Dokumentation DES PRAKTIKUMSPROJEKTES

Informatik 3

Inhalt

[**Erste Sitzung - Einführung in Linux (UBUNTU)** 1](#_Toc117689025)

[Zweite Sitzung - Aufgabenstellung und Erklärung 3](#_Toc117689026)

[Dritte Sitzung - erste Programmierung 5](#_Toc117689027)

**Erste Sitzung - Einführung in Linux (UBUNTU)**

Zunächst wurde uns in der ersten Sitzung eine umfassende Einführung für Linux in der Ubuntu Umgebung gegeben.

Allgemeine Befehle wurden uns erklärt und demonstriert.

**Befehle von Ubuntu**

cd steht für change directory und führt uns zu dem gewünschten Ziel.

z.B. cd 10\_Workspace/03\_github/Info3Prak und dann befinden wir uns in diesem Ordner.

-rm steht für Remove und löscht Elemente.

**Für das Hochladen mit Terminal**

* cd 10\_WORKSPACES/03\_github/
* git clone
* ls
* cd Konstruktor\_Schachbrett/
* cp chess1.cpp
* cp chess1.cpp ./../..
* git clone
* cp chess1.cpp ./ChessInf3Pra/
* cd ChessInf3Pra/
* git status
* git add chess1.cpp
* git commit -m "VERSIONSNAME"
* git config --global user.email "EMAILADRESSE"
* git status
* git commit -m "VERSIONSNAME"
* git push INDIVIDUELLEADRESSE@github.com/novalidcode/ChessInf3Pra.git

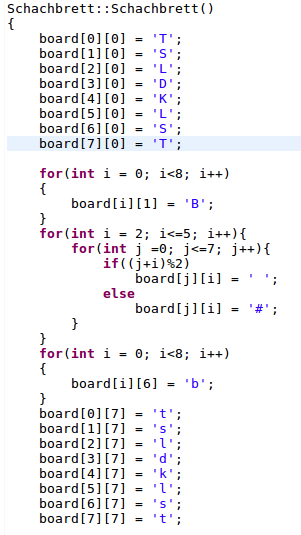
# Zweite Sitzung

# In der zweiten Sitzung haben wir uns mit der allgemeinen Gestalt des Programmes auseinandergesetzt. Hierbei haben wir einen Strukturdiagramm erstellt um uns einen Leitfaden für das Gesamte Projekt zu erstellen.

# C:\Users\amalek-sup\Downloads\c65832e0-d767-4ee0-b8c9-a1fe2591f09c.jfifg - Aufgabenstellung und Erklärung

# 

# Dritte Sitzung - erste Programmierung



Zunächst wird dafür eine Klasse Schachbrett definiert.

Der Konstruktor dieser Klasse wird initialisiert und gerufen: Der "Board Array" soll so angeordnet sein, dass die Figuren auf der Linken Array Platziert werden es werden 8 Zeilen erstellt mit dem Ziel, 8 Spalten für das Spielbrett zu erzeugen.

Auf der rechten Seite des Board Arrays wird zunächst Standardgemäß [0] eingegeben denn dieser sei zunächst die Startlinie des Brettes, also das Erste Feld für den Spieler [Auf der Konsole im Terminal die UNTERSTE Zeile]. Die erste Row ist mit „T“ für Turm, die zweite mit „S“ für Springer, die dritte mit „L“ für Läufer, die vierte mit „D“ für Dame, die fünfte mit „K“ für König und dann die sechste bis achte sind die Doppelfiguren wie im Default Anordnung des Spieles vorgesehen.

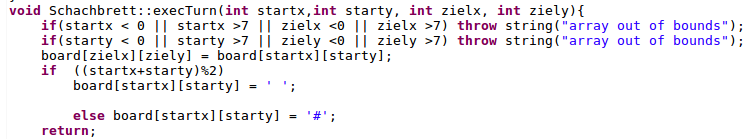
Der Code wiederholt dann jede der Acht Rows und erzeugt für jede Row eine Spalte pro Iteration.

Für alle Spalten in zweiter Zeile werden „B“ für Buben eingesetzt.

Danach versucht man mit Modulo „der Restlose Verteiler, die schwarzen Felder zu markieren. Das geschieht in dem alle leerstehende Felder mit Leer bezeichnet aber sollte die Zahl der Spalte und Zeile nicht restlos Teilbar sein dann wird die angesprochene Zeile/Spalte Feld mit „#“ bezeichnet.

Dieser wird dann für alle Spalten wiederholt solange wir nicht i+j haben denn diese sollen später mit # bezeichnet werden.

