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

TU Kaiserslautern
Fachbereich Informatik

Übungsblatt 12: Programmieren in C (WS 2018/19)

Abgabe: Montag, 28.01.19, 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 1: Wege aus dem Labyrinth

In dieser Aufgabe ist ein modulares Programm zur Suche eines Wegs durch ein Labyrinth zu schreiben. Zur Suche des Wegs durch das Labyrinth wird ein rekursiver Backtracking-Ansatz vorgeschlagen, wie er in der Vorlesung angedeutet wurde (analog zum Springer-Problem aus der Vorlesung). Es darf aber auch ein eigener Lösungsansatz entwickelt und in der Funktion `search_exit` realisiert werden.

Die zentrale Datenstruktur ist ein globales zweidimensionales Array von Zeichen. Mit '#' werden die Labyrinth-Wände modelliert, 'S' ist das Startfeld und 'Z' das Zielfeld.

Das Programm soll aus folgenden Modulen bestehen:

- `main.h`
  Diese header-Datei wird in alle .c-Dateien eingebunden. Sie enthält eine Konstante `MAX_SIZE` (maximale Kantenlänge des Labyrinths, hier mit dem Wert 100), globale Datentypen (u.a. eine Struktur `point` mit den Integer-Koordinaten `x` und `y`) und die extern-Deklaration des Labyrinths `char maze [MAX_SIZE][MAX_SIZE]`. Benötigen Sie weitere globale Konstanten, Datentypen und Variablen, so sind sie dieser Datei hinzuzufügen.
- `init.h` und `init.c`
  In diesem Modul wird die globale Variable `maze` mit einem Labyrinth (s.o.) initialisiert. Eine Funktion `struct point init_maze (void);` liest dabei zunächst die Höhe und Breite des Labyrinths als zwei `int`-Werte ein, dannach wird zeilenweise das Labyrinth mittels `fgets` eingelesen und `maze` entsprechend initialisiert. Die Funktion gibt dann die Größe des verwendeten Labyrinths als Punkt (= Höhe und Breite) zurück.
- `in_out.h` und `in_out.c`
  Enthält eine Funktion `void print_maze (struct point size);`, die das Labyrinth `maze` der Größe `size` auf dem Bildschirm ausgibt.
- `search.h` und `search.c`
  Dieses Modul umfasst zwei Funktionen. Die Funktion `struct point search_entrance (struct point size);` sucht im Labyrinth der Größe `size` das Startfeld 'S' und gibt dessen Koordinate als Punkt zurück. Die Funktion `enum bool search_exit (struct point pos);` ist der Kern des Programms. Diese Funktion sucht ausgehend vom Feld `pos` einen Weg zum Zielfeld 'Z'. Der Weg zum Zielfeld soll in der Variablen `maze` durch die Zeichen `^`, `v`, `<`, `>` beschrieben werden. Wurde ein Weg zum Ziel gefunden, gibt die Funktion `true`, ansonsten `false` zurück.
- `main.c`
  Die `main`-Funktion ruft alle o.g. Funktionen in sinnvoller Reihenfolge auf.

Mögliche Lösungsstrategie (mittels Rekursion und Bachtracking): Man prüft zunächst, ob die aktuelle Position sich auf dem Zielfeld 'Z' befindet; in diesem Fall Rekursionsabbruch und Rückgabe von `true`. Ist das nicht der Fall, wird versucht, nach oben zu gehen. Ist das Feld

$[y-1, x]$ frei oder das Zielfeld, wird in das aktuelle Feld '^' geschrieben und rekursiv die Suche an der Position $[y-1, x]$ fortgesetzt (Achtung: $y-1$ darf nicht negativ sein!). Ist der Weg nach oben nicht möglich, wird alternativ nach rechts, dann nach unten und zuletzt nach links versucht zu gehen. Ist in allen 4 Richtungen kein Weiterkommen möglich, muss das aktuelle Feld wieder als unbesetzt markiert und `false` zurückgegeben werden.

Beispiel: Eingabe

```
20 25
#########################
S       # # #           #
# ##### # # # ###########
# #     # # # #  ## #   #
# ####### # # #       # #
# #   # # # # ######### #
#   #     #             #
# ##### # # # ###########
# #     # # #           #
# ####### # ####### # # #
#    #    # # #   # # # #
###### #### # #   # # # #
#           # #   # # # #
# ########### ### # # # #
#           #     # # # #
################## # # # #
#                  # # # #
# ################## # ###
#                    #   Z
#########################
```

Ausgabe: `Weg zum Ziel gefunden.`

Bei Verwendung des oben beschriebenen Lösungsstrategie hätte das Labyrinth bei Programmende folgende Struktur:

```
#########################
>v      # # #           #
#v##### # # # ###########
#v#     # # # #  ## #   #
#v####### # # #       # #
#v#>>v# # # # ######### #
#>>^#>>>>v#>>v          #
# ##### #v#^#v###########
# #     #v#^#>>>>>>>>v  #
# #######v#^####### #v# #
#    #v<<<#^# #   # #v# #
######v####^# #   # #v# #
#     >>>>>^# #   # #v# #
# ########### ### # #v# #
#           #     # #v# #
################## # #v# #
#                  # #v# #
# ################## #v###
#                    #>>>Z
#########################
```

Texteditor (Teil 2): Dynamische Daten

Bisher war unsere zentrale Datenstruktur im Texteditor ein zweidimensionales Array von Zeichen. Jede Zeile fasste dabei `LINE_LENGTH` Zeichen. Dies soll nun optimiert werden. Bei der

Eingabe eines Textes soll mit der Systemfunktion `malloc` Speicher für diesen Text angelegt werden. Die zentrale Datenstruktur reduziert sich dadurch auf ein eindimensionales Array von Pointern auf solche dynamisch angelegten Texte.

a) Ändern Sie die zentrale Datenstruktur `text_field` in ein eindimensionales Array, das Pointer auf Zeichenketten speichern kann. Ändern Sie Ihr Programm, sodass es mit dieser neuen Datenstruktur korrekt übersetzt wird.

b) Fügen Sie dem Modul `sub_functions` eine neue Funktion `char * new_line (char text[])` hinzu, die für `text` passend Speicher allokiert und einen Pointer auf diesen Speicher zurückgibt. Diese muss in der Editor-Funktion `insert` verwendet werden, um den neu angelegten Text der zentralen Datenstruktur hinzuzufügen.

c) Geben Sie in der Funktion `delete_line` den nicht mehr benötigten Speicher mit Hilfe der Systemfunktion `free` wieder frei.

d) Ändern Sie außerdem die Funktion `search_in_line`. Hierbei sollen die ursprünglichen Array-Zugriffe durch Pointer-Operationen ersetzt werden.

Hinweis

- Bitte laden Sie in Exclaim wieder alle Dateien des Editors hoch wie bei der letzen Aufgabe des vorigen Übungsblattes.