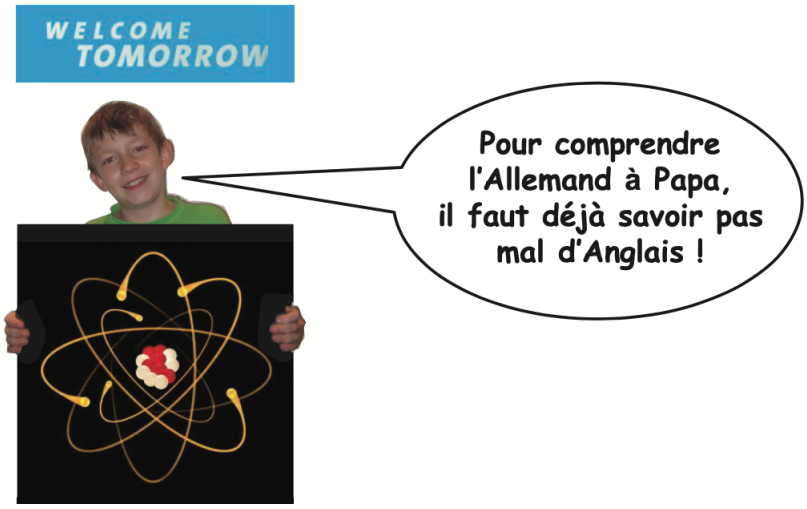
Informatik I Zusammenfassung

Paul Luca Nesemeier pnesemeier@ethz.ch

Chemical Engineering



Professor:

Prof. Dr. Philippe Henry Hünenberger phil@igc.phys.chem.ethz.ch

Informatik I for Chemical Engineering,
Chemistry and Biochemistry
Department of Chemistry and Applied Biosciences
ETH Zürich

15. Mai 2022

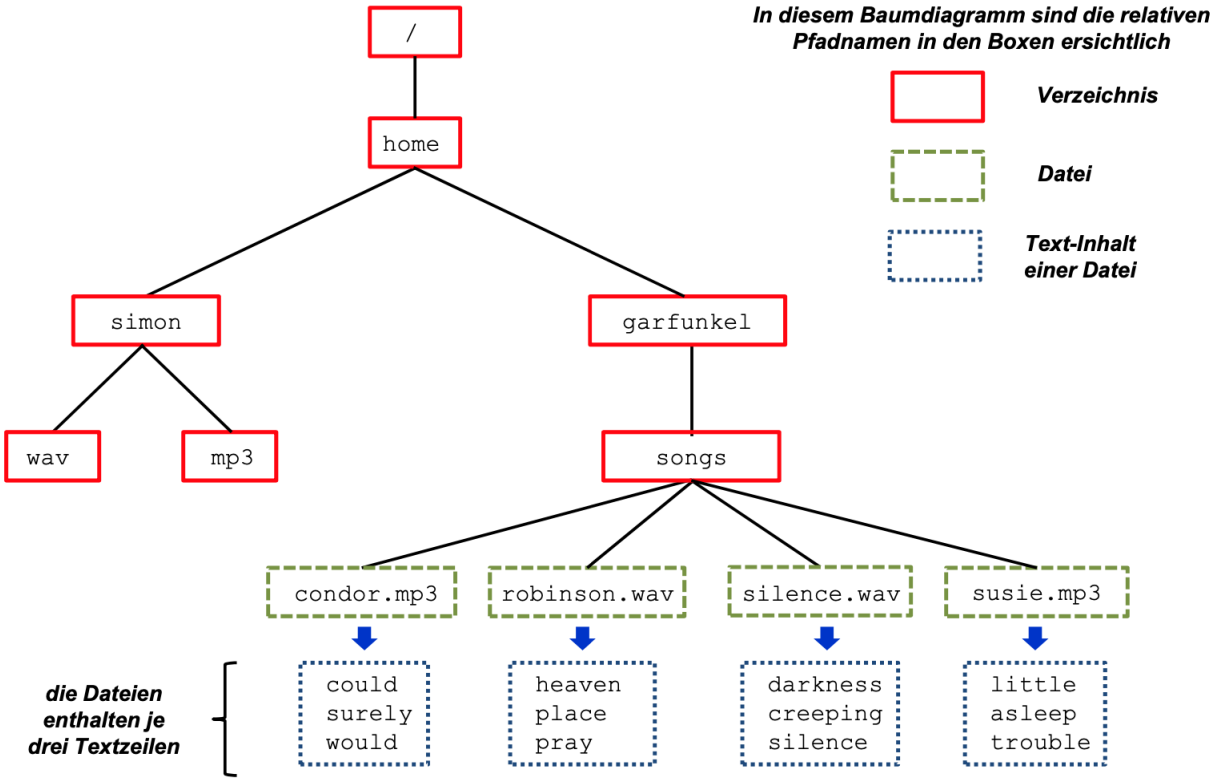
Inhaltsverzeichnis

1 Introduction

Willkommen zu diesem Versuch einer Zusammenfassung von Informatik I, einem Fach, dass man entweder hasst oder liebt. Komischerweise fiel mir Informatik etwas leichter und ich konnte (zumindest bei den Aufgaben) immer gut alles verstehen und lösen. Dieses Dokument wird dir wahrscheinlich keine 6.0 bescheren, aber hoffentlich bringt es dich ein gutes Stück über eine 4.0. Alle Angaben in diesem Dokument sind ohne Gewähr auf Richtigkeit oder Wahrheit. Bei Fragen, Anmerkungen oder Fehlern würde ich mich über eine kurze Mail freuen. Zusätzlich zu diesem PDF gibt es noch zwei Quizlet Sets zur Prüfungsvorbereitung. Sie sind an den entsprechenden Stellen verlinkt.

2 UNIX

UNIX ist eine viel verwendete Programmiersprache, wofür und wieso ist hier aber nicht wichtig. So gut wie immer kommt in der Prüfung ein folgendes Diagramm vor.



Darauf folgt dann immer entweder eine Anzahl an commands, deren Auswirkung man beschreiben muss oder eine Anweisung die man in commands umformen muss. Im generellen sind die Abkürzungen der Commands immer sehr informativ. Wir werden die von Hüneberger in den Übungen genutzten Commands im Folgenden alle durchgehen. Ein Karteikartenset für die Befehle findet ihr unter diesem Link. Alle Befehle sind in einer Tabelle mit Beschreibung zusammengefasst.

2.1 Navigation

Mit der grundsätzlichen Navigation meine ich hier Zeichenfolgen, welche Ordner oder relative paths repräsentieren.

Zeichen	Bedeutung
/	Oberster Ordner des Systems (sog. root Ordner)
~	home-Ordner des Nutzers, heißt in der Prüfung auch immer home
..	Das Verzeichnis über dem aktuellen Verzeichnis
.	Das aktuelle Verzeichis

