Übungsserie 3

Programmierung mit C

Abgabedatum: 30.05.2021 - 18:00 Uhr

Damit für alle die gleichen Regeln gelten, müssen Ihre C-Programme in Standard-C (C11) mit dem GCC oder Clang kompilieren. Vermeiden Sie insbesondere plattformspezifische Includes, z.B. `windows.h`*. Schalten Sie alle Warnungen Ihres Compilers an, um hilfreiche Informationen zu erhalten und Fehler aufzuspüren. GCC und Clang unterstützen folgende Optionen, die zu verwenden sind:* `-Wall -Wextra -pedantic -std=c11 -Werror`*. Bitte beachten Sie eventuell benötigte Bibliotheken zum Linken, z.B.* `-lm` *für das Linken gegen die Mathebibliothek. Eine ausgezeichnete Referenz zu C finden Sie unter: http://en.cppreference.com/w/c.*

Bitte beachten Sie den genauen Abgabezeitpunkt in Moodle. Serien per E-Mail oder verspätete Abgaben können nicht gewertet werden. Ebenso wie Textaufgaben die nicht als PDF abgegeben wurden. Bitte laden Sie Ihre Abgabe in einem Zip-Archiv hoch sobald es mehr als eine Datei beinhaltet. Es werden ausschließlich eigenständig erarbeitete Lösungen bewertet. Identische Abgaben werden mit 0 Punkten bewertet.

Aufgabe 1 (27 Punkte) Entwerfen Sie ein C-Programm `findMaxZeroSequence.c` zur Berechnung der längsten Sequenz von Nullen in der Binärrepräsentation von einer positiv als auch negativen Ganzzahl `number`.

Die Genauigkeit der Binärrepräsentation soll als globale Konstante definert werden und entspricht 32 Bit. Die Ganzzahl `number` selber soll als Kommandozeilenparameter übergeben werden und daraufhin in den Datentyp `int64_t` konvertiert werden. Fangen Sie dabei mögliche Fehler ab und geben Sie eine Fehlernachricht aus. Berechnen Sie daraufhin die Binärrepräsentation und finden Sie die längste Sequenz von Nullen. Geben Sie im Anschluss die eingegebene Ganzzahl inklusive des Vorzeichens und deren Binärrepräsentation aus. Außerdem soll die Länge der längsten Sequenz als 2-stellige Zahl ausgegeben werden, wobei eine 1-stellige Zahl mit führender 0 aufgefüllt wird.

Stellen Sie für jede der im folgenden gegebenen Funktionsdeklarationen eine Implementierung mit der beschriebenen Funktionalität bereit und verwenden Sie diese.

1. `int convertDezToDualString(const int64_t number, char *stringDualNumber)`, diese Funktion soll eine Dezimalzahl `number` in eine Dualzahl umwandeln und in einem C-String abspeichern. Die Funktion soll EXIT_SUCCESS, falls die Konvertierung erfolgreich war, und EXIT_FAILURE zurückgeben, falls Fehler wie die Eingabe einer zu großen Zahl auftreten. Der verwendete C-String `stringDualNumber` soll dabei dynamisch in der `main`-Funktion erzeugt werden und entsprechend übergeben werden. Überprüfen Sie bei der dynamischen Speicherreservierung auch, ob die Erzeugung erfolgreich war und führen Sie eine entsprechende Fehlerbehandlung durch.

2. Die Funktion `void createComplementOnTwo(char *stringDualNumber)` soll das Zweierkomplement von einem gegebenen C-String bilden und auf dieser Variablen wieder zurückgeben.

3. `unsigned short countMaxZeroSequence(const char *stringDualNumber)` zählt für einen gegebenen C-String die längste Sequenz von Nullen und gibt diese als `unsigned short` zurück.

Alle möglichen Fehlernachrichten sollen dabei in die Standardfehlerausgabe geschrieben werden und alle anderen Nachrichten in die Standardausgabe.

Testen Sie das Programm sorgfältig und geben Sie mindestens 4 Testfälle und die dazugehörigen Ergebnisse im Quellcode als Kommentar mit an.