Wir setzen auch für den Gegenspieler dieselben Figuren aber in Kleinbuchstaben. Das ist im Grunde nur Wiederholung der obengenannten Strategie.



Dieser Funktion beginnt an der Stelle der Aktuellen Figur des Spielers und bewegt ihn an die Position seines vermeintlichen Zieles.

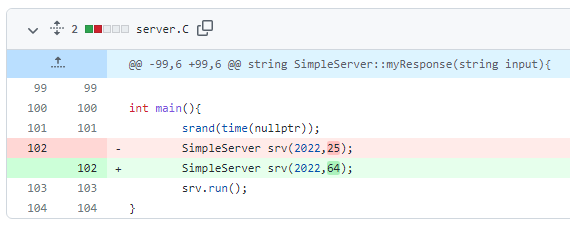
Der Code startet mit der Kontrolle ob „Startx und Starty außerhalb des zulässigen Bereichs sind. Die Zulässige Bereich ist definiert innerhalb von 0 und 7.

Sollten die gegebenen Werte NICHT im zulässigen Bereich sein wird „Array out of bound“ als Fehlermeldung angezeigt.

Des Weiteren wird geprüft ob auch die „Zielx“ und „ziely“ ebenfalls im zulässigen Bereich von 0 bis 7 liegen und die gleiche Fehlermeldung taucht auf, wenn das nicht der Fall ist.

Sollte die Fehler nicht auftreten dann werden die neue zielx und ziely Werte für Startx und Starty werte eingesetzt. Danach wird nochmal geschaut ob die Summe von Startx plus Starty restlos durch zwei teilbar ist um die Farbe der Felder nach einem Spielzug nochmal zu korrigieren.

Der Code versucht festzustellen, ob sich eine Figur auf dem Brett befindet. Wenn dies der Fall ist, wird wahr zurückgegeben; andernfalls wird false zurückgegeben. Die erste Codezeile prüft auf 'a' und 'z'. Wenn einer gefunden wird, gibt die Funktion false zurück. Andernfalls gibt die Funktion true zurück. Die zweite Codezeile prüft auf 'A' und 'Z'. Wenn einer gefunden wird, gibt die Funktion false zurück. Andernfalls gibt die Funktion true zurück. Der Code versucht festzustellen, ob eine bestimmte Figur auf dem Brett ein "W" ist oder nicht. Die erste if-Anweisung gibt false zurück, da piece kleiner als 'a' und größer als 'z' ist. Die zweite if-Anweisung gibt true zurück, da piece gleich 'A' und gleich 'Z' ist.



Der Code versucht, zwei Serverinstanzen mit derselben Portnummer, aber unterschiedlichen IP-Adressen zu erstellen.

Danach wird der server gestartet.

**Server.C Datei**

https://github.com/novalidcode/CIP3/blob/main/server.C

Der Code beginnt mit der Deklaration einer Variablen namens myResponse.

Dies ist die Funktion, die verwendet wird, um auf Eingaben des Benutzers zu reagieren.

Die nächste Codezeile erstellt eine Instanz von ChessSpecial, einer in chess\_special.hpp definierten Klasse, und weist sie der Variablen chess zu.

Die nächste Codezeile deklariert einen String namens myResponse und initialisiert ihn mit „Hello World!

Der Code ist ein Server, der die ChessSpecial-Klasse implementiert.

Der Zweck dieses Codes besteht darin, ein Schachspiel zwischen zwei Spielern zu erstellen und dann das Ergebnis des Spiels zurückzugeben.

Der Konstruktor für SimpleServer nimmt eine ganzzahlige Portnummer und eine ganzzahlige Länge entgegen, die ebenfalls an TCPserver weitergegeben werden.

Um die ChessSpecial-Klasse zu implementieren, müssen wir sie zuerst erstellen: ChessSpecial::ChessSpecial() { chess = new IChess(); } Dadurch wird eine neue Instanz unseres Schachobjekts mit Standardeinstellungen erstellt.

Nachdem wir unser Schachobjekt erstellt haben, können wir damit beginnen, Methoden dafür zu implementieren: void ChessSpecial::move(int x

Der Code beginnt mit der Überprüfung, ob der Benutzer „INIT“ oder „HASH“ eingegeben hat.

Wenn ja, stellt es ein Schachbrett auf und beginnt zu spielen.

Wenn nicht, dann gibt es eine Fehlermeldung aus.

In der nächsten Zeile prüft der Code, welche Eingaben gemacht wurden.