2.2 Befehle

Die Befehle liste ich hier in Tabellenform auf

Befehl	Bedeutung
`csh`	Wird einmal beim Öffnen des Terminals ausgeführt, startet die c-shell
`pwd`	Gibt den absoluten Weg des aktuellen Verzeichnis (Vz.)
`cd` dir	Ändert das aktuelle Vz. zu dir (wenn es dir gibt)
`cd` oder nur `cd`	Ändert das aktuelle Vz. zum *home* directory
`cd` ..	Ändert das aktuelle Vz. zum eins höheren Vz.
`cd` .	Ändert das aktuelle Vz. zum aktuellen Vz. (kein Effekt)
`ls`	Listet die Inhalte des aktuellen Vz.
`ls` dir	Listet die Inhalte des Vz. dir
`ls` [file/dir ...]	Listet die Inhalte der aufgezählten Vz., sie werden nach einander gedruckt. Bei einer file wird der relative Pfad zur Datei gedruckt
`ls` -R	Listet auch die Inhalte der Vz. im aktuellen Vz. und dessen weitere Vz. usw.
`ls` -p	Listet die Inhalte des aktuellen Vz. auf mit einem / am Ende jedes Eintrags
`ls` -l	Listet die Inhalte des aktuellen Vz. mit mehr Informationen auf (zum Beispiel Berechtigungen für die Dateien)
`ls` -a	Listet die Inhalte des aktuellen Vz., auch die versteckten (Welche mit einem . starten)
`ls` -t	Listet die Inhalte des aktuellen Vz. sortiert nach letztem Änderungsdatum
`ls` -r	Listet die Inhalte des aktuellen Vz. in verkehrter Reihenfolge (nützlich in Kombination mit -t)
`mkdir` [dir ...]	Erstellt ein neues leeres Vz. (mehrere gleichzeitig möglich)
`touch` [file ...]	Erstellt eine Datei mit dem gegebenen Namen wenn sie nicht existiert, wenn sie existiert wird das Änderungsdatum auf den jetzigen Zeitpunkt gestellt (mehrere gleichzeitig möglich)
`cp` [source ...] dest	Kopiert die source (muss eine file sein) zur dest (neuer Filename oder existierendes Vz.) (mehrere sources möglich)
`cp -r` [source ...] dest	Kopiert die source (kann auch Vz. sein) zur dest (neuer Filename / Vz. oder existierendes Vz.) (mehrere sources möglich)
`mv` [source ...] dest	Bewegt die source (existing file / Vz.) zur dest (kann neue file / Vz. sein)(mehrere möglich)

Befehl	**Bedeutung**
`rm` [target ...]	Löscht eine Datei (mehrere möglich)
`rmdir` [target ...]	Löscht ein leeres Vz. (mehrere möglich)
`rm` -r [target ...]	Löscht eine Datei / Vz. welches auch gefüllt sein kann (mehrere möglich)
`chmod` perm file/dir	Ändert die Berechtigungen für eine Datei / Vz. (siehe Berechtigungen)
`echo` text	Druckt den in Anführungszeichen gesetzten Text zum standard output (Nützlich mit |)
`cat` [file ...]	Mit keinem Argument, druckt input zum Output; bei einer gegebenen Datei wird der Inhalt zum Output gegeben, bei mehreren Dateien werden die Inhalte hintereinander zum Output gegeben
`tee` file	Kopiert den Input und gibt ihn zum Output und in die Datei
`man` command	Gibt eine Anleitung für den Befehl
`which` command	Gibt den Pfad der Systemdatei welche den Befehl beinhaltet
`alias`	Gibt alle alias des Befehl an
`diff` file1 file2	Gibt die Unterschiede Zeile für Zeile mit Zeilennummern zweier Dateien zum Output
`paste` file1 file2	Bringt die Dateien spaltenweise zusammen zum Output
`wc` [file ...]	Gibt die Anzahl von Zeilen, Wörtern und Zeichen in der Datei (mit -l nur die Zeilen)
`head` -n num file	Gibt die ersten x Zeilen einer Datei an (x = num)
`tail` -n num file	Gibt die letzten x Zeilen einer Datei an (x = num)
`grep` pattern [file ...]	Sucht und gibt die in pattern angegebenen Zeilen in der Datei und gibt sie zum standard output
`grep` -i pattern [file ...]	Macht die Suche spezifisch auf Groß- und Kleinschreibung
`grep` -v pattern [file ...]	Sucht nach Zeilen die das pattern nicht enthalten
`grep` -n pattern [file ...]	Gibt zusätzlich die Zeilennummern an
`grep` -c pattern [file ...]	Gibt zusätzlich die Anzahl an Zeilen an die das pattern enthalten
`sort` [file ...]	Sortiert die Inhalte der der Datei
`uniq` [file ...]	Löscht alle doppelten Zeilen in einer gegebenen Datei
`more` file	Druckt die Inhalte der Datei seitenweise (press `q` to quit)
`less` file	Druckt die Inhalte der Datei seitenweise (press `q` to quit) (more fancy version, fragt mich bitte nicht)

Wie man sieht sind diese Befehle zum Großteil mit ihrer Abkürzung recht intuitiv. Die Angaben mit einem -, spezifizieren den Befehl und können auch kombiniert werden, wie zum Beispiel `ls -Rla`.

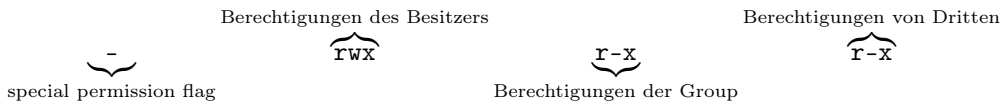
2.3 Berechtigungen

In Unix kann man drei verschiedenen Gruppen drei verschiedene Rechte zuteilen.

Um zunächst einmal diese Berechtigungen zu sehen, nutzen wir den oben genannten `ls` mit dem Zusatz `-l`, also kombiniert `ls -l`. Hier werden nun alle Dateien gelistet zusammen mit folgendem Output:

```
[user@comp per]$ ls -l
-rwxrwxrwx user domainuser filename
```

Wir teilen das ergebnis in drei Teile, die zweite Spalte gibt den Besitzer der File aus (in diesem Fall der user selbst) und die dritte Gruppe gibt an welcher Gruppe man selbst angehört gefolgt vom Namen der Datei. Was uns hier aber interessiert ist der erste Teil.



Die special permission flag ist unwichtig (`-` bei files und `d` bei Directories). Der drei Buchstaben Code jeder Gruppe ist wie folgt definiert: `r` steht für read, `w` für write und `x` bedeutet execute. Sollte die gegebene Berechtigung nicht gegeben sein, steht anstatt des Buchstabens ein `-`.

2.3.1 1. Methode zum Verändern der Berechtigungen

Als Besitzer der Datei kann man die Berechtigungen natürlich auch abändern mithilfe des Befehls `chmod`. Dazu kann man folgenden Sytax nutzen:

```
[user@comp per]$ chmod u-r filename
```

Hierbei werden nun die Leserechte (`r`) des Users (`u`) weggenommen. Dabei kann man `r` natürlich auch durch `w` oder `x` und `u` durch `g` (group) oder `a` (all) ersetzen. Um Leserechte zu geben ersetzt man das - mit einem +.

2.3.2 2. Methode zum Verändern der Berechtigungen

Diese Methode ist nun sehr viel schneller, denn man arbeitet mithilfe eines sogenannten octal code. Hierbei wird wieder `chmod` verwendet, jedoch schreibt man dahinter einen dreistelligen code, wobei die erste Stelle für die Rechte des Users steht, die zweite für die der Group und die dritte für die von allen.

Der Code setzt sich nun wie folgt zusammen. Leserechte bekommen die Nummer 4, Schreibrechte die 2 und Ausführrechte die 1. Nun addiert man einfach die die Zahlen der Rechte und schreibt den drei Zahlen langen octal code.

2.4 Sonstiges wichtiges

In dieser Sektion ist wichtig zu wissen das viele, bzw. fast alle Befehle einen Input und einen Output erzeugen. Um diese Outputs direkt für weitere Inputs zu nutzen gibt es folgende tools.

2.4.1 Piping

Das Piping kann man sich schon wie der Name sagt wie ein Rohr vorstellen. So verändert man zum Beispiel den Inhalt einer Datei mit einem Befehl und kann dann das Pipe Symbol "|"dahinter schreiben. Dieses fungiert nun als "Verbindungsrohr"welches den Output des Befehls direkt als Input für einen Befehl den man dahinter schreibt nutzt. Folgende Befehlskette nutzen wir als Beispiel.

```
[user@comp per]$ sort file.txt | uniq | wc
```

Diese Command Zeile würde nun den Text in der file.txt alphabetisch sortieren, die doppelten Einträge löschen und die Wörteranzahl ausgeben. Anstatt die Datei in drei separaten Schritte zu modifizieren bzw. auszulesen kann man dies nun in einem Befehl tun.

2.4.2 Redirection

Redirection ist ein tool, das man nutzen kann um Outputs von Befehlen direkt in Dateien auszugeben. Hierbei gibt es wenige Zeichen die man kennen muss.

