Institut für Informatik
Prof. Dr. Stefan Conrad

Sommersemester 2021
Übung 7

**Hardwarenahe Programmierung**
Gruppe 17 (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 – Jump & Run

In dieser Aufgabe geht es darum, ein Jump- & Run-Spiel („platformer“) zu implementieren.

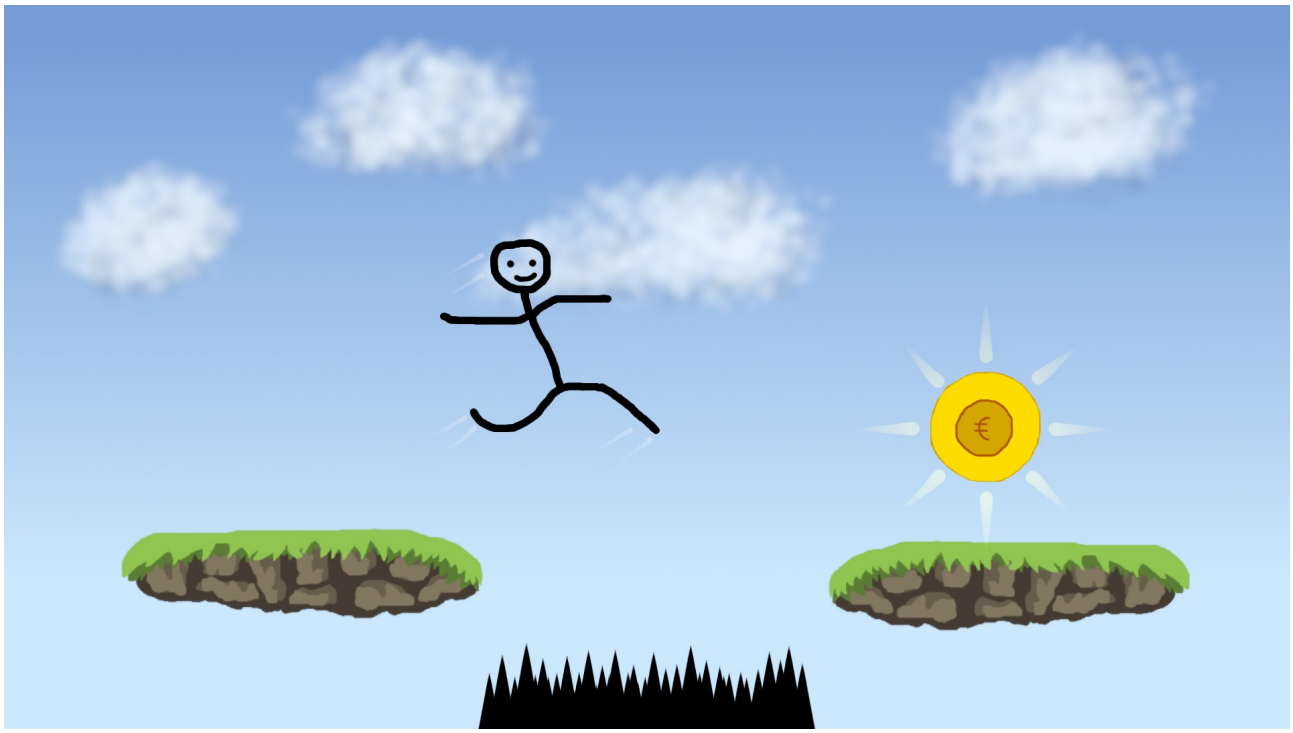

Abbildung 1: Eine Spielfigur springt über einen Abgrund, um eine Goldmünze einzusammeln.