Es vergleicht die Eingabe mit 0-4 und führt bei Gleichheit die Funktion aus, die dieser Zahl entspricht (0=init; 1=hash; 2=exec).

Die letzten beiden Zeilen dieses Blocks prüfen auf gültige Eingaben und führen eine entsprechende Wendung aus.

Der Code soll überprüfen, ob der Benutzer einen gültigen Zug eingegeben hat.

Wenn ja, dann führt es die Runde aus.

Wenn nicht, dann gibt es eine Fehlermeldung aus.

Der Code beginnt mit einer Zeichenfolge namens „coords“, die mit einer leeren Zeichenfolge initialisiert wird.

Der Code fragt dann den Benutzer nach Startkoordinaten und Zielkoordinaten.

Die erste Zeile der While-Schleife prüft, ob cin fehlgeschlagen ist, was bedeutet, dass es EOF oder eine andere Fehlerbedingung erreicht hat.

Wenn dies der Fall ist, wird das Programm seinen fröhlichen Weg fortsetzen, ohne den Benutzer erneut um eine Eingabe zu bitten.

Andernfalls fordert es den Benutzer erneut zur Eingabe auf, indem "Startkoordinaten eingeben:" ausgedruckt wird.

Dann liest es sx und sy als ganze Zahlen und zx und zy als Zeichen ein (Zeichen werden verwendet, weil sie jeden Buchstaben darstellen können).

Nach dem Einlesen dieser Werte druckt es sie zusammen mit ihrem entsprechenden Koordinatenwert (sx-'a',sy-1,zx-'a',zy-1) auf dem Bildschirm aus, bevor es überprüft, ob dieses Koordinatenpaar gültig ist, indem es Schach verwendet. >istGültig().

Wenn nicht, dann ist dieses Koordinatenpaar ungültig und die Ausführung wird am Ende der do{}while-Schleife fortgesetzt, wo alle Eingaben erschöpft sind.

Der Code führt die Runde eines Schachspiels aus.

**Client.C Datei**

<https://github.com/novalidcode/CIP3/blob/main/client.C>

Der Code beginnt mit der Initialisierung eines TCP-Clientobjekts und der Verbindung mit dem Host.

Das Spiel wird dann mit einem leeren Brett initialisiert, das später im Code gefüllt wird.

Die While-Schleife beginnt mit der Frage, ob der Benutzer mit dem Schachspielen beginnen möchte oder nicht.

Wenn dies der Fall ist, werden sie nach ihrem Zug gefragt (was über cin erfolgt).

Es prüft dann, ob dieser Zug n oder N war und setzt goOn entsprechend.

Der Code ist ein einfacher TCP-Client.

Der Code erstellt zunächst eine Instanz der TCPClient-Klasse und stellt dann eine Verbindung mit dem Host mit dem Namen „localhost“ an Port 2022 her.

Der Code fährt fort, indem er eine Eingabeaufforderung ausgibt und fragt, ob der Benutzer eine Schachpartie beginnen möchte.

Wenn sie ein Spiel nicht beginnen möchten, werden sie erneut aufgefordert und gefragt, ob sie mit dem Schachspielen aufhören möchten.

Wenn sie sich dafür entscheiden, das Schachspielen nicht aufzugeben, druckt es eine weitere Eingabeaufforderung aus, in der sie gefragt werden, was ihr Zug im Schach sein wird.

Es wartet dann, bis der Benutzer 'N' oder 'y' eingibt.

Der Code beginnt mit der Initialisierung des Spiels mit einer Zeichenfolge von "INIT".

Der Code sendet dann eine Nachricht an den Server und wartet auf eine Antwort.

Bekommt er ein OK, fährt er fort.

Andernfalls stoppt es und druckt einen Kommunikationsfehler aus.

Bei der nächsten While-Schleife wird es interessant: while(goOn){ string cstate = game->getState(); c.sendData(string("HASH")); msg = c.receive(64); Wenn (cstate == msg) {goOn=0; } else { cout <<"Vom Server empfangene Nachricht: "<

Der Code ist eine Schleife, die so lange läuft, bis der Spieler gewinnt oder verliert.

Die erste Codezeile richtet das Spiel mit einem Anfangszustand ein, der Hash 0 ist.

Die nächste Zeile sendet Daten an den Server und wartet auf eine Antwort.

Der Zustand des Spielobjekts wird dann überprüft, um zu sehen, ob es sich geändert hat, bevor weitere Daten gesendet werden.

