Heinrich Heine Universität Düsseldorf

Hardwarenahe Programmierung - Übungsblatt 2

- Hinweise:
  - Zeichen in einem `char`-Array *ohne* Nullterminator sind noch kein String. Denken Sie daran, Ihre Strings zu terminieren!
  - Wenn Sie unerwartete Zeichen in Ihrer Ausgabe sehen oder die Ausgabe sich unterscheidet, obwohl Sie keinen Unterschied sehen, dann handelt es sich möglicherweise um (nicht darstellbare) Zeichen. Diese entstehen häufig, weil zufällige Bytes im Speicher als Text interpretiert wurden, da der Nullterminator fehlte.
  - Die Hinweise von Blatt 1 gelten auch noch auf diesem und zukünftigen Übungsblättern.

  Termine für Einzelgespräche:
  - Mittwoch, der 20. Oktober, von 12:30 Uhr bis 16:00 Uhr.
  - Donnerstag, der 21. Oktober, von 10:30 Uhr bis 14:00 Uhr.
  - Freitag, der 22. Oktober, von 8:30 Uhr bis 12:00 Uhr.

Vorbereitungsaufgabe 1 Beispiele zu Strings

Viele moderne Programmiersprachen speichern die Länge von Strings in einer Variable, zum Beispiel `.length` in Java. In C wird die Länge nicht gespeichert. Stattdessen wird das Ende eines Strings mit einem sogenannten "Nullterminator" markiert. Um die Länge des Strings zu finden müssen die Zeichen bis zu diesem Nullterminator gezählt werden. Dazu werden nacheinander Bytes im Speicher überprüft, bis eins gefunden wird, das den Wert 0 hat. Wenn der Nullterminator vergessen wurde, dann führt dies häufig zu Speicherzugriffsfehlern, da die Suche dann erst bei ungültigen Speicherzugriffen endet.

Im Ordner `string_examples` finden Sie einige Beispiele zur richtigen und falschen Verwendung von Strings.

Pflichtaufgabe 2 Strings

Graf Zahl besitzt eine Sammlung von Eingabe-Strings, die aus verschiedenen Teilen bestehen und mit einem Trenn-String getrennt sind. In folgendem Beispiel besteht ein String aus den vier Teilen `Pferd`, `Ente`, `Kuh` und `Gans`, getrennt mit dem String `","` (ein Komma).

```
Pferd,Ente,Kuh,Gans
```

Er möchte diese Teile nach folgendem Schema aufteilen:

- Teile mit einer Länge von 4 Zeichen sollen in einen Ausgabe-String `output` geschrieben werden, wieder getrennt mit dem gleichen trennenden String:

  ```
  Ente,Gans
  ```

  Sie dürfen annehmen, dass der Zeiger `output` auf einen ausreichend großen Speicherbereich zeigt.

- Teile der Länge 0 sollen verworfen und nicht mitgezählt werden.

- Die restlichen Teile sollen in das Array `not_length_4` (ein zweidimensionales `char`-Array) geschrieben werden. Sie dürfen annehmen, dass das Array gerade so groß genug ist, damit alle im Eingabestring enthaltenden Teile dieser Art hineinpassen.

  ```
  Pferd           // wird in not_length_4[0] geschrieben
  Kuh             // wird in not_length_4[1] geschrieben
  ```

1. Implementieren Sie in der Datei `string_split/string_split.c` die folgende Funktion:

   ```
   size_t string_split(
       char *output,
       char not_length_4[10][20],
       const char *input,
       const char *separator
   )
   ```

   - `output` ist der Ausgabestring für Teile der Länge 4.
   - `not_length_4` ist das Ausgabe-Array.
   - `input` ist die Eingabe. Sie dürfen annehmen, dass nur 7-Bit-ASCII-Zeichen vorkommen.
   - `separator` ist der trennende String. Sie dürfen annehmen, dass dieser String eine Länge von mindestens einem Zeichen hat.
   - Der Rückgabewert soll die Anzahl der Strings sein, die in `not_length_4` geschrieben wurden.

2. Wir haben Ihnen die Datei `Makefile` zur Verfügung gestellt. Der Befehl `make run` erstellt automatisch die Tests für diese Aufgabe und führt sie aus.