# 2 Aufgabenstellung:

- Ihr Programm soll eine Datei entgegennehmen, in der ein Level gespeichert ist. Ein Level hat eine Breite von bis zu 80 Zeichen zuzüglich eines Zeilenumbruchs, eine unbegrenzte Höhe und besteht aus:
  (a) Stabilen Plattformen, dargestellt durch das Zeichen „`#`“.
  (b) Genau einer Spielfigur, dargestellt durch das Zeichen „`S`“.
  (c) Wertvollen Goldmünzen, dargestellt durch das Zeichen „`G`“.
  (d) Tödlichen Stacheln, dargestellt durch eines der Zeichen „`<>v^`“.
  (e) Mindestens einem Ziel „`Z`“.
  (f) Ästhetisch ansprechenden Wolken, die das Spielgeschehen nicht beeinflussen.
  (g) Leerzeichen, um alles an den rechten Platz zu rücken.

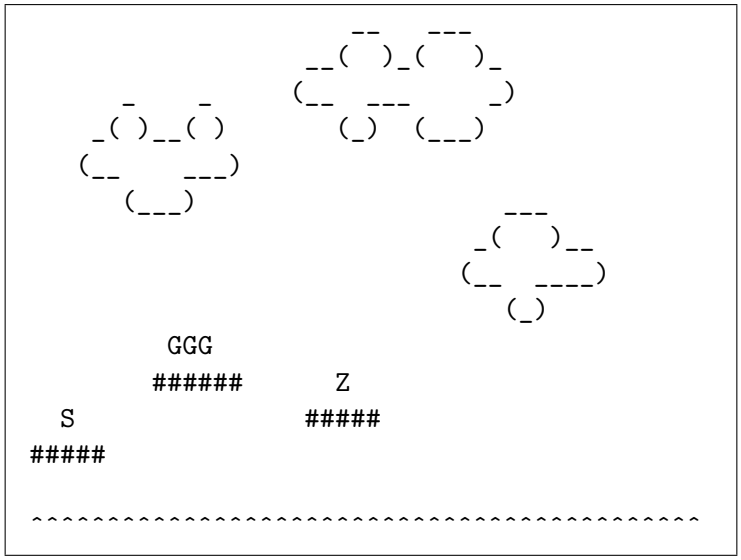


Abbildung 2: Ein Level mit Spielfigur, drei Plattformen, Goldmünzen, Wolken, Stachelboden und Ziel.

- Weiterhin soll das Programm Steuer-Befehle über die Kommandozeile oder eine Datei entgegennehmen können.
- Zu Beginn des Spiels und nachdem ein Befehl verarbeitet wurde, soll das Level auf der Standardausgabe oder in eine Datei ausgegeben werden.
- Davor soll die Anzahl der bisher seit Spielbeginn verarbeiteten Befehle ausgegeben werden.

## 2.1 Steuerung der Spielfigur

- Die Spielfigur soll bei der Eingabe eines Zeilenumbruchs ein Feld nach Rechts gehen, falls dort keine Plattform ist.
- Zusätzlich soll die Spielfigur ein Feld nach unten bewegt werden (fallen), falls sie nicht auf festem Boden steht.
- Analog zum Fallen soll die Spielfigur nach Oben bewegt werden, während sie springt.
- Die Spielfigur springt, falls Leerzeichen und ein Zeilenumbruch eingegeben wurden.
- Die Eingabe von `EOF` beendet das Spiel.
- Andere Zeichen sollen ignoriert werden und haben keinen Einfluss auf das Spiel.

## 2.2 Sprünge

- Die Spielfigur kann 3 Felder hoch springen.

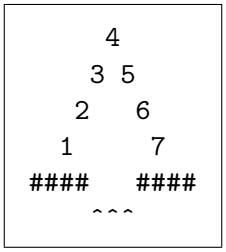


Abbildung 3: Die Flugbahn einer Spielfigur, die in 7 Spielschritten über einen Abgrund springt.

- Die Level sind so gestaltet, dass die Spielfigur nicht rausfallen oder oben rausspringen kann.
- Die Spielfigur kann nicht nochmal springen, falls sie sich in der Luft befindet.