- Ein ">" auf das ein Dateiname folgen muss erstellt eine Datei (oder ersetzt die Datei mit dem selben Namen) und schreibt den Output des vorangegangenen Befehls hinein.
- Ein ">>" auf das ein Dateiname folgen muss, hängt den Output des vorangegangen Befehls an das Ende der bestehenden Datei an.
- Wenn man hinter die beiden obigen Zeichen noch ein "&" schreibt (also >& oder >>&), so wird nicht nur das Output in die Datei geschrieben sondern auch die Errors.

Es gibt noch <, welches aber nach meinen Informationen nicht prüfungsrelevant ist.

2.4.3 Wildcards

Es gibt in Unix zwei sogenannte Wildcards welche relevant sind. Wildcards sind Platzhalter, das heißt sie ersetzen definite Dateinamen, sodass man eine Gruppe von Dateien mit Befehlen bearbeiten kann.

- Zum Einen gibt es "*", welches eine beliebige Zeichenfolge ersetzt. So würde `*.txt` alle txt Dateien anzielen. Man kann * auch nutzen um alle Dateitypen mit dem selben Namen anzuzielen wie zum Beispiel bei `Text.*`, welches alle Dateitypen mit dem Namen Text anzielt.
- Zum Anderen gibt es "?", welches ein einzelnes Zeichen beliebig macht. So könnte `H?nd.txt` die Textdateien `Hand.txt` als auch `Hund` repräsentieren, jedoch kann auch jedes andere Zeichen eingesetzt werden.

3 Zahlen- und Textformate

3.1 Zahlensysteme

3.1.1 Binärsystem

Das Binärsystem wird hier nicht weiter erläutert, es gibt gute YT-Videos die es in Kürze erklären.

3.1.2 Oktalsystem

Im Folgenden gebe ich ein kurzes "Kochrezept"(s/o an Yana) für das Umwandeln von Binärzahlen in Oktalzahlen.

1. Gliedere die Binärzahl in dreier Päckchen von rechts nach links. Sollten beim letzten Päckchen noch Stellen fehlen kannst du sie einfach mit Nullen ergänzen.
2. Bestimme die Binärzahl der jeweiligen Päckchen. Wenn du sie zusammensetzt hast du deine Oktalzahl.

Beispiel 3.1:

Drücke die Binärzahl $[10100111100010000]_2$ als Oktalzahl aus.

1. Die Dreierpäckchen lauten hier von rechts nach links wie folgt: $[000]_2, [010]_2, [001]_2, [111]_2, [100]_2 and [010]_2$. Das letzte Päckchen wurde mit einer Null aufgefüllt.
2. Mithilfe der des Binärsystems bekommen wir nun die Oktalzahl welche $[247420]_8$ lautet.

3.1.3 Hexadecimalsystem

Das Hexadecimalsystem läuft mit wenigen Untersdchieden relativ ähnlich ab. Auch hier gebe ich ein kurzes "Kochrezept"(s/o an Yana) für das Umwandeln von Binärzahlen in Hexadecimalzahlen. Zunächst erinnern wir uns, dass die Hexadecimalstellen wie folgt benannt werden:

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$[0000]_2$	$[0001]_2$	$[0010]_2$	$[0011]_2$	$[0100]_2$	$[0101]_2$	$[0110]_2$	$[0111]_2$	$[1000]_2$	$[1001]_2$	$[1010]_2$	$[1011]_2$	$[1100]_2$	$[1101]_2$	$[1110]_2$	$[1111]_2$

1. Gliedere die Binärzahl in vierer Päckchen von rechts nach links. Sollten beim letzten Päckchen noch Stellen fehlen kannst du es einfach mit Nullen ergänzen.
2. Bestimme die Binärzahl der jeweiligen Päckchen. Wenn du sie zusammensetzt hast du deine Hexadecimalzahl.

Beispiel 3.2:

Drücke die Binärzahl $[111011111001101001]_2$ als Oktalzahl aus.

1. Die Dreierpäckchen lauten hier von rechts nach links wie folgt: $[1001]_2, [0110]_2, [1110]_2, [1011]_2$ und $[0011]_2$. Das letzte Päckchen wurde mit zwei Nullen aufgefüllt.
2. Mithilfe der des Binärsystems bekommen wir nun die Hexadecimalzahl welche $[3BE69]_{16}$ lautet.

3.2 Integers

Integers sind ganze Zahlen welche mit `int` abgekürzt werden. Negative Integers sind nicht erlaubt, daher wird meistens die erste Zahl des Binärcodes als sog. Sign-Bit genutzt, eine 1 bedeutet dabei ein negatives und eine 0 ein positives Vorzeichen. Eine Integer ist 32 Bit groß (ergo 4 Byte), daher kann sie 2^{32} Zahlen tragen, wovon die Hälfte negativ ist. In `C++` sind die Integers fast immer signed.

3.3 Fixed point notation

Die fixed point notation erlaubt es, auch Dezimalzahlen als Binärzahl zu schreiben. Für die Zahl vor dem Komma wird die Binärzahl immernoch so wie gewohnt gebildet. Hinter dem Komma verändert sich die Situation etwas, bleibt aber vom Konzept her gleich. Die erste Stelle steht nun für die 2^{-1} Potenz etc. Durch ein Beispiel wird die Situation klarer.

Beispiel 3.3:

Drücke die Zahl 11.6875 als Binärzahl aus.

1. Als erstes kümmern wir uns um die Zahlen vor dem Komma. Hier kommt man auf die 8 als höchste kleinste Zweierpotenz, dazu kommen die 2 und die 1 somit bekommen wir hierfür eine Binärzahl von $[1011]_2$.
2. Nun widmen wir uns der Nachkommastelle, da wir über 0.5 sind können wir das schonmal abziehen und bei uns im Kopf die erste Stelle der Binärzahl mit 1 vormerken. Somit bleiben uns noch 0.1875. Die nächst kleine inverse Zweierpotenz die nun passt ist $2^{-3} = 0.125$, abgezogen heißt das dann dass uns noch 0.0625 bleiben, was exakt der 2^{-4}. Potenz entspricht. Somit erhalten wir unsere Binärzahl hinter dem Komma mit $[1011]_2$.
3. Zusammen ergibt das nun die unsigned Binärzahl $[1011.1011]_2$. Signed hieße diese Binärzahl $[10110.1011]_2$ für das positive Vorzeichen.

Final ist noch zu erwähnen dass man die fixed point Schreibweise natürlich auch in ihrer finalen Form weiter umformen kann zu einer oktalen oder sogar hexadecimalen Zahl.

3.4 Floating point notation

Ok hier wirds lustig, aber hear me out, wenn mans einmal checkt, gehts.
Grundsätzlich nutzen wir vier verschiedene Variablen die im Moment noch keinen Sinn machen, jedoch später hoffentlich klarer werden:

- **S** das *sign* bit, ist einstellig und gibt das Vorzeichen an, wie beim fixed point steht hier 0 für positiv und 1 für negativ.
- **M** die *Mantissa* besteht aus 23bit (in einem 32bit Betriebssystem) oder 52bit (in einem 64bit Betriebssystem). Sie repräsentiert die eigentliche Zahlenfolge und ist eine verschobene Form der Fixed Point Schreibweise.
- **E** der Exponent beschreibt den Exponenten der zweier Potenz, welche die Dimension der Binärzahl angibt, mehr dazu im Beispiel.
- E_0 Ist eine Konstante welche bei 32bit 127 beträgt und bei 64bit 255 also der 8. und 9. Zweierpotenz entspricht.

Wir unterscheiden nun die zwei Fälle aus der Prüfung und können so dann hoffentlich das Konzept verstehen.

3.4.1 Dezimalzahl zu Floating Point

Auch hier stelle ich wieder ein allgemeines Kochrezept vor:

1. Als erstes konvertiert man die Dezimalzahl in fixed point Schreibweise. Weiterhin kann man hier direkt **S** ablesen, also das Vorzeichen interpretieren.
2. Nun zieht man die höchstmögliche 2er Potenz aus der Klammer, das heißt man verschiebt das Komma solange nach links, bis nur noch eine 1 vor dem Komma steht. Wie oft man das Komma verschoben hat gibt uns nun den Exponenten der Zweier Potenz, wenn wir zu diesem nun noch 127 (32bit) oder 255 (64bit) (also E_0) addieren erhalten wir unseren *Exponenten* **E**. Die Binärzahl in welcher das Komma verschoben wurde repräsentiert unser *Mantissa* **M**.

