Hochschule RheinMain
Fachbereich DCSM - Informatik
Marcus Thoss, M.Sc.
Prof. Dr. Robert Kaiser
Christian Neugebauer

Hardwarenahe Programmierung I

WS 2021/22
LV 1512

Abgabe 1

Zur Berücksichtigung der Leistung muss bis zum 03.12.2021 23:59 Uhr eine Zip-Datei mit dem Dateinamen *Nachname_Vorname*_hwp1_21_a01.zip per E-Mail an christian.neugebauer@hs-rm.de gesendet werden. In der Zip-Datei muss ein PDF-Dokument mit den Antworten und Ihr C-Projekt enthalten sein. Benennen Sie die PDF-Datei wie folgt, *Nachname_Vorname*_hwp1_21_a01.pdf. Im C-Projekt sind nur Quelltextdateien und das Makefile.

Aufgabe 1.1 (Fragen (10P.)):

a) Betrachten Sie den Quellcode **(1 P.)**

```
unsigned char a = 0x0e;
a <<= (1 | 2);
```

Welchen Wert hat a nach der Ausführung, dezimal und hexadezimal? Geben Sie Zwischenschritte der Auswertung des Ausdrucks in der zweiten Zeile an.

b) Wandeln Sie den folgenden Quellcode so um, dass statt einer do {} while(); eine while() {}-Schleife verwendet wird, aber insgesamt dieselbe Ausgabe erfolgt. **(1 P.)**

```
do {
   printf("abc");
} while(0);
```

c) Betrachten Sie den angegebenen Codeausschnitt und beantworten Sie:

1) Nur drei der Semikola führen dazu, dass der Quellcode *Syntaxfehler* enthält, so dass er nicht compilierbar ist. Finden Sie sie und streichen Sie sie. Geben Sie an, was Sie gestrichen haben.**(1 P.)**
2) Warum ist das Programm nun ausführbar, aber immer noch nicht fehlerfrei? **(1 P.)**
3) Geben Sie an, welches weitere Semikolon gestrichen werden muss, damit das Programm korrekt beendet wird. **(1 P.)**
4) Welches weitere Semikolon können Sie streichen, ohne die Durchführung der Schleifen und Rechnungen zu verändern? **(2 P.)**

```
int main(void);
{
  int a=1;
  ;
  while( a < 4);
    ++a;

  do ;
  {
   --a;
  };
  while (a > 2);

  return 0;
}
```

d) Welche Option sollten Sie *immer* beim Compilieren mit `gcc` angeben, um sich vor Unsauberkeiten und möglichen Fehlerquellen in Ihrem Code warnen zu lassen? Recherchieren Sie außerdem: Wie können Sie Warnungen zu Fehlern hochstufen lassen? Was geschieht bei einem Fehler im Unterschied zu einer Warnung? **(1 P.)**

e) Welchen `int`-Wert haben die folgenden Ausdrücke? Geben Sie Zwischenschritte an. **(1 P.)**

1) `(3<5 - 2)`
2) `(!0 == 2)`

f) Nehmen Sie `int n = 0;` als definiert an. Welchen Wert hat `n` nach der Ausführung folgender Anweisungen? **(1 P.)**

1) `n = 1 | 2;`
2) `(n = 1) || (n = 2);`

Aufgabe 1.2 (C-Projekt Implementierung ggT (15P.)):

Implementieren Sie den Algorithmus zur Bestimmung des größten gemeinsamen Teilers zweier Zahlen a und b nach Euklid in einer `ggt()`-Funktion mit den `int`-Parametern `a` und `b` und einem `main()`-Testprogramm.

- Algorithmus für $ggT(a, b)$ mit $a, b > 0$ nach Euklid (vgl. auch Übungsaufgaben zur LV Einführung i.d. Informatik):
  - falls $a = b$, dann ist $ggT(a, b) = a$.
  - falls $a < b$, dann wende den Algorithmus ggT an auf $(a, b - a)$.
  - falls $b < a$, dann wende den Algorithmus ggT an auf $(a - b, b)$.
- Eine alternative Formulierung des Algorithmus in Hinblick auf die Umsetzung als Funktion `ggt()` mit einer prozeduralen Programmiersprache wäre:
  - falls $a = b$, dann gib a zurück.
  - falls $a < b$, dann gib $ggT(a, b - a)$ zurück.
  - falls $b < a$, dann gib $ggT(a - b, b)$ zurück.