Hinweise:
Achten Sie darauf, das dynamisch reservierter Speicher wieder freigegeben werden muss. Bei der Berechnung der längsten Sequenz von Nullen sollen auch führende sowie schließende bzw. nachfolgende Nullen berücksichtigt werden. Eine Ganzzahl vom Typ `int64_t` besitzt in C eine Länge von 64 Bit und entspricht somit dem `long long` Datentyp. Den Datentyp `int64_t` finden Sie in der Bibliothek `stdint.h`. Eine Vorzeichen abhängige 32 Bit Ganzzahl hat einen Wertebereich von $-2.147.483.648$ bis $2.147.483.647$. Vergessen Sie bei dem C-String nicht die terminierende Null. Schreiben Sie ihren Programmcode sauber, einheitlich und leserlich. Verwenden Sie aussagekräftige Namen für die Variablen und kommentieren Sie ggf. den Code ausreichend.

Beispiel:
Input: 4239_{10} bzw. $0000\,0000\,0000\,0000\,0001\,0000\,1000\,1111_2$
Output: 19

Aufgabe 2 (33 Punkte) Ziel dieser Aufgabe ist das Programmieren eines Spiels, bei dem der Spieler einen Begriff raten muss. Der Begriff wird zunächst zufällig ausgewählt und alle Buchstaben werden durch `*` ersetzt.

Beispiel: Der zufällig gewählte Begriff ist `C-COMPILER`, dann ersetzt das Programm alle Buchstaben durch `*`. Dem Spieler würde also `*-********` angezeigt werden.

Der verdeckte Begriff wird dem Spieler nun angezeigt und ein zufälliger Punktebetrag zwischen 100 und 1000 in Hunderterschritten gewürfelt. Nun gibt der Spieler einen Buchstaben ein und erhält für jedes Auftreten des Buchstaben jeweils den zufällig bestimmten Punktebetrag. Sollte der Buchstabe nicht im gesuchten Begriff vorkommen, wird der gewürfelte Punktebetrag dem Spieler abgezogen.

Fortsetzung des Beispiels: Der zufällige Punktebetrag ist 300. Der Spieler wählt als Buchstabe `C`, dann erhält er 600 Punkte und das `C` im Begriff wird aufgedeckt. Es wird also `C-C*******` angezeigt. Als nächstes ist der zufällige Punktebetrag 500 und der Spieler wählt den Buchstaben `A`, dann werden dem Spieler 500 Punkte abgezogen, weil der Buchstabe `A` nicht im zu erratenden Begriff vorkommt.

Das Spiel wird beendet, wenn der Begriff erraten wurde. Sollte die Punktzahl größer dem kleinsten Betrag in der Highscore-Liste sein, dann wird ein Eintrag hinzugefügt.

Allgemeine Hinweise:

- Umlaute (ä,ü,ö) und ß sind in unserem Programm nicht zu berücksichtigen.

- Denken Sie daran einmal allokierten Speicher wieder freizugeben.
- Achten Sie auf die Trennung von Funktionsdeklarationen (`.h` - Header-Dateien) und Funktionsdefinitionen (`.c` - Source-Dateien).
- Beachten Sie, dass `scanf` Whitespaces (z.B. Enter-Taste) im Eingabebuffer abfängt, wenn Sie dem Typ-Spezifizierer ein Leerzeichen voranstellen, also `" %c"` als ersten Parameter für `scanf` benutzen (statt `"%c"`).
- Die Datei `Riddles.h` finden Sie unter https://moodle2.uni-leipzig.de/mod/resource/view.php?id=1479143.

a) (16 Punkte) Die Dateien `Game.c` und `Game.h` enthalten folgende Funktionen:

- `char *createRiddle( char const *const str )`:
  `createRiddle` ersetzt alle Buchstaben des übergebenen Begriffs durch `*` und gibt diesen C-String zurück. Sollte beim Speicher reservieren ein Fehler auftreten, wird eine Fehlermeldung ausgegeben und das Spiel mit der Funktion `exit` beendet.

- `int uncoverLetter( char *riddle, const char *solution, char letter )`:
  `uncoverLetter` deckt den übergebenen Buchstaben `letter` im verdeckten C-String `riddle` auf (`solution` enthält den gesuchten Begriff) und gibt die Anzahl der aufgedeckten Buchstaben zurück. Sollte ein Buchstabe bereits aufgedeckt sein oder im gesuchten Begriff nicht vorkommen, dann wird 0 zurückgegeben.

  Für das obige Beispiel, würde `uncoverLetter` mit C als erstem Buchstabentipp, also '`C-C*******` in `riddle` schreiben und 2 zurückgeben.