3. Nun setzen wir alles zusammen in Binärzahlen, als erstes kommt das *sign* bit, danach folgt 8 oder 9 stellige *Exponent* und als letztes schreibt man die **M**, welche man meistens noch mit Nullen auffüllen muss um die Betriebssystem spezifische Länge der *Mantissa* zu erfüllen.

Gut, das klingt jetzt erst mal sehr abstrakt aber hold on, das Beispiel kommt.

Beispiel 3.4:

Drücke -187.375 als Floating point in einem 32bit System aus. Wir gehen nun streng nach Kochrezept vor.

1. Zunächst wissen wir, dass **S** = 1 ist durch das negative Vorzeichen. Als nächstes wandeln wir die Zahl in fixed point um. Hierbei kommen wir auf folgendes Ergebnis: $[10111011.011]_2$

2. Nun verschieben wir hier das Komma (also den Punkt in englischer Schreibweise) bis nur noch die linkeste 1 vor dem Komma steht. Das wären in diesem Fall die 7 damit lautet die Zweier Potenz 2^7 und unser **E** ist $Potenz + E_0 = E$, also hier 134. Für unser M nehmen wir jetzt nur noch unseren kommaverschobenen Term, schneiden die erste 1 weg und erhalten M = 0111011011. Wenn wir nun alles zusammensetzen erhalten wir

$$Z = \underbrace{1}_{S}\,\underbrace{10000110}_{E}\,\underbrace{01110110110000000000000}_{M}$$

Zwei Dinge sind hier eventuell noch etwas unklar, wie vorher angekündigt wurde E umgewandelt in Binärcode und M wurde rechts mit Nullen aufgefüllt.

3.4.2 Floating Point zu Dezimahlzahl

In die andere Richtung ist es jetzt etwas anders, hier benötigt man die Formel

$$(-1)^S(1+M)\cdot 2^{(E-E_0)}$$

zum Umrechnen. Hier kann man auf das lange Kochrezept verzichten, die Formel ist größtenteils ausreichend. Zwei Punkte sind, dass zum Einen das 1 + M bedeutet, die 1 vor dem Komma, die wir bei der Umwandlung in die Gegenseite weggenommen haben, wieder so mit Komma hinzugefügt wird. Weiterhin muss man E erst aus der Binärzahl umwandeln.

Beispiel 3.5:

Drücke

$$Z = \underbrace{0}_{S}\,\underbrace{10001001}_{E}\,\underbrace{00110100101000000000000}_{M}$$

als Dezimalzahl in einem 32bit System aus.

1. Zunächst wissen wir durch **S** = 0. Als nächstes wandeln wir E in eine Dezimalzahl um und kriegen hierbei 137 raus. Als letztes befreien wir **M** von seinen hinteren Nullen ($[00110100101]_2$) und stellen auf die linke Seite noch die 1 und ein Komma sodass wir $[1.00110100101]_2$ erhalten. Als letztes wissen wir durch die Aufgabe noch dass $E_0 = 127$ sein muss weil wir 32bit haben.

2. Jetzt können wir einfach in unsere Formel einsetzen und erhalten

$$(-1)^0 \cdot [1.00110100101]_2 \cdot 2^{(137-127)} = (1) \cdot [1.00110100101]_2 \cdot 2^{10}$$

als Ergebnis. Somit können wir das Komma um 10 Stellen nach rechts schieben und erhalten $[10011010010.1]_2$. Wenn wir diese fixed point Schreibweise nun zurück in eine Dezimalzahl umwandeln bekommen wir 1234.5 als Ergebnis.

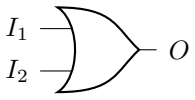
Ein Stichwort welches für floating point Aufgaben genutzt wird ist einfach die Wahl des Betriebssystems also z.B.: "Drücke die Zahl xy in 32bit aus!"

3.5 Textformate

Das von uns genutzte Textformat heißt ASCII und steht für American Standard Code for Information Interchange. Es besteht aus 7 bits ergo 128 Zeichen, die wichtigsten Passagen sind die Zahlen von 0-9 bei 48-57, Die Großbuchstaben von 65-90 und die Kleinbuchstaben von 97-122. Alle diese Zahlen werden teilweise auch hexadecimal geschrieben mit einem vorangestelltem x.

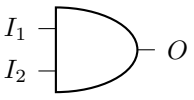
4 Logische Gitter

Logische Gitter sind so gut wie immer Teil der Prüfung und können free points sein. Im folgenden werden die Arten von logischen Gittern einmal alle vorgestellt, die Grafik ist nicht von mir sondern von Alexander Schochs PVK Info 1 Kurs, welcher unter Zusammenfassungen auf der VCS Website gefunden werden kann.



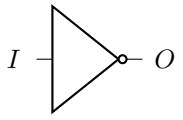
I_1	I_2	O
0	0	0
0	1	1
1	0	1
1	1	1

(a) OR Gate



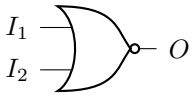
I_1	I_2	O
0	0	0
0	1	0
1	0	0
1	1	1

(b) AND Gate



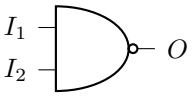
I	O
1	0
0	1

(c) NOT Gate



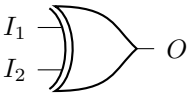
I_1	I_2	O
0	0	1
0	1	0
1	0	0
1	1	0

(d) NOR Gate



I_1	I_2	O
0	0	1
0	1	1
1	0	1
1	1	0

(e) NAND Gate



I_1	I_2	O
0	0	0
0	1	1
1	0	1
1	1	0

(f) XOR Gate

Abbildung 1: Typische Logische Gatter. Das «N»-Präfix steht für «not», das «X»-Präfix für «exclusive».

Die Legende der Schaltkreise ist in der Klausur auch immer gegeben. Um diesen Teil der Prüfung zu üben würde ich empfehlen die Prüfungsaufgaben selbst zu machen, es gibt genügend und sie sind am nächsten an der Realität.

5 Datenbanken

Dies ist ein kurzes Kapitel aus der Vorlesung welches sich mit dem Handling von Datenbanken beschäftigt. Hierbei gibt es 6 Befehle welche relativ intuitiv sind. Wir werden die folgenden Vorlesungsbeispiele nutzen um alle Begriffe zu erklären, dabei geht es um Listen einer Bibliothek, die teilweise personenbezogenen Daten und andererseits buchbezogene Daten beinhalten.

5.1 Minus

Person A

ID Number	Name	Address	Town	Canton
52.137	Wild K.H.	Ochsenstr. 16	Zürich	Zürich
61.100	Scherrer P.	Salzerstr. 20	Biel	Bern
90.101	Cotti B.	Via Almoli 3	Lugano	Ticino
91.200	Huber A.	Bahnhofstr. 20	Basel	Basel-Stadt

Person B

ID Number	Name	Address	Town	Canton
52.137	Wild K.H.	Ochsenstr. 16	Zürich	Zürich
79.206	Huber A.	Bahnhofstr. 20	Winterthur	Zürich
90.101	Cotti B.	Via Almoli 3	Lugano	Ticino
88.001	Boudry C.	Rue du Soleil 1	Crans	Valais

`MINUS` macht das was es sagt, wenn wir nun Person B `MINUS` Person A machen, erhalten wir am Ende:

ID Number	Name	Address	Town	Canton
79.206	Huber A.	Bahnhofstr. 20	Winterthur	Zürich
88.001	Boudry C.	Rue du Soleil 1	Crans	Valais

5.2 Intersect

Person A

ID Number	Name	Address	Town	Canton
52.137	Wild K.H.	Ochsenstr. 16	Zürich	Zürich
61.100	Scherrer P.	Salzerstr. 20	Biel	Bern
90.101	Cotti B.	Via Almoli 3	Lugano	Ticino
91.200	Huber A.	Bahnhofstr. 20	Basel	Basel-Stadt

