Institut für Informatik
Prof. Dr. Stefan Conrad

Sommersemester 2021
Übung 7

Hardwarenahe Programmierung
Gruppe 09 (Florian)

Bei diesem Abschlussprojekt bearbeiten Sie eine umfangreichere C-Programmieraufgabe, in der Sie Ihre gesammelten Fähigkeiten anwenden sollen. Die Aufgabe ist erstmals nicht kleinschrittig für Sie vorbereitet. Fertigen Sie zuerst einen grobe Skizze in Form von Kommentaren oder Funktionsaufrufen an. Ersetzen Sie die Kommentare nach und nach durch funktionsfähigen Code.

Wichtig:

- *Die Abgabe ist diesmal nicht um 23:55 Uhr, sondern um 8:00 Uhr.*
- *Sie müssen das Abschlussprojekt in einem Einzelgespräch erklären und dazu Fragen beantworten. Testen Sie **vorher** Ihre technische Ausstattung (Webcam/Mikrofon oder Smartphone) und halten Sie einen **Lichtbildausweis** bereit.*
- *Tests können nur beweisen, dass ein Programm Fehler hat. Das Gegenteil zu zeigen, also das ein Programm immer fehlerfrei arbeitet, ist nahezu unmöglich. Verlassen Sie sich daher nicht zu sehr auf die vorgegebenen Tests, sondern testen Sie selbstständig die Sonderfälle in Ihrer Implementierung.*

1 Abschlussaufgabe – Dungeon

In dieser Aufgabe geht es darum, vor Monstern zu fliehen und aus einem Dungeon zu entkommen.

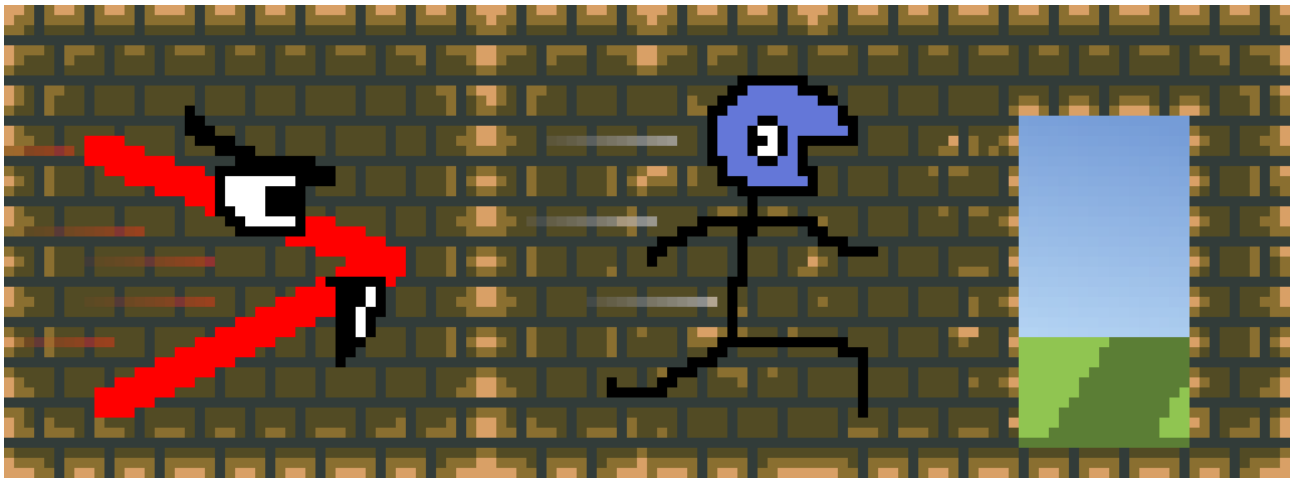

Abbildung 1: Dramatisierte Darstellung der Flucht aus dem Verlies.

2 Aufgabenstellung:

- Ihr Programm soll eine Datei entgegennehmen, in der ein Level des Dungeon gespeichert ist. Ein Level besteht aus:
  (a) Mindestens einem Ausgang, dargestellt durch das Zeichen „`A`“.
  (b) Genau einer Spielfigur, dargestellt durch das Zeichen „`S`“.
  (c) Wänden, dargestellt durch das Zeichen „`#`“.
  (d) Leeren Feldern, dargestellt durch Leerzeichen.
  (e) Einer Anzahl an Monstern, dargestellt durch eines der Zeichen „`<>v^`“.

```
########
#  S   #
#    < #
#  A   #
########
```

Abbildung 2: Beispiel mit Spielfigur „`S`“, Ausgang „`A`“und einem Monster „`<`“, das nach Links schaut.