  Sie können davon ausgehen, dass der C-String `solution` nur Großbuchstaben enthält. Die Funktion `uncoverLetter` soll aber auch mit einem gegebenen Kleinbuchstaben in `letter` funktionieren.

- `int gamingLoop( const char *solution )`:
  `gamingLoop` führt das eigentliche Spiel durch, d.h. es lässt den Spieler solange Buchstaben raten, bis er den gesuchten Begriff komplett aufgedeckt hat. Nach vollständigem Aufdecken des Begriffs, endet die Funktion `gamingLoop` und gibt die Spielerpunktzahl zurück.

  Im Einzelnen macht sie Folgendes:
  - mittels `createRiddle` den verdeckten C-String des Begriffs erstellen
  - die eigentliche Spielschleife durchführen (bis der Begriff aufgedeckt wurde)
    - zu erspielende Punktzahl (zwischen 100 und 1000 in Hunderterschritten) würfeln und dem Spieler anzeigen
    - einen Buchstaben abfragen, mittels `uncoverLetter` aufdecken und aktuellen Rätselstand (z.B. `C-C*M*****`) anzeigen
    - Punkte wie oben beschrieben bestimmen und dem Spieler anzeigen
  - Punkte zurückgeben

- `void startGame()`:
  `startGame` wählt einen zufälligen Begriff aus der mitgegebenen `Riddles.h`, startet im Anschluss die Funktion `gamingLoop` und gibt die erspielte Punktzahl aus.

b) (13 Punkte) Die Dateien `Highscore.c` und `Highscore.h` sollen Folgendes enthalten:

- `Highscore.h` soll Folgendes `#define` und folgendes Struct enthalten:

```
#define HS_SIZE 10

typedef struct
{
    char name[20];
    int32_t points;
} HSEntry;
```

- `HSEntry *readHS( const char *filename )`:
  `readHS` liest die Highscore-Liste der Länge `HS_SIZE` aus der angegebenen Datei und gibt einen Zeiger darauf zurück. Sollte beim Speicher allokieren ein Fehler auftreten, dann soll eine Fehlermeldung ausgegeben und das Programm mit der Funktion `exit` beendet werden. Wenn ein Fehler beim Lesen der Datei auftritt (z.B. Datei existiert nicht), dann soll die Liste mit leeren Namen (`""`, Stringlänge 0) und 0 Punkten initialisiert werden.

- `void writeHS( const char *filename, const HSEntry *list )`:
  `writeHS` schreibt die übergebene Highscore-Liste in die angegebene Datei. Sollte das Öffnen fehlschlagen, gibt die Funktion eine Fehlermeldung aus und beendet sich.

- `bool isNewEntry( const HSEntry *list, int points )`:
  `isNewEntry` gibt `true` zurück, wenn der kleinste Eintrag in der Highscore-Liste weniger Punkte hat als der angegebene Parameter `points`. Sie können davon ausgehen, dass die Liste sortiert ist.

- `void addEntry( HSEntry *list, const char *name, int points )`:
  `addEntry` fügt die Kombination aus Name und Punkten in die Highscore-Liste ein (wenn Punktzahl `points` ausreicht). Dabei soll beachtet werden, dass der Parameter `name` bzgl. der Länge nicht limitiert ist. Nach Ausführen der Funktion, soll die Highscore-Liste korrekt sortiert sein.

  Hinweis: Es empfiehlt sich die Funktion `qsort`[1] zu nutzen.

- `void printHighscores( const HSEntry *list )`:
  `printHighscores` gibt die Highscore-Liste als Kombination aus Name und Punktzahl absteigend aus.

c) (3 Punkte) Erweitern Sie die Funktion `startGame` aus der `Game.c` so, dass bei einer ausreichend großen Punktzahl für einen Highscore-Eintrag, der Name des Spielers abgefragt und die Highscore-Liste entsprechend erweitert wird. Dazu muss die Highscore-Liste zunächst eingelesen und abschließend wieder geschrieben werden. Weiterhin soll die aktualisierte Highscore-Liste dem Spieler angezeigt werden.

d) (1 Punkt) In der Datei `main.c` soll das Spiel in der `main`-Funktion mit der Funktion `startGame` gestartet werden.

[1] Vgl. http://en.cppreference.com/w/c/algorithm/qsort