a) Legen Sie in einem neuen Verzeichnis die Quelldateien `ggt.c`, `ggt.h` und `ggttest.c` sowie ein `Makefile` an.

b) Versionieren Sie das Verzeichnis einschließlich eines neuen `readme.md` in Ihrem `gitlab`-Repository.

c) Das `Makefile` soll das Erstellen (Build) des Programms `ggttest` aus `ggt.c` und `ggttest.c` bei Eingabe von `make` (ohne Parameter) ermöglichen. Sehen Sie auch

   - einen Kommentarblock wie in den anderen Quelldateien (s.u.) und
   - ein `clean`-Target

   in Ihrem `Makefile` vor.

   Verwenden Sie beim Aufruf von `gcc` die Option `-Wall` und korrigieren Sie Ihren Quelltext, bis alle Warnungen beseitigt sind!

   Hinweis: Versionieren Sie Dateien zwischendurch immer wieder, spätestens dann, wenn alles korrekt ist, besser auch schon nach Teilerfolgen oder wenn neue Features hinzukommen. Geben Sie bei neuen Versionen immer durch Kommentare („Message“) an, was sich geändert hat (`git commit -m "..."`).

d) Entwerfen Sie ein Flussdiagramm für die `ggt()`-Funktion. Zeichnen Sie es zunächst wie für die bisherigen Aufgaben auf Papier oder mit einem Grafikprogramm Ihrer Wahl.

e) In `ggt.h` soll enthalten sein:

   - Ein Kommentarblock mit Dateiname, Autor und Inhaltsangabe.
   - Ein Schutz gegen verschachteltes `#include` mittels

```
#ifndef GGT_H
#define GGT_H
...
#endif
```

- Der Prototyp (Deklaration) einer `ggt()`-Funktion mit zwei `int`-Parametern `a` und `b`. Über dem Prototyp soll eine Kurzbeschreibung der Funktion mit ihren Parametern als Kommentar stehen.

f) In `ggt.c` soll enthalten sein:

- Ein Kommentarblock mit Dateiname, Autor und Inhaltsangabe.
- Ein `#include` von `ggt.h` und `#includes`, die Sie sonst noch benötigen.
- Die Implementierung der `ggt()`-Funktion. Beachten Sie: die Parameter-Variablen `a` und `b` stehen Ihnen im Funktionsrumpf als *lokale* Variablen zur Verfügung.

Hinweis: Laut Algorithmus ruft sich die `ggt()`-Funktion selbst auf. Diese Verschachtelung wird als *Rekursion* bezeichnet. Es handelt sich um eines der grundlegendsten Programmierkonzepte, das in manchen Fällen elegante Lösungen eines Problems erlaubt. Andererseits sollten Rekursionen vor allem hinsichtlich der Abbruchbedingung und des Zurückkehrens aus den tieferen Rekursionsschichten sorgfältig durchdacht und implementiert werden.

g) In `ggttest.c` soll enthalten sein:

- Ein Kommentarblock mit Dateiname, Autor und Inhaltsangabe.
- Ein `#include` von `ggt.h` und `#includes`, die Sie sonst noch benötigen.
- Die Implementierung einer `main()`-Funktion, die eine Textzeile Ihrer Wahl zur Begrüßung ausgibt, dann `ggt()` mit den Konstanten `A` und `B` aufruft und je nach Ergebnis von `ggt()` mit 0 oder -1 zurückkehrt.

Hinweis: Die zwei Zahlen, für die der ggT berechnet wird, sollen in dieser Aufgabe zunächst als Konstanten `A` und `B` per `#define` in `ggttest.c` mit Werten Ihrer Wahl festgelegt werden.

h) Zählen Sie in der *globalen* `int`-Variable `ggtCalls` die Anzahl der `ggt()`-Aufrufe, die nötig sind und geben Sie zum Schluss diese Anzahl zusammen mit dem gefundenen Teiler aus.

Deklarieren Sie hierzu `ggtCalls` in `ggt.h` mit

```
extern int ggtCalls;
```

Definieren Sie die Variable in `ggt.c` mit

```
int ggtCalls;
```

Prüfen Sie während des Ablaufs, ob mehr als `MAX_ITERATIONS` (offensichtlich eine weitere Konstante, die Sie definieren müssen) stattfinden, und beenden Sie das Programm in diesem Fall mit einer Fehlermeldung und negativem Rückgabewert.