Person B

ID Number	Name	Address	Town	Canton
52.137	Wild K.H.	Ochsenstr. 16	Zürich	Zürich
79.206	Huber A.	Bahnhofstr. 20	Winterthur	Zürich
90.101	Cotti B.	Via Almoli 3	Lugano	Ticino
88.001	Boudry C.	Rue du Soleil 1	Crans	Valais

Wenn wir nun Person `INTERSECT` Person eingeben, entsteht eine Database, welche die Einträge aufführt, welche in beiden Databasen vorhanden ist, analog zu $(A \cap B)$ in der Mengenlehre. So würden wir bei Person B `INTERSECT` Person A folgende Tabelle erhalten:

ID Number	Name	Address	Town	Canton
52.137	Wild K.H.	Ochsenstr. 16	Zürich	Zürich
90.101	Cotti B.	Via Almoli 3	Lugano	Ticino

5.3 Union

Der `UNION` Befehl vereint zwei Datenbanken und führt doppelte Einträge nur einmal auf. Man kann auch einzelne Beiträge hinzufügen. So erhalten wir folgendes Ergebnis, wenn wir mit den Befehl *Person A* `UNION` *(69.420, Jürss E., Timstr. 1, Zürich, Zürich)* nutzen.

ID Number	Name	Address	Town	Canton
52.137	Wild K.H.	Ochsenstr. 16	Zürich	Zürich
61.100	Scherrer P.	Salzerstr. 20	Biel	Bern
90.101	Cotti B.	Via Almoli 3	Lugano	Ticino
91.200	Huber A.	Bahnhofstr. 20	Basel	Basel-Stadt
69.420	Jürss E.	Timstr. 1	Zürich	Zürich

5.4 Select

Der `SELECT` Befehl macht was er sagt, man kann mit ihm Einträge die eine bestimmte Bedingung erfüllen heraussortieren. Der Syntax lautet

`SELECT FROM` *[database name]* `WHERE` *[condition]* `GIVING` *[name of new database].*

Als Beispiel nutzen wir den Befehl `SELECT FROM` *Person B* `WHERE` *Canton=Zürich* `GIVING` *[Person X].* Somit erhalten wir die Database Person X:

ID Number	Name	Address	Town	Canton
52.137	Wild K.H.	Ochsenstr. 16	Zürich	Zürich
79.206	Huber A.	Bahnhofstr. 20	Winterthur	Zürich

5.5 Project

`PROJECT` ist ein Befehl, der erlaubt, eine Tabelle neu zu sortieren nach beliebigen Kriterien. Der Syntax dieses Befehls lautet

`PROJECT` *[database name]* `OVER` *[attributes]* `GIVING` *[name of new database].*

Als Beispiel nehmen wir den Befehl `PROJECT` *Person A* `OVER` *Canton, Town, Name* `GIVING` *Person D*

Canton	Town	Name
Zürich	Zürich	Wild K.H.
Bern	Biel	Scherrer P.
Ticino	Lugano	Cotti B.
Basel-Stadt	Basel	Huber A.

5.6 Join

Der `JOIN` Befehl kombiniert zwei Databases anhand von einer in beiden einzelnen Tabellen enthaltenen Kategorie und gibt nur die Einträge in eine neue database, welche in dieser Kategorie matchen, alle anderen Einträge werden vernachlässigt. Der Befehl hat den folgenden Syntax:

`JOIN` *[database name 1][database name 2]* `EQUATE` *[attribute 1][attribute 2]* `GIVING` *[name of new database].*

Als Beispiel nutzen wir die folgenden beiden Tabellen:

Person A

ID Number	Name	Address	Town	Canton
52.137	Wild K.H.	Ochsenstr. 16	Zürich	Zürich
61.100	Scherrer P.	Salzerstr. 20	Biel	Bern
90.101	Cotti B.	Via Almoli 3	Lugano	Ticino
91.200	Huber A.	Bahnhofstr. 20	Basel	Basel-Stadt

Borrowed

ID Number	Book number	Days borrowed
52.137	10.892	21
79.206	3.478	17
91.200	12.123	3

mit folgendem Befehl: `JOIN` *Person A, Borrowed Books* `EQUATE` *ID Number, ID Number* `GIVING` *Person E.* Das resultiert in der folgenden Tabelle

ID Number	Name	Address	Town	Canton	Book number	Days borrowed
52.137	Wild K.H.	Ochsenstr. 16	Zürich	Zürich	10.892	21
91.200	Huber A.	Bahnhofstr. 20	Basel	Basel-Stadt	12.123	3

6 Begrifflichkeiten

In den meisten Prüfungen werden Begrifflichkeiten abgefragt, deshalb sind hier alle Begriffe aus vorherigen Prüfungen aufgeführt (ab 2015). Auch hierfür existiert ein Karteikartenset, es ist unter diesem Link zu finden.

6.1 Computerstruktur

Begriff	Bedeutung
`Latch`	Electronic circuit with two NOR gates, stores 1 bit of information
`Register`	Small element of memory integrated in the CPU
`Flag`	1 bit register
`Cache`	Fast access memory, connected to processor
`Bus`	Electronic (physical) connection between different elements of a computer
`Coprocessor`	Second processor next to the CPU for spezialized operations (e.g. FPU for Floating Point operations, Sound chips for audio rendering)
`Core Voltage`	Voltage that the cpu operates at
`Pipelining`	Ability to perform several basic steps in a staggered way in a single clock cycle
`Vectorization`	Ability to perform operations on vector operands and not only scalar operands
`Parity bit`	Additional bit added to string that indicates if the number is even or odd
`Clock cycle`	Time between to single electronic pulses of a CPU, therefore the time that a CPU needs between two basic operations
`Bit`	Repräsentiert eine Stelle, 8 bits sind ein byte.
`Moore's Law`	States that the power of CPU increases by a factor of 10 every 6 years

6.2 Algorithms

Begriff	Bedeutung
`Brute-force`	Tries all possible solutions
`Greedy`	Selecting always the option giving the largest immediate progress towards the goal
`Divide & Conquer`	Splits problem in two subproblems, solves and combines them
`Backtracking & Prunning`	Attempts to sort the solutions in a tree and then goes thorugh it and scipping trees with a worse solution than the current one

6.3 Network

Begriff	Bedeutung
`Computer Network`	Interconnected collection of autonomous computer
`Distributed System`	Set of slave computers subordinated to a master computer
`All-to-All Communication`	Message sent to all computers and they decide themselves if its important
`Point-to-Point Communication`	Message sent via a fixed connection between sender and receiver
`Layer`	7 different levels of increasing data complexity [in eckigen Klammern] (physical [bits], data link[frames], network [packets], transport [segments], session [data], presentation [data] and application[data])
`Interface`	Transforms Data between different layers
`Protocol`	Conventions which are used for Transmission of data in a network between two computers at a layer
`Gateway`	Computer that connects different networks, can connect to any layer
`Shared memory`	Different processors operate on one memory
`Distributed Memory`	Different processors operate individual memories

6.4 C++

Begriff	Bedeutung
`Declaring a variable`	Creating a variable but not assigning it a value
`Initializing a variable`	An already declared variable is filled with a value for the first time
`Passed by value`	If a variable is changed in a function it isn't corrupted outside of the function
`Passed by reference`	If a variable is changed in a function it is corrupted outside of the function

6.5 Molecular Simulation

Begriff	Bedeutung
`Quantum Simulation`	Uses electrons and nuclei and quantum mechanics to define a problem
`Classical Simulation`	Uses atoms and classical mechanics to define a problem
`Mesoscopic Simulation`	Uses volume elements and variables to define a problem
`Force Field`	Potential Energy function between molecules (van der Waals and electrostatic) and in molecules (covalent)
`Molecular dynamics Simulation`	Performed by numerically integrating Newtons equation of motion in time using a classical force field
`Periodic boundary conditions`	Model the outside of a finite simulated system by replicating the system infinite times

7 C++

7.1 Variablen

Variablen bilden die Grundbasis für ein C++ Code. Man declaredsie indem man die Art der Variable schreibt und dahinter den Namen welchen man wählen möchte: `int x;`. Den Wert der Variable kann man wie folgt festlegen `x = 3;`. Natürlich geht auch beides in einem Schritt `int y = 4;`.

