Dr. habil. Bernd Schürmann
Dr. Annette Bieniusa
pinc-support@cs.uni-kl.de

TU Kaiserslautern
Fachbereich Informatik

Übungsblatt 10: Programmieren in C (WS 2018/19)

Abgabe: Montag, 14.01.19, 12:00

1. Zur Beantwortung von Fragen und Hilfe bei Problemen stehen wir Ihnen immer in der Praktische Übung, Mittwoch 15:30, Raum 32-410-PC zur Verfügung sowie per Email an `pinc-support@cs.uni-kl.de`!
2. Sie können sich im Exclaim System unter `https://softech.cs.uni-kl.de/exclaim` mit Ihrem persönlichen STATS-Account einloggen und Dateien zu den einzelnen Übungen hochladen.
3. Die Aufgaben auf diesem Übungsblatt sind alle freiwillig zu bearbeiten.

Aufgabe 0: Vorlesungsumfrage

Der FB Informatik führt in jedem Semester eine Online-Vorlesungsumfrage durch. Wir würden uns sehr freuen, konstruktive Rückmeldung zur Vorlesung und den Übungen zu erhalten, denn Ihr Feedback hilft uns, unsere Veranstaltung weiter zu verbessern. Alle weiteren Informationen finden Sie hier:

`https://vlu.informatik.uni-kl.de/`

Aufgabe 1: Zufallszahlen

Abgabe: `zufall.c` in Exclaim

Die Bibliothek `stdlib.h` enthält einen Pseudo-Zufallszahlengenerator, mit Hilfe dessen man zufällig Integer zwischen 0 und `RAND_MAX` (Konstante, definiert in `stdlib.h`) erzeugen kann.

Der Pseudo-Zufallszahlengenerator muss zunächst (einmal!) über `srand(unsigned integer seed)` initialisiert. Verwendet man für verschiedene Programmaufrufe den gleichen Wert für den Parameter `seed`, wird die gleiche Folge von Zufallszahlen erzeugt.[1]

Durch jeden folgenden Aufruf von `int rand (void)` wird dann eine neue Pseudo-Zufallszahl erzeugt.

a) Schreiben Sie eine Funktion `int rand_interval(int von, int bis)`, die eine Zufallszahl in einem gegebenen Intervall `[von,bis]` erzeugt.

b) Für ein Würfelspiel wollen Sie einen Würfel simulieren und dazu Zahlen zwischen 1 und 6 zufällig erzeugen. Damit das Spiel fair ist, sollten alle Zahlen mit gleicher Wahrscheinlichkeit erzeugt werden. Um zu testen, wie gut der Generator ist, simulieren Sie 100.000.000 Würfelereignisse und zählen, wie häufig die einzelnen Zahlen gewürfelt werden.

c) Für die Abgabe in Exclaim verwenden Sie als Seed für den Generator den Wert `123456` und erzeugen folgende Ausgabe:

```
1 : 1667340
2 : 1667833
3 : 1663587
4 : 1667633
5 : 1667099
6 : 1666508
```

[1] Die Verwendung von gleichen Zufallssequenzen ist hilfreich, um reproduzierbare Tests zu schreiben. Will man tatsächlich zufällige Sequenzen, kann beispielsweise die aktuelle Zeit als Initialisierungswert verwenden werden: `srand(time(NULL))` verwendet die Anzahl der Sekunden seit Januar 1, 1970 als Initialisierungswert.

Aufgabe 2: Monte-Carlo-Methode

Abgabe: `pi.c` in Exclaim

Zufallszahlen können auch dazu verwendet werden, um die Kreiszahl π mit der Monte-Carlo-Methode approximativ zu berechnen. Dazu werden zwei Zahlen $x, y \in [0, 1)$ gewürfelt. Das Zahlenpaar entspricht dabei einer Koordinate im Einheitsquadrat. Für jedes Zahlenpaar ist nun zu testen, ob sich die Koordinate innerhalb oder außerhalb des 1. Quadranten des Einheitskreises befindet. Das Verhältnis der Punkte im Einheitskreis zur Gesamtzahl der Punkte entspricht $\pi/4$ (wenn man beliebig viele Punkte mit einem perfekten Zufahllszahlengenerator erzeugt).

Laden Sie von der Vorlesungshomepage die Vorlage `pi.c` herunter. Verwenden Sie die Funktion `double drand(void)`, um ein Programm zu implementieren, das π mit Hilfe der Monte-Carlo Methode approximiert. Wie verbessert sich das Ergebnis, wenn 100, 1.000, 1.000.000, 1.000.000.000 zufällig erzeugte Punkte verwendet werden?

Aufgabe 3: Ellipsen

Abgabe: `ellipse.c` in Exclaim

Schreiben Sie eine Funktion, die die Summe einer unbestimmten Zahl von Integer-Werten berechnet.

a) Laden Sie die Vorlage `ellipse.c` von der Vorlesungshomepage herunter und machen sich mit der Programmstruktur vertraut.

b) Implementieren Sie folgenden zwei Versionen:

- Das Ende der Parameter wird durch den Wert `0` markiert (`summe1`).
- Die Anzahl der Parameter, die aufsummiert werden sollen, wird im ersten Parameter übergeben (`summe2`).

Aufgabe 4: Hangman!

Abgabe: `hangman.c` in Exclaim

In dieser Aufgabe implementieren Sie das bekannte Wortratespiel “Hangmann” als C-Programm. Zeigen Sie zunächst von einem zu ratenden Wort für die einzelnen Zeichen jeweils nur Striche an. Anschließend fragen Sie nach Buchstaben. Sollten die Buchstaben im zu ratenden Wort vorkommen, so werden die entsprechenden Positionen aufgedeckt, d.h. an dieser Stelle werden die tatsächlichen Zeichen angezeigt.

Ihr Programm soll folgende globale Variablen und Funktionen enthalten:

- `char picture[][];`
  Das Array repräsentiert die Zeichenfläche für Ihren Galgen. Bei einem nicht geratenem Wort soll der Galgen auf einem 12x20-Feld folgendermaßen aussehen:

```
xxxxxxxxxxxxxxx
x   x         x
x  x          -
x x         (   )
xx         \  -  /
x           \ | /
x             |
x             |
x            / \
x           /   \
x
xxxxxxxxxxxxxxxxxxx
```

- `char * terms[N] = {"Hangman", "ProgrammierenInC", ...};`

Array mit N Begriffen. Ergänzen Sie die Liste aus der Vorlage mit weiteren Begriffen.

- `char searchString[20];`
  Zeichenkette, die die bisher geratenen Buchstaben enthält.
- `int searchTermIndex, errorCount;`
  Index des zu suchenden Begriffs in `terms` und Anzahl bisher falsch geratener Buchstaben.
- `int init (void);`
  Bestimmt per Zufallszahlengenerator einen Begriff aus der Liste `term`. Die Variable `searchString` wird mit so vielen Strichen '_' belegt, wie der Suchbegriff Buchstaben enthält.
- `void print (void);`
  Gibt den bisher erstellten Galgen (`picture`) auf dem Bildschirm aus.
- `int updateSearchString (char ch);`
  Überprüft, ob der Buchstabe `ch` im Suchwort enthalten ist. Falls ja, werden alle Vorkommen von `ch` in `searchString` eingesetzt; andernfalls soll der Galgen erweitert und ausgegeben werden (siehe Funktion `updateGallow`) und `errorCount` inkrementiert werden. Der Rückgabewert der Funktion soll die Anzahl noch nicht geratener Buchstabenpositionen sein.
- `void updateGallow (int errorNo);`
  Belegt die Variable `picture` mit einem Teil des Galgenbildes, der der Fehlernummer entspricht (z.B. einem Arm). Zur Vereinfachung ist diese Funktion in der Vorlage vorgegeben.
- `int main (void);`
  Die main-Funktion besteht im Kern aus einer do-while-Schleife, wobei eine Iteration der Schleife einer Runde im Spiel entspricht. Zu Beginn jeder Runde wird der aktuelle Suchbegriff mit den Platzhaltern angezeigt. Dann wird der nächste Buchstabe vom Benutzter abgefragt. Ist dieser ein Kleinbuchstabe, kann der Suchbegriff geprüft werden und der bisher erstellte Galgen sowie das bisher geratene Wort ausgegeben werden. Alle anderen Zeichen werden überlesen bzw. ignoriert. Die Schleife bricht ab, wenn der gesuchte Begriff geraten oder der Galgen fertig erstellt wurde. Dann soll ein entsprechender Abschlusstext ausgegeben werden (`"Gewonnen!"` bzw. `"Verloren!"`). Am Ende soll der geratene Begriff ausgegeben werden (z.B. `"Der gesuchte Begriff war 'BerndIstDerBeste'"`.)

Hinweise:

- Den Zufallszahlengenerator können Sie aus Aufgabe 1 übernehmen.
- Es sollen nur Kleinbuchstaben eingegeben werden. Bei der Überprüfung, ob der Buchstabe enthalten ist, sollen Großbuchstaben in Kleinbuchstaben gewandelt werden. Hierfür können Sie die Funktion `char tolower(char c)` aus der Bibliothek `ctype.h` verwenden oder zur Übung selbst schreiben. Die Funktion `tolower` wandelt Großbuchstaben in Kleinbuchstaben und lässt Kleinbuchstaben unverändert.
- Um Ihr Programm auf Exclaim zu testen, setzen Sie die Präpozessorkonstante `TEST` auf 1; dann wird immer der Suchbegriff `"BerndIstDerBeste"` ausgewählt.

Beispiel für die Ausgabe während einer Spielrunde:

```
Gesuchter Begriff:  Be__dI__De_Be__e
Naechster Buchstabe (nur Kleinbuchstaben, keine Umlaute):  n
  xxxxxxxxxxxxxx
  x   x
  x  x
  x x
  xx        \     /
  x          \ | /
  x            |
  x            |
  x           / \
  x          /   \
  x
xxxxxxxxxxxxxxxxxxx
Gesuchter Begriff:  Be_ndI__De_Be__e
Naechster Buchstabe (nur Kleinbuchstaben, keine Umlaute):  o
Der Buchstabe 'o' ist nicht enthalten.
  xxxxxxxxxxxxxx
  x   x
  x  x         -
  x x         (   )
  xx        \  -  /
  x          \ | /
  x            |
  x            |
  x           / \
  x          /   \
  x
xxxxxxxxxxxxxxxxxxx
```