei Fehlern. Testen Sie ihr Programm mit Zeichenketten verschiedener Länge.

[3] `'E'` ist in der deutschen Sprache der häufigste Buchstabe.