7.1.1 Arten von Variablen

Es gibt verschiedene Arten von Variablen für verschiedene Zwecke:

Name	Input	Größe
`int`	Ganze Zahlen	4 Byte
`long`	Ganze Zahlen	8 Byte
`float`	Dezimalzahlen	4 Byte
`double`	Dezimalzahlen	8 Byte
`bool`	true/false	1 Byte
`char`	ein Zeichen	1 Byte
`string`	Text	variabel

7.1.2 Umwandeln von Variablen

Wenn Variablen in andere Datentypen umgeschrieben werden oder falsch angegeben sind (`int x = 3.5;`) werden sie automatisch umgewandelt. Teilweise kann man dies nutzen wie zum Beispiel um mit einer Rechnung eine Bool Variable zu füllen. Im Folgenden werden die wichtigen Umwandlungen mit jeweils einem Beispiel in Klammern aufgezeigt.

- `int` (1) zu `double` (1.0)

- `double` (5.8) zu `int` (5), es wird immer abgerundet
- `bool` zu `int`, false ergibt 0, true ergibt 1
- `int` zu `bool`, alles außer 0 wird true

7.2 Operatoren

Operatoren werden in C++ genutzt um Rechnungen, Vergleiche und andere Abläufe durchzuführen. Im Folgenden werden die wichtigen Operatoren aufgezählt und dargestellt. Die meisten sind sehr intuitiv, also keine Sorge.

7.2.1 Arithmetisch

Symbol	Name	Nutzen	Beispiel	Result
+	Addition	Obvious	3 + 6	9
−	Subtraktion	Obvious	7 - 4	3
∗	Multiplikation	Obvious	2 * 3	6
/	Division	Obvious	8 / 2	4
%	Modulo	Gibt den Rest einer `int`-Division	7 % 2	1
++	Increment	Erhöht um 1	-	-
−−	Decrement	Reduziert um 1	-	-

Bei increment und decrement ist zu beachten, dass die Platzierung wichtig ist. Ist der Operator vor der Variablen (`++x`) so wird die Variable direkt verändert (hier erhöht), ist er dahinter (`x++`) so wird die Variable erst nach der Ausführung des Commands verändert.

7.2.2 Zuweisung

Symbol	Name	Nutzen	Beispiel	Result
=	Zuweisung	Obvious	`int x = 5;`	5
+ =	Erhöhung	Obvious	x += 5;	10
− =	Reduktion	Obvious	x -= 5;	5
∗ =	Multiplikation	Obvious	x *= 5;	25
/ =	Division	Obvious	x /= 5;	5
% =	Modulo	Obvious	x %= 2;	1

7.2.3 Vergleich

Vergleichsoperatoren werden genutzt um zwei Variablen zu vergleichen, sie werden meistens in logischen Funktionen (siehe nächstes Kapitel) genutzt um die Grenzen festzulegen.

Symbol	Name	Nutzen	Beispiel	Result
==	Gleich?	Obvious	`bool x = 2 == 2;`	true
!=	Ungleich?	Obvious	x != 2;	false
>	Größer?	Obvious	x > 2;	false
>=	Größer/Gleich?	Obvious	x >= 2 ;	true
<	Kleiner?	Obvious	x < 2;	false
<=	Kleiner/Gleich?	Obvious	x <= 2;	true

7.2.4 Logisch

Symbol	Name	Nutzen
&&	Logisches Und	True wenn beide Werte true
||	Logisches Oder	True wenn mind. einer der Werte true ist
!	Logisches Not	kehrt das Wahrheitsstatement um

&& and || haben jeweils Short Circuit Properties, dass heißt sie können schon einen Wert weitergeben obwohl sie erst eine Bedingung gesehen haben. Bei dem Logischen Und wäre dies der Fall, wenn die erste Kondition falsch wäre (Ganze Kondition falsch), bei dem Logischen Oder ist dies der Fall wenn die erste Kondition richtig wäre (Ganze Kondition Richtig).

7.2.5 Sonstige Operatoren

Hier führe ich nun noch zwei weitere Operatoren auf, zum einen den Reference Operator (). Wird dieser in der Deklaration einer Funktion genutzt `int &x`, so wird der Wert, der der Variablen eigentlich nur für die Funktion gegeben wurde, auch außerhalb dieser Funktion gegeben.

```
1  void swapNums(int &x, int &y) {
2    int z = x;
3    x = y;
4    y = z;
5  }
6
7  int main() {
8    int firstNum = 10;
9    int secondNum = 20;
10
11   cout << "Before swap: " << "\n";
12   cout << firstNum << secondNum << "\n";
13
14   // Call the function, which will change the values of firstNum and secondNum
15   swapNums(firstNum, secondNum);
16
17   cout << "After swap: " << "\n";
18   cout << firstNum << secondNum << "\n";
19
```

```
20    return 0;
21 }
```

Der Syntax und die Struktur dieser Funktion wird später erklärt, wichtig ist hier nur, dass als erstes 10 und 20 ausgegeben wird und als zweites 20 und 10, da die Werte aus der Funktion beibehalten werden.

Keine Sorge, der letzte Operator ist etwas leichter zu verstehen, er nennt sich Ternary Operator und wird verwendet um eine Zuweisung abhängig zu machen. So ist der Syntax dieses Operators:

x = [Bedingung die true oder false ausgibt] **?** [Ausgabe wenn **true**] : [Ausgabe wenn **false**]

```
int x;
x = (7 < 9) ? 4 : 2;
cout << x << endl;
x = (7 == 9) ? 4 : 2;
cout << x << endl;
```

Hier gibt die erste Zeile 4 aus und die zweite Zeile gibt 2 aus.

7.3 Logik

Nun steigen wir in das Herz der Funktionen ein, die Logik. Es gibt drei Funktionstypen die wir nutzen.

7.3.1 If

Eine If-Funktion nimmt eine `bool` an und führt dann den entsprechenden Code, welcher in geschweiften Klammern steht, aus. Man kann diese Funktion auch erweitern mit einem `else if`, welches eine weitere Kondition checkt, wenn die erste falsch wird. Zusätzlich kann man ein `else` am Ende hinzufügen, welches eine alternative Funktion ausführt, wenn die vorherigen `If`-Statements alle falsch waren.

```
int x = rand() % 3 - 1; // random number from -1 to 1

if(x < 0) {
   cout << "x ist -1" << endl;
} else if (x == 0) {
   cout << "x ist 0" << endl;
} else {
   cout << "x ist 1" << endl;
}
```

7.3.2 While

Eine `while` Funktion wird solange ausgeführt, solange die `bool` in der Klammer wahr bleibt. Somit ist auch wichtig, dass die Variable, welche die `bool` bestimmt, nach jeder Ausführung abgeändert wird. Dabei gibt es folgende drei Möglichkeiten.