- Weiterhin soll das Programm Steuer-Befehle über die Kommandozeile oder eine Datei entgegennehmen können.
- Nachdem ein Befehl verarbeitet wurde, soll das Level auf der Standardausgabe oder in eine Datei ausgegeben werden.
- Zusätzlich soll davor die Anzahl der bisher verarbeiteten Befehle und der zuletzt verarbeitete Befehl ausgegeben werden.
- Zu Beginn des Levels soll das Spielfeld ebenfalls einmal ausgegeben werden.

Die Höhe und Breite der Level ist unbeschränkt. Achten Sie darauf, keinen Stacküberlauf auszulösen.

2.1 Spielfigur

Die Spielfigur (`S`) wird durch die folgenden Befehle gesteuert:

- „`w`“: Ein Feld nach Oben gehen.
- „`a`“: Ein Feld nach Links gehen.
- „`s`“: Ein Feld nach Unten gehen.
- „`d`“: Ein Feld nach Rechts gehen.

Nach der Eingabe eines Zeilenumbruchs sollen die Befehle verarbeitet werden. Die Eingabe von `EOF` beendet das Spiel. Andere Eingaben als `wasd`, Zeilenumbruch und `EOF` sollen ignoriert werden.

Spielfiguren können auf allen Nicht-Wand-Feldern laufen.

2.2 Wände

Wände (`#`) blockieren ein Spielfeld, sodass weder Spielfigur noch Monster an dieser Stelle laufen können.

Sie dürfen annehmen, dass alle Level sind so gestaltet sind, dass eine Spielfigur den von Mauern umgrenzten Spielbereich nicht verlassen kann.

2.3 Monster

Monster können in eine von vier Richtungen schauen:

- „<“: Links.
- „>“: Rechts.
- „^“: Oben.
- „v“: Unten.

Nach jedem verarbeiteten Befehl laufen Monster ein Feld in Blickrichtung weiter. Wenn ein Monster in eine Wand oder einen Ausgang laufen würde, dann dreht es sich stattdessen um 180 Grad:

```
>  # // Schritt 1
 > # // Schritt 2
  ># // Schritt 3
  <# // Schritt 4
 < # // Schritt 5
<  # // Schritt 6
```

Abbildung 3: Ein Monster läuft gegen eine Wand und ändert die Richtung.

Wenn mehrere Monster auf dem selben Feld stehen, dann soll nur das Monster angezeigt werden, dass zuerst in der Level-Datei vorkam.

Wenn ein Monster und eine Spielfigur nach der Verarbeitung eines Befehls auf dem gleichen Feld stehen, dann soll nur das Monster angezeigt werden. Weiterhin soll die Nachricht *„Du wurdest von einem Monster gefressen.“* ausgegeben und das Spiel beendet werden.

2.4 Ausgänge

Nachdem eine Spielfigur einen Ausgang betritt, soll sich das Programm beenden und die Nachricht *„Gewonnen!“* ausgeben. Die Spielfigur wird in diesem Fall nicht mehr angezeigt, da sie den Ausgang bereits in die Freiheit durchschritten hat.

2.5 Beispiele

```
##### // Level zu Beginn.
#S A#
#####
1 d   // Erster Befehl: Ein Feld nach Rechts.
##### // Level nach dem erstem Befehl.
# SA#
#####
2 d   // Zweiter Befehl: Noch ein Feld nach Rechts.
##### // Level nach dem zweitem Befehl.
#  A#
#####
Gewonnen! // Das Programm beendet sich.
```

Abbildung 4: Ausgabe für ein einfaches Level. Nach zwei Schritten ist der Ausgang erreicht.

```
###### // Level zu Beginn.
#S <A#
######
1 a   // Erster Befehl: Ein Schritt nach Links in die Wand.
###### // Level nach dem erstem Befehl.
#S< A#
######
2 w   // Zweiter Befehl: Ein Schritt nach Oben in die Wand.
###### // Level nach zweitem Befehl.
#<  A#
######
Du wurdest von einem Monster gefressen. // :(
```

Abbildung 5: Die Spielfigur läuft mehrmals in Wände und wird von einem Monster gefressen.

3 Rückgabewerte

Das Programm soll im Fehlerfall eine kurze, aussagekräftige Meldung auf der Standardfehlerausgabe ausgeben und folgende Rückgabewerte liefern:

- 0, wenn alles ordnungsgemäß funktioniert.
- 1, wenn eine Datei nicht geöffnet werden konnte.
- 2, wenn eine Datei nicht gelesen werden konnte.
- 3, wenn eine ungültige, zu viele oder doppelte Optionen an das Programm übergeben wurden.