## 2.3 Spielende

- Das Level wird beendet, sobald die Spielfigur das Ziel erreicht. Es wird die Nachricht *„Gewonnen! Punktzahl: “* gefolgt von der Anzahl der gesammelten Goldmünzen ausgeben.
- Das Level wird wiederholt, falls die Spielfigur von Stacheln getroffen wurde. Die ausgegebene Nachricht in diesem Fall ist *„Leider verloren. Das Level wird neu gestartet.“*.

## 2.4 Sonderfälle

- Falls die Spielfigur auf einem Feld mit Wolken oder Stacheln stehen sollte wird nur die Spielfigur angezeigt. Wolken sollen dadurch nicht zerstört werden.
- Diagonale Sprünge zwischen zwei Wänden hindurch sind nicht möglich.

```
       # <---- Nicht erreichbar.
       S#
######## <---- Auch nicht erreichbar.
```

Abbildung 4: Diese Spielfigur kommt nicht mehr weiter.

- Diagonale Bewegungen werden als eine Bewegung nach Rechts und dann nach Oben/Unten behandelt. Der Zwischenschritt wird nur bei Stacheln oder erreichtem Ziel ausgegeben.

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

```
        S
########
Leider verloren. Das Level wird neu gestartet.
```

Abbildung 5: Spielfigur springt (links). Spiel wird bei Zwischenschritt (rechts) wegen Stacheln beendet.

- Ein Sprung endet vorzeitig, falls die Spielfigur von unten eine Plattform trifft.

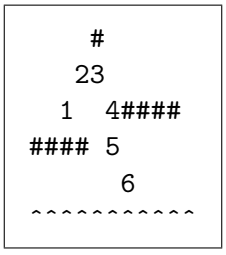


Abbildung 6: Die Flugbahn einer Spielfigur, die nach dem sechsten Schritt in Stacheln landen würde.

## 2.5 Aufruf des Programms:

- Dem Programm werden beim Aufruf die Eingabe-Datei, Ausgabe-Datei und mehrere Level-Dateien übergeben, die nacheinander durchgespielt werden.
- Falls „-“ als Pfad für die Eingabe-/Ausgabedatei übergeben wurde oder falls keine Dateien übergeben wurden, wird stattdessen die Standardeingabe/-ausgabe verwendet.
- Falls keine Level-Dateien übergeben wurden, wird stattdessen `level/1.txt` verwendet werden.

Einige Beispiel-Aufrufe:

```
./jumpandrun eingabe.txt ausgabe.txt level1.txt level2.txt level3.txt
./jumpandrun
./jumpandrun -
./jumpandrun eingabe.txt
./jumpandrun eingabe.txt -
./jumpandrun - ausgabe.txt
./jumpandrun - - level1.txt
```

Beachten Sie, dass die meisten Eingabedateien auf den ersten Blick leer aussehen werden, da dort meist nur Leerzeichen und Zeilenumbrüche drin stehen.

## 2.6 Rückgabewerte

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

- 0, wenn alles ordnungsgemäß funktioniert.
- 1, wenn eine Datei nicht geöffnet werden konnte.
- 2, wenn es bei einem Lesevorgang einen Fehler gab. Beachten Sie, dass nicht nur der erste Lesevorgang, sondern auch noch spätere Lesevorgänge fehlschlagen können.

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

## 3 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 wurde wahrscheinlich ein zu großes Array auf dem Stack angelegt oder möglicherweise der Stack mit rekursiven Funktionsaufrufen gesprengt.
- `make codestyle`: Überprüft teilweise die Einhaltung der Code-Vorgaben aus dem Lernmodul.
- `make clean`: Räumt auf und löscht Ausgabe-Dateien, da diese nicht hochgeladen werden müssen.

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

## 4 Wichtige Anforderungen:

- Der Speicher ihres Spielfelds und eventuell weiterer Datenstrukturen muss mit `malloc` und `free` dynamisch verwaltet werden. Insbesondere darf es keine feste Obergrenze für die Größe der Level-Datei, für die Dauer eines Spiels oder die Anzahl an verschiedenartigen Objekten auf dem Spielfeld (abgesehen von der Spielfigur) 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 Code-Vorgaben im Lernmodul. Insbesondere müssen Sie Ihren Code in Funktionen mit einer Länge von jeweils höchstens 60 Zeilen gliedern. Dies gilt auch für die main-Funktion. 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!

