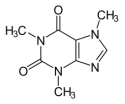
Einführung in die C++-Programmierung

Progammierkurs an der Uni Konstanz, SS2018

Übungen

Alle Aufgaben sind mit den bisher gelernten Konstrukten lösbar. Natürlich dürft ihr euch zusammensetzen, und eure Lösungsansätze diskutieren. Damit ihr mit der Syntax vertraut werdet, solltet ihr das Programm aber trotzdem (zumindest die ersten 15 Aufgaben) alleine schreiben und einzeln abgeben. Benennt eure Dateien bitte wie jeweils bei der Aufgabe angegeben und verwendet in den Dateinamen keine Leer und Sonderzeichen, mit Ausnahme von Unterstrichen. Der Quelltext und die Kommentare sollten grundsätzlich auf Englisch geschrieben werden. Besonders schwere Aufgaben werden von wilden Tieren bewacht und können erstmal übersprungen werden. Die