Zum Einen kann man die Abänderung in die Funktion selber schreiben.

```
int x = 0;

while(x < 10) {
   cout << x << endl;
```

```
5   x++;
6 }
```

Sonst kann man diese aber auch direkt in die `bool` des while loops schreiben. Hier wird auch noch einmal der Unterschied von $++\mathbf{x}$ und $\mathbf{x}++$ klar.

```
1 int x = 0;
2
3 while(x++ < 10)
4   cout << x << endl; // druckt die Zahlen von 1 bis 10
5
6 x = 0;
7
8 while(++x < 10)
9   cout << x << endl; // druckt die Zahlen von 1 bis 9
```

Wenn unsere Kondition auf 0 zugehen soll (`while`$(x > 0)$) kann man auch einen short-cut nutzen.

```
1 int x = 10;
2
3 while(--x)
4   cout << x << endl; // druckt die Zahlen von 9 bis 1
```

7.3.3 for

Ein `for`-Loop ist eine Erweiterung des `while`-Loops und ist mit folgendem Syntax aufgebaut:.

```
1 for([Initialisierung]; [Kondition]; [Repetition]) {
2   cout << i << endl;
3 }
```

Die Intialisierung wird am Anfang des Loops ausgeführt, die Kondition wird nach jeder Wiederholung überprüft und sollte sie falsch sein, wird der Loop beendet. Das letzte Argument wird am Ende jedes Durchgangs ausgeführt. Hier ein kleines Beispiel:

```
1 for(int i = 0; i < 10; i++) {
2   cout << i << endl; // prints numbers from 0 to 9
3 }
```

7.4 Funktionsstruktur

Wir wollen die Funktionsstruktur anhand eines Beispiels betrachten.

```
1 double function1(double x) {
2     double sum = 1.0;
3     for(int i = 1; i < 10; i++) {
4         sum += i * x;
5     }
6     return sum;
7 }
```

- `double` indiziert, welchen Datentyp die Funktion zurückgeben soll. Falls nichts zurückgegeben werden soll, wird `void` verwendet (Häufig um Variablen mithilfe von Funktionen zu ändern).

- `function1` ist der Name der Funktion.
- `double x` ist die Input Variable (*Argument*), mehrere Argumente werden mit Kommata separiert.
- `return sum` gibt den Output zurück, hier die Variable `sum`. Wenn es keinen Output geben soll, wie zum Beispiel bei einer `void` Funktion, schreibt man einfach `return;`.

Wenn die Funktion in `int main()` genutzt wird, wird ein sog. Prototype genutzt welcher vor der `main` Funktion steht und der ersten Zeile der Funktion entspricht, nur abgeschlossen mit einem Semikolon (oder Punktstrich für komische Österreicher, in der Beispiel Funktion wäre das:

```
1 double function1(double x);
```

Um diese Funktion nun in der `int main()` Funktion zu implementieren, called man diese Funktion. Dabei muss gibt man die Input Variablen (Argumente), welche genutzt werden sollen, in Klammern geschrieben, mit Kommata separiert an. Die Argumente können entweder als Variablen, welche in `main` definiert sind, oder als absolute Zahlen angegeben werden. So würden wir unsere Beispiel Funktion in `main` wie folgt callen:

```
1 int main() {
2 double x = 3.6;
3 double y;
4
5 y = function1(x);
6
7 cout << y << endl;
8 }
```

7.5 iostream

`iostream` ist eine Bibliothek aus der zwei Befehle häufig genutzt werden, `cin`, für inputs und `cout` für outputs. Mit `cin` können Variablen vom Nutzer gefüllt werden und mit `cout` werden Ergebnisse oder Textzeilen ausgegeben.

```
1 int semester;
2
3 cout << "Bitte geben sie ihr Semester ein: ";
4 cin >> semester;
5 cout << "Der Student ist im " << semester << ". Semester. << endl;
```

Wie man hier sieht, muss jeder Zeile mit einem Semicolon abgeschlossen werden, `cout` Argumente werden mit « abgetrennt und `cin` mit ». Richtiger Text muss in Anführungszeichen gesetzt werden und am Ende einer `cout` Zeile steht meistens ein `endl` um einen Zeilenumbruch zu machen.

7.6 Arrays

Arrays werden genutzt um Zahlen oder Zeichen in einer Kette zu lagern. Dabei gibt es auch zwei Dimensionale Arrays, welche man sich wie Matrizen vorstellen kann. Hierbei beginnt der Index sowohl in einem 1D als auch in einem 2D Array immer mit 0, das heißt die erste Zeile und Spalte starten bei 0 und der letzte Eintrag ist der $(N-1)$. Eintrag.

Bei der Initialisierung von Arrays ist es wichtig, den Variablentyp der eingefügt wird vorweg zu schreiben. Wenn man das Array nicht gleich bei der Initialisierung befüllt, kann man auch die Array Größe direkt angeben. Folgende Beispiele machen das hoffentlich klarer.

```
double Array1[5];
Array1[2] = 1.5; // das 3. Element wird auf 1.5 gesetzt

char Array2[] = {'B', 'u', 'r', 'n', 'o', 'u', 't', 'v', 'o', 'r',
                 'p', 'r', 'o', g', 'r', 'a', 'm', 'm', 'i', 'e', 'r', 't'};
                 // Ist ein 22 Zeichen langes Array.

int 2DArray[][] = {{1, 2, 3}, {4, 5, 6}};
cout << 2DArray[1][0] << " and " << Array2[21] << endl; // Ergibt: "4 and t".
```

Um zwei Einträge zu tauschen muss man eine Dummy Variable einführen, welche als Zwischenspeicher fungiert. So können wir im Vorherigen Array wie folgt tauschen.

```
char dummy = Array2[3];
Array2[3] = Array2[21];
Array2[21] = dummy;

cout << Array2[3] << " and " << Array2[21] << endl; // Ergibt: "t and n".
```

Arrays nutzen ein Sentinel Element, welches ein in das Array eingefügten Wert darstellt, welcher dem zu Suchenden Element entspricht. Daher wird ein Suchalgorithmus immer eine Antwort funktionieren, selbst wenn der Wert gar nicht im Array enthalten ist.

7.7 Fehler

In der Prüfung werden häufig Codes gegeben welche auf Fehler geprüft werden sollen. Diese Aufgaben sind ziemlich lang und es ist schwer alle Fehler zu finde, aber ein paar Punkte sollten leicht verdient sein.

7.7.1 Syntax Fehler

Syntaktische Fehler tauchen auf wenn der Compiler den Code nicht verarbeiten kann, weil zum Beispiel ein Zeichen fehlt oder ein Semikolon anstatt eines Kommas steht. Weitere Beispiele sind eine nicht deklarierte Variable, eine fehlende geschweifte Klammer, ein fehlendes `return`-Statement (Wobei bei `void` Funktion kein `return`-Statement benötigt ist) etc..

7.7.2 Semantische Fehler

Semantische Fehler erkennt der Compiler nicht als falsch an, sondern sind eher mathematischer Natur. So zählt dazu zum Beispiel das Teilen durch 0, sonstige funktionsspezifische Fehler (z.B. negativer ln), ein Loop der nie endet oder ein Eintrag in einem Array, welches nicht existiert.

7.7.3 Logische Fehler

Logische Fehler sind nicht auf den Code zurückzuführen sondern beziehen sich eher auf die Aufgabe selbst. So zählen dazu falsche Einheiten, ein falscher Algorithmus oder eine falsche Formel/Berechnung.

8 Algorithmen

Algorithmen geben nicht wirklich neuen Stoff, sie sind nur eine Sammlung an Lösungsansätzen, welche hilfreich für spezifische Probleme sind. Die meisten dieser Ansätze nehme ich aus dem PVK Skript von Alexander Schoch, kürze sie aber etwas ab. So ist dies eine Sammlung von vorgeschriebenen Kochrezepten, rein theoretisch wenn man sehr gut programmieren kann, kann man sie sich aber auch selber schreiben in der Prüfung.

8.1 Iteration

In diesem Beispiel wird die Factorial (n!) Funktion mit einer iterative Methode gelöst. Sie ist eher intuitiv.

```
int factorial(int n) {
  int product = 1
  for(int i = 1; i <= n; i++)
    product *= i;
  return product;
}
```

8.2 Rekursion

Wir benutzen erneut die Factorial Funktion, um den rekursiven Ansatz anzugucken.

```
int factorial(int n) {
  if(n <= 1)
    return 1;
  return n * factorial(n-1);
}
```

Wie man sieht, wird hier der genutzt das man die Factorial Funktion so umschreiben kann $n! = (n-1)! \cdot n$. Die Funktion nutzt hierbei das `return`-Statement um die Formel zu lösen. Am Ende des Tages ist es aber Geschmackssache, was man nutzt.