# 5 Beispiele

```
0                  <---- Kommentar: Bisher 0 Befehle verarbeitet.


  G   Z            <---- Goldmünze G und Ziel Z.
S #   #            <---- Spielfigur S startet hier.
#
^^^^^^^^^^^^^^^^^^ <---- Stachelige Stacheln.
1                  <---- Ein Befehl verarbeitet (Leerzeichen + Zeilenumbruch)


 SG   Z            <---- Spielfigur springt.
  #   #
#
^^^^^^^^^^^^^^^^^^
2

  S                <---- Spielfigur sammelt Goldmünze ein.
      Z
  #   #
#
^^^^^^^^^^^^^^^^^^
3
   S               <---- Spielfigur springt immer noch.

      Z
  #   #
#
^^^^^^^^^^^^^^^^^^
4

    S              <---- Spielfigur beginnt zu fallen.
      Z
  #   #
#
^^^^^^^^^^^^^^^^^^
5


     SZ            <---- Spielfigur fällt weiter.
  #   #
#
^^^^^^^^^^^^^^^^^^
6


      S             <---- Spielfigur hat gerade so das Ziel erreicht.
  #   #
#
^^^^^^^^^^^^^^^^^^
Gewonnen! Punktzahl: 1
```

Abbildung 7: Kommentierte Ausgabe für ein einfaches Level. Die Spielfigur springt, sammelt eine Goldmünze ein und erreicht gerade so noch das Ziel.

```
0              <---- Kommentar: Bisher 0 Befehle verarbeitet.
GGGGGGGG
GGG#GGGG
GGGGGGGG
GS#GGGZG       <---- Spielfigur und Ziel in dieser Zeile.
########
1
GGGGGGGG
GGG#GGGG
GSGGGGGG       <---- Spielfigur springt nach oben wegen Plattform rechts.
G #GGGZG
########
2
GGGGGGGG
GGS#GGGG
G  GGGGG       <---- Goldmünzen wurden während Zwischenschritt eingesammelt.
G #GGGZG
########
3
GGSGGGGG       <---- Sprung weiter nach oben
GG #GGGG       <---- aufgrund von Plattform in dieser Zeile.
G  GGGGG
G #GGGZG
########
4
GG SGGGG       <---- Weiter auf Plattform.
GG #GGGG
G  GGGGG
G #GGGZG
########
5
GG   GGG
GG #SGGG       <---- Spielfigur fällt von Plattform.
G  GGGGG
G #GGGZG
########
6
GG   GGG
GG #  GG
G  GGSGG       <---- Spielfigur fällt weiter.
G #GGGZG
########
7
GG   GGG
GG #  GG
G  GG  G
G #GGGSG       <---- Spielfigur erreicht Ziel.
########
Gewonnen! Punktzahl: 10
```

Abbildung 8: Die Spielfigur springt seitlich gegen eine Plattform und sammelt dabei Goldmünzen ein, auch bei den Zwischenschritten.

```
0              <---- Kommentar: Bisher 0 Befehle verarbeitet.

 Z
S
#G             <---- Goldstück
^^^
1              <---- Schritt 1

 Z

#S             <---- Spielfigur sammelt Goldmünze ein.
^^^
2              <---- Schritt 2

 Z

#G
^^S            <---- Spielfigur landet auf Stacheln.
Leider verloren. Das Level wird neu gestartet.
0              <---- Spiel startet neu (bei Schritt 0).

 Z
S
#G
^^^
1               <---- (Wieder) Schritt 1

 S              <---- Spielfigur springt zum Ziel.

#G
^^^
Gewonnen! Punktzahl: 0
```

Abbildung 9: Die Spielfigur versucht gierig eine Goldmünze einzusammeln und stirbt. Danach startet das Spiel neu, die Spielfigur springt und erreicht das Ziel (ohne Goldmünze).