Falls gleichzeitig mehrere Fehler auftreten sollten darf Ihr Programm sich einen davon aussuchen, diesen wie oben behandeln und den entsprechenden Rückgabewert liefern.

4 Aufruf des Programms:

- Das Programm kriegt beim Aufruf optional den Namen der Level-Datei übergeben. Falls keine Datei übergeben wurde soll das Level in der Datei `level/1.txt` geöffnet werden.
- Die Eingabe-Datei mit Befehlen wird mit `-i` spezifiziert. Falls keine Eingabedatei übergeben wurde, wird die Standardeingabe verwendet.
- Die Ausgabe-Datei mit Befehlen wird mit `-o` spezifiziert. Falls keine Ausgabedatei übergeben wurde, wird die Standardausgabe verwendet.

Einige Beispiel-Aufrufe:

```
./dungeon
./dungeon -i eingabe.txt
./dungeon -o ausgabe.txt
./dungeon -i eingabe.txt -o ausgabe.txt level.txt
./dungeon level.txt -i eingabe.txt -o ausgabe.txt
```

5 Testen des Programms

Wir stellen Ihnen eine Makefile-Datei zur Verfügung. Mit dem Befehl `make run` können Sie ihr Programm kompilieren und testen. Dadurch werden mehrere Unterregeln ausgeführt:

- `make compile`: Kompiliert das Programm.
- `make compile_fsanitize`: Kompiliert das Programm mit den zusätzlichen Compiler-Optionen `-fsanitize=address` und `-fsanitize=undefined`.
- `make valgrind`: Überprüft die Funktionsweise des Programms mit `valgrind`.
- `make fsanitize`: Überprüft die Funktionsweise des Programms mit `fsanitize`.
- `make aufrufe`: Überprüft, ob das Programm Kommandozeilenparameter richtig behandelt.
- `make stack`: Überprüft, ob das Programm bei reduzierter Stackgröße immer noch läuft. Falls dieser Test fehlschlägt haben Sie wahrscheinlich ein zu großes Array auf dem Stack angelegt oder möglicherweise den Stack mit rekursiven Funktionsaufrufen gesprengt.
- `make clean`: Räumt auf und löscht alle Ausgabe-Dateien, da diese nicht hochgeladen werden müssen.

Nachfolgende Leerzeichen („trailing whitespace“) werden beim Vergleich der Ausgabe ignoriert.

6 Wichtige Anforderungen:

- Der Speicher ihrer Datenstrukturen muss mit `malloc` und `free` dynamisch verwaltet werden. Insbesondere darf es keine feste Obergrenze für die Größe der Level-Datei oder für die Anzahl an Eingaben geben, ab der Ihr Programm plötzlich nicht mehr funktioniert. Dies bedeutet auch, dass Sie das Spielfeld nicht auf dem Stack anlegen dürfen, denn der Stack hat eine begrenze Größe. Die Funktionen `calloc` und `realloc` dürfen auch verwendet werden.
- Vor Beendigung des Programms muss der gesamte Speicher wieder freigegeben werden und geöffnete Dateien müssen geschlossen werden.
- Valgrind darf weder Speicherzugriffsverletzungen noch Speicherlecks oder sonstige Fehler melden.
- Funktionen sowie Datenstrukturen müssen in einer separaten Header-Datei deklariert werden.
- Das Programm muss alle Tests fehlerfrei bestehen, d.h. `make run` darf keine Fehler liefern.
- Testen Sie Ihr Programm selbst, um eventuelle Sonderfälle Ihrer Implementierung zu überprüfen.
- Ein Testdurchlauf (`make run`) darf nicht länger als fünf Minuten dauern und nicht mehr als 4 Gigabyte RAM belegen. Selbst mittelmäßige Implementierungen sollten allerdings deutlich unter dieser Grenze liegen.
- Achten Sie auf die Einhaltung der Coderichtlinien im Lernmodul. Insbesondere müssen Sie Ihren Code in Funktionen mit einer Länge von jeweils höchstens 60 Zeilen gliedern und globale Variablen dürfen nicht verwendet werden.
- Zum Bestehen müssen Sie Ihren Code in einem Einzelgespräch erklären und Fragen beantworten. Vereinbaren Sie dazu mit Ihrer/Ihrem Tutor:in einen Termin.
- Sagen Sie ihren Termin bitte rechtzeitig ab, wenn absehbar ist, dass die Tests zum Ende der Abgabefrist nicht erfolgreich durchlaufen werden.

Viel Erfolg!