Wenn er sich geändert hat, wird der neue Wert zurückgesendet und erneut überprüft, bis entweder der Spieler gewinnt oder verliert.

Wenn in diesem Fall goOn == 0 ist, wissen wir, dass der Benutzer verloren hat, weil cstate == msg .

Der Code beginnt mit der Deklaration einer Variablen, die die Koordinaten des Startpunkts enthält.

Das Spiel wird dann initialisiert und mit einem Aufruf von print() gestartet.

Der Code prüft, ob Eingabekoordinaten vorhanden sind, was er in einer Endlosschleife durchführt.

Wenn es keine Eingabekoordinaten gibt, druckt es "ERROR" und setzt seinen fröhlichen Weg fort.

Wenn Eingabekoordinaten vorhanden sind, werden sie auf Gültigkeit geprüft, bevor sie als Teil der rundenbasierten Spiellogik ausgeführt werden.

Wenn Sie Schritt für Schritt sehen möchten, was in diesem Programm vor sich geht, sehen Sie sich meine Videolektion an: https://www.youtube.com/watch?v=L7ZY0z8iVwA

Der Code ist ein Ausschnitt des Spiels "Chess", das einen Fehler enthält.

Der Code druckt das Schachbrett aus und beginnt dann mit der Eingabe von Koordinaten für den Zielort auf dem Brett.

Wenn eine Koordinate ungültig ist, wird ERROR ausgegeben, und zwar so lange, bis alle Koordinaten gültig sind.

Der Code beginnt mit der Deklaration einer Zeichenfolge namens coords.

Dies ist die Position des Charakters des Spielers auf der Karte.

Die erste Zeile deklariert, dass coords ein Array von ganzen Zahlen sein wird, was bedeutet, dass es mehrere Werte enthalten wird.

Die nächste Zeile fügt sx und sy zu den Koordinaten hinzu, die zwei Koordinaten dafür sind, wo Ihr Charakter auf der Karte beginnt.

Als nächstes fügen wir zx und zy hinzu, um die x- und y-Koordinaten dafür zu erhalten, wo Ihr Charakter landet, nachdem er sich auf der Karte bewegt hat.

Der Code sendet dann mit sendData() Daten von c an den Server.

Es sendet 32 Zeichen auf einmal, weil so viele Bytes in einem einzelnen Zeichenwert (32) enthalten sind.

Dann wartet es auf eine Antwort vom Server mit Receive().

Wenn Receive() zurückkehrt, prüft es, ob msg nach dem Empfang von 32 Bytes irgendwelche Zeichen übrig hat oder nicht, indem es sie mit 0-2 (einschließlich) vergleicht.

Wenn sie alle aufgebraucht sind, bedeutet dies, dass beim Senden der Daten etwas schief gelaufen ist, sodass cout „exec error“ gefolgt von „msg = “ gefolgt von dem, was gesendet wurde, ausgibt, bevor Sie mit Schritt 4 unten fortfahren.

Wenn sie noch nicht aufgebraucht sind, bedeutet dies, dass alles gut gelaufen ist, also gibt cout "OK" aus, gefolgt von seiner eigenen Nachricht, gefolgt von dem, was gesendet wurde, bevor es fortfährt

Der Code sendet die Zeichenfolge "GET\_TURN" an den Server und erhält dann eine Nachricht mit "OK" zurück.

Der Code beginnt mit der Deklaration einer Variablen namens game.

Dies ist das Objekt, das verwendet wird, um alle Informationen über das Spiel zu speichern, wie z. B. den Zug des Spielers und den aktuellen Punktestand.

Als nächstes deklariert es zwei Variablen namens msg und endl.

Der erste wird mit einem Anfangswert von "Server führt Turn: " deklariert.

Der zweite wird mit einem Anfangswert von "Ausführen der Runde fehlgeschlagen" deklariert.

Der Code beginnt dann damit, eine Meldung auf dem Bildschirm auszugeben, die besagt: "Server führt Turn aus: ".

Dies geschieht, weil beim Ausführen Ihres Programms Meldungen wie diese ausgegeben werden, um Sie darüber zu informieren, was während der Ausführung Ihres Programms vor sich geht.

Als nächstes, wenn ein gültiger Zug gemacht wurde (mit anderen Worten, wenn der Benutzer 'a' anstelle von '1' eingegeben hat), würde es diesen Zug ausführen und eine weitere Nachricht mit der Aufschrift "Ausführen der Runde erfolgreich" ausgeben, gefolgt von etwas mehr erklärendem Text was während ihres Umzugs passiert ist.