Pflichtaufgabe 3 Make

Mit Hilfe von Make kann man das Kompilieren von mehreren Dateien bequemer machen und nur Dateien neu kompilieren, welche sich geändert haben. Ihre Aufgabe ist es nun, zu dem von uns vorgegebenen Programm die Datei `Makefile` zu erstellen. Die Dateien finden Sie in dem Ordner `make` und den dazugehörigen Unterordnern `game`, `monster`, `player`, `sound` und `texture`.

- Erstellen Sie eine Makefile-Datei mit Regeln, die die C-Dateien `game.c`, `monster.c`, `player.c`, `sound.c`, `texture.c` und `main.c` *einzeln* zu Objekt-Dateien kompilieren. Weiterhin sollen die Dateien neu kompiliert werden, wenn sich eine Abhängigkeit ändert. Erstellen Sie hierzu passende Regeln. Die erzeugten Objekt-Dateien sollen die gleichen Namen wie die zugrundelegenden C-Dateien haben.

  Zum Beispiel würde die Datei `game.c` neu zur Objekt-Datei `game.o` kompiliert werden, wenn sich die Abhängigkeiten `game.c`, `game.h`, `monster.h`, `player.h`, `sound.h` oder `texture.h` ändern.
- Weiterhin soll es eine Regel `main` geben, die aus den Objekt-Dateien das Programm `main` erzeugt.
- Es soll eine Regel `all` geben, die von der Regel `main` abhängt und ansonsten leer ist. (wirklich!)
- Es soll eine Regel `clean` geben, die die erstellten Dateien löscht.
- Benutzen sie das Test-Skript `test.sh`, um Ihr Programm zu testen.

Hinweis: Lesen Sie hierfür in der Dokumentation von GNU Make die Kapitel 1-4 durch, um diese Aufgabe lösen zu können. Kapitel 5 "Mehrere Quelldateien: Zerlegen und zusammenbauen" aus dem Buch "C von Kopf bis Fuß" ist ebenfalls hilfreich.

Pflichtaufgabe 4 GDB

In dieser Aufgabe bekommen Sie von uns ein fertiges Programm geliefert, welches allerdings fehlerhaft ist, und sollen dieses mit dem **GNU debugger** (`gdb`) debuggen.

Hinweise:

- Im Makefile sind zwei verschiedene Regeln zum bauen des Programms. Der Befehl `make all` baut das Programm auf normale Weise. Wenn Sie nun versuchen das Programm mit `gdb` zu debuggen, werden Sie feststellen, dass dies nicht wirklich möglich ist. Dazu gibt es die Regel `make debug` welche das Programm mit speziellen Flags baut, sodass einige zusäztliche Informationen in die erzeugte Datei geschrieben werden.
- Der zu behebende Fehler ist in der Funktion in der das Programm abstürzt. Alle weiteren Funktionen brauchen Sie nicht zu beachten und sollen diese auch nicht verändern.
- Es müssen und sollen keine Zahlenwerte verändert werden.
- Der Fehler ist nur eine Kleinigkeit (welche wie so oft in C, aber schon zu einem Absturz führt). Sie müssen und dürfen nur ein Zeichen löschen, hinzufügen oder ändern um den Fehler zu beheben.
- Innerhalb des Programms ist es nicht möglich Ausgaben mit `printf` oder ähnlichem zu erzeugen, um die Wahrscheinlichkeit zu erhöhen, dass sie tatsächlich `gdb` zum debuggen verwenden.
- Die Befehle, welche sich auf dem **GDB cheatsheet** befinden, sind mehr als ausreichend, um den Fehler zu beheben. Vorallem die folgenden Befehle sind hier nützlich:
  - `run`
  - `backtrace`
  - `info`
  - `print`

  Sie sollten außerdem die Stepping-Befehle (`step, next, finish, continue`) und `watch` beherrschen.
- Das Programm ist erfolgreich korrigiert, wenn Sie alle Anforderungen hier beachtet haben und das Programm nicht mehr abstürzt.
- Mit dem Skript `./test.sh` können Sie ihre Abgabe testen.
- Wenn Sie zu viel geändert haben, können Sie mit `./test -restore` alle Dateien wieder auf ihren Ursprungszustand zurücksetzen.