Übungsblatt 3

Ziel dieses Übungsblattes ist es, dass Sie sich mit dem Prinzip der Rekursion vertraut machen.

Aufgabe 1 – Maximum bestimmen

In Programmierung 1 haben wir das Maximum in einem Array iterativ bestimmt. In dieser Aufgabe sollen Sie das Maximum in einem Integer-Array rekursiv bestimmen. Sie können auch Hilfsfunktionen verwenden, falls notwendig. Funktionsprototyp und Main-Funktion finden Sie in dem folgenden Programmausschnitt:

```
int bestimmeMax(int[],int);

int main()
{
  int arr[] = {3,17,5,21,8,12};
  int max = bestimmeMax(arr,6);
  printf("Das Maximum ist %d", max);
  return 0;
}
```

Aufgabe 2 – Suchen von Elementen in einem Array

In der Vorlesung wurde ein rekursiver Algorithmus zur Suche von Elementen in einem sortierten Feld vorgestellt.

a) Implementieren Sie diesen Algorithmus. Ändern Sie dabei die Funktion so ab, dass zu Beginn eines Aufrufs der rekursiven Funktion die Argumente left, right und middle ausgegeben werden.

b) Testen Sie Ihren Algorithmus anhand des folgenden Feldes aus indem Sie die Zahlen 17 und 125 suchen.

```
int arr[] = {7, 8, 14, 17, 19, 23, 45, 47, 56, 58, 61, 73, 75, 79,
81, 88, 92, 95, 101, 110, 123, 125, 134, 135, 156, 167, 177, 178,
179, 234, 237, 888};
```

c) Skizzieren Sie den Aufrufbaum für die Tests aus b). Geben Sie zu jedem Aufruf die aktuellen Parameter an. Nummerieren Sie die Aufrufe entsprechend der Reihenfolge, in der die Funktionen aufgerufen bzw. beendet werden.

Aufgabe 3a – Aufbau eines Dungeons (Zusatz)

In unserer bisherigen Spielewelt mussten Räume noch umständlich direkt im Programmcode erzeugt werden.

Einfacher und wesentlich flexibler ist es, wenn die Räume über eine externe Konfigurationsdatei erstellt (rooms.txt im Stud.IP) und zu einem Dungeon aufgebaut werden. Den Aufbau der Datei finden Sie auf der rechten Seite. In der ersten Zeile steht die Anzahl der Räume in der Datei. Danach folgen die einzelnen Räume durch Zeilenumbruch voneinander getrennt. Die Werte eines Raumes sind durch Komma getrennt (ID,Name,LP_Change). Am Ende der Räume folgt eine Leerzeile, bevor die Verbindung der Räume aufgelistet ist. In jeder Zeile steht die ID des Raums und welche anderen Räume im NORDEN, OSTEN, SÜDEN, WESTEN zu finden sind. Die 0 steht dabei für eine Wand.

```
12
1,"Eingang",0
2,"Vorratskammer",0
3,"Thronsaal",0
4,"VerfluchterRaum",-1
5,"Heilbrunnen",2
6,"Hoelleneingang",0
7,"Schlafgemach",0
8,"Friedhof",-1
9,"Tempel",1
10,"Geheimraum",0
11,"Garten",0
12,"Kueche",0

1,6,0,0,2
2,4,0,3,0
3,0,0,0,0
4,0,0,0,5
5,0,0,0,0
6,7,9,0,0
7,0,0,0,8
8,0,0,0,0
9,0,10,0,0
10,11,0,12,0
11,0,0,0,0
12,0,0,0,0
```

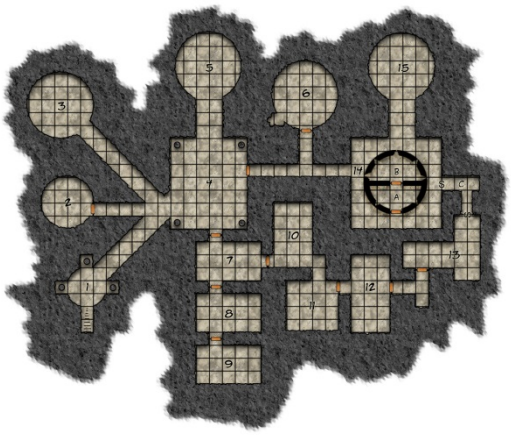
Schreiben Sie eine Funktion **createMap().** Diese bekommt den Namen der Raumdatei übergeben, liest die Datei ein und erzeugt zunächst alle aufgelisteten Räume in einem Array. Die Id des Raumes kann dabei sehr gut als Index in dem Array verwendet werden. Legen Sie in jeden Raum einen zufälligen Gegenstand. Dazu hatten wir in einer früheren Aufgabe eine globale Variable TREASURES angelegt und diese mit Hilfe der Funktion **readTreasureFile()** befüllt. Legen Sie in jeden Raum zufällig einen Gegenstand aus dieser globalen Variable.

Nachdem alle Räume erstellt wurden, müssen diese noch entsprechend der Informationen in der Datei verbunden werden. Am Ende liefert die Funktion den Pointer auf den ersten Raum zurück. Verwende Sie alle bisher implementierten Funktionen wie z.B. **createRoom(), setRoomInDirection().** zum Lösen dieser Aufgabe.

Sollte Ihre aktuelle Spielewelt nicht funktionsfähig sein, dann finden Sie im Stud.IP eine Datei ohne die createMap() Funktion („Spielewelt(ohne Map).txt“).

Sollte Ihnen die Aufgabe zu schwer erscheinen, dann können Sie auch direkt zur Aufgabe 3b übergehen und sich die Datei herunterladen, die bereits die Funktion createMap() enthält („Spielewelt(mit Map).txt“).

Aufgabe 3b – Durchsuchen eines Dungeons



Den Aufbau eines Dungeons kann man als bidirektionalen Graphen auffassen. Zum Durchsuchen eines solchen Graphen eignen sich u.a. rekursive Algorithmen.

Laden Sie sich die Datei „Spielewelt(mit Map).txt“, sollte ihre eigene Spielewelt derzeit nicht funktionsfähig sein. In dieser Datei wird bereits ein Dungeon korrekt aufgebaut.

Hilfreich wäre es nun den kompletten Dungeon auf der Konsole auszugeben. Schreiben Sie hierzu eine rekursive Funktion void printDugeon(Room*). Die Funktion bekommt den Pointer auf den Startraum übergeben, durchläuft rekursiv alle weiteren Räume und gibt auf der Konsole Informationen über die Räume aus.

Schreiben Sie nun eine **rekursive** Funktion **countRooms()**. Die Funktion bekommt einen Pointer auf den Eingang des Dungeons übergeben und soll die Gesamtanzahl an Räumen ermitteln. Die ermittelte Anzahl ist der Rückgabewert der Funktion.

Ebenfalls könnte man sich vorstellen, dass man nach bestimmten Gegenständen suchen kann. Sie möchten zum Beispiel wissen, in welchem Raum ein Heiltrank zu finden ist. Schreiben Sie eine rekursive Funktion **Room* findItem(char *itemName).** Die Funktion bekommt eine Zeichenkette mit dem Namen des Gegenstandes übergeben und liefert den ersten Pointer auf einen Raum zurück in dem dieser Gegenstand liegt. Sollte in keinem Raum dieser Gegenstand liegen, dann kommt NULL zurück.