8.3 Suchalgorithmen

8.3.1 Sequential Search

Die einfachste Form eines Suchalgorithmus ist der sequentielle Ansatz, der das Array einfach durchgeht und die Einträge abgleicht.

```
int sequentialSearch (int n[], int N, int goal) {
    for (int i = 0; i< N ; i++) {
        if (n[i] == goal)
        return i;
    }
    return -1;
}
```

Wenn man diesen Prozess etwas optimieren möchte kann man auch folgenden Code verwenden:

```
int sequentialSearch (int n[], int N, int goal) {
    n[N] = goal;
    int i = 0;
```

```
    while (n[i] != goal){
    i++;
    }
    return i;
}
```

So würde, wenn der gesuchte Eintrag schon bei den ersten paar Stellen steht, enorm viele Wiederholungen des Loops gespart werden.

8.3.2 Binary Search

Binary Search kann nur in einem sortierten Array genutzt werden. Sie fängt in der Mitte der Liste an und guckt ob der gesuchte Eintrag größer oder kleiner ist. Somit wird der Suchbereich mit jeder Wiederholung halbiert, was effektiv sein kann.

```
int binarySearch (int n[], int N, int goal) {
    int left = 0, right = N - 1, middle;
    while (left <= right){
        middle = (right + left) / 2;
        if (goal < n[middle])
            right = middle - 1;
        else if (goal > n[middle])
            left = middle + 1;
        else
            return middle;
    }
    return -1;
}
```

8.4 Sortieralgorithmus

8.4.1 Selection Sort

Selection Sort sucht sich als erstes das kleinste Element und sortiert dieses dann ganz am Anfang ein. Dann sucht es sich den nächstkleineren Eintrag und sortiert diesen über dem kleinsten Eintrag ein usw..

```
void selectionSort (int n[], int N) {

    // repeat the sorting as long as there are unsorted elements
    for (int i = 0; i < N; i++){
        //finding the smallest Element sE
        int sE = i;
        for (int j = i; j < N; j++) {
            if (n[j] < n[sE])
                sE = j;
        }
    // Now swap the sE with the first element of the unsorted part
    int dummy = n[i];
    n[i] = n[sE];
    n[sE] = dummy;
 }
 return;
```

```
}
```

8.4.2 Insertion Sort

In der Insertion Sort vergleicht man jeweils zwei Elemente und tauscht sie, falls ihre Reihenfolge falsch ist. So geht man dann für die Zahl bis sie am richtigen Platz ist und macht dann mit der nächsten Zahl weiter.

```
void selectionSort (int n[], int N) {

    // repeat the sorting as long as there are unsorted elements
    for (int i = 0; i < N; i++){
        // start at the initial position of this element
        int j = i;
        while (j > 0 && n[j] < n[j-1]) {
            // exchange the elements
            int dummy = n[j];
            n[j] = n[j-1];
            n[j-1] = dummy;
            // reconfigure the position one down so
            // we stay at our current number
            j--;
        }
    }
    return;
}
```

8.5 Integrationsalgorithmen

Im folgenden führe ich nun kurz zwei Integrationsalgorithmen auf. Die Integration mit Rechtecken und Trapezen. Hierbei ist die Funktion mit `func (double x)` gegeben und gibt den Funktionswert $f(x)$ aus. Weitere Inputs sind die Grenzen a und b, sowie die Anzahl an Einheiten die zur Berechnung genutzt werden N. Somit wird die Integration mit steigendem N immer genauer.

8.5.1 Rectangular Integration

```
double RechteckInt (double a, double b, int N) {
    double sum = 0; // setzt die Integrationssumme initial auf 0
    double h = (b - a) / N; // berechnet die Länge der einzelnen
                            // Integrationsabschnitte
    for (int i = 0; i < N; i++) {
        sum += h * func (a + (i + 0.5) * h);
        // Die Länge in x ist h, die Länge in y wird mit dem
        // Funktionswert in der Mitte des x Abschnittes ermittelt
        }
    return sum;
}
```

8.5.2 Trapez Integration

Wir nutzen folgende Formel für die Fläche eines Trapez

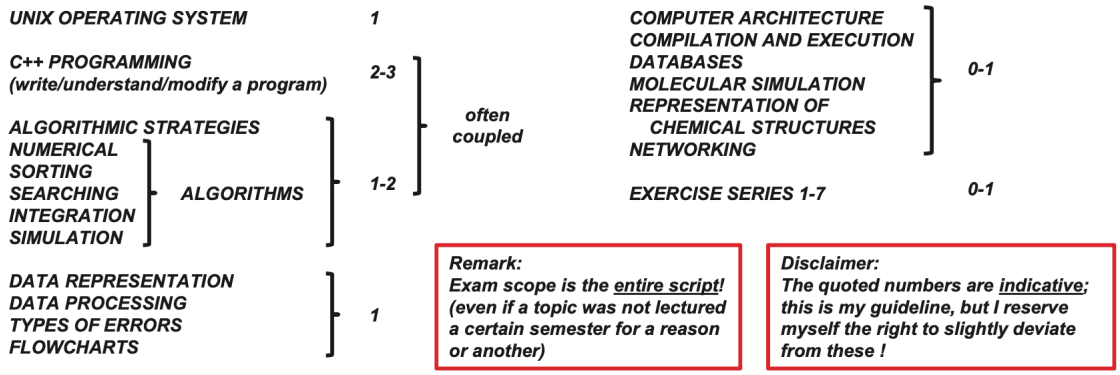
$$A_{Trapez} = \frac{1}{2} \cdot (a + c) \cdot h$$

wobei hier h für die Grundfläche steht, a für die Höhe am linken Rand des Trapezes und c für die Höhe am rechten Rand des Trapez.

```
double RechteckInt (double a, double b, int N) {
    double sum = 0; // setzt die Integrationssumme initial auf 0
    double h = (b - a) / N; // berechnet die Länge der einzelnen
                            // Integrationsabschnitte
    for (int i = 0; i < N; i++) {
        // Die Funktion wird oben erklärt
        sum += h * (func (a + (i   * h)) + func (a + (i + 1) * h)) / 2;
        }
    return sum;
}
```

9 Prüfungsstruktur

Die Prüfung besteht (normalerweise) aus 6 Aufgaben, welche immer einen ähnlichen Aufbau haben. Prof. Hünenberger gibt folgendes Diagramm an für die Struktur:



9.1 Aufgabe 1: UNIX

Die UNIX Aufgabe wird im UNIX Teil ausführlich beschrieben und sind eigentlich einfache Punkte mit genug Übung.

9.2 Aufgabe 2: Data reprenstation and processing

9.2.1 Zahlenformate

In dem ersten Teil dieser Aufgabe wird häufig die Umwandlung von binären zu oktalen oder hexadecimalen Formaten abgefragt oder anders herum. Siehe hierzu das Kapitel Zahlensysteme.

9.2.2 Diverses

Hier wird häufig eine Aufgabe genutzt welche auch Zahlen und Textformaten gewidmet ist, jedoch sind diese nicht bekannt und müssen selbstständig erschlossen werde.

9.2.3 Logische Gitter

So gut wie immer ist ein logisches Gitter der dritte Teil der Aufgabe, auch das sollten leicht verdiente Punkte sein. Siehe hierzu das Kapitel Logische Gitter.

9.3 Aufgabe 3: Algorithms and programming I

Auch diese Aufgabe ist in Teilaufgaben unterteilt. Der erste Teil ist immer ein von einem schlechten Studenten geschriebener Code. Hierbei hilft die Sektion Fehler.

Manchmal folgt dann eine Aufgabe wo man den Sinn bzw. die Funktion eines Codes bestimmen muss.

Häufig kommt auch eine Abfrage von Begrifflichkeiten zu verschiedenen Themen dran, hierzu habe ich alle bisher in Prüfungen vorkommenden Begriffe im Kapitel Begrifflichkeiten zusammengetragen. Hoffen wir mal dass Prof. Hünenberger etwas faul ist.

9.4 Aufgabe 4 - 6: Algorithms and programming others

In den letzten drei Aufgaben müssen häufig 2 - 3 Funktionen geschrieben werden. Hierbei kommen zum einen mathematische Funktionen vor, zum anderen aber auch Chemie related stuff, hier hilft es definitiv die Algorithmen ungefähr zu kennen, zum Teil kommen auch Aufgaben aus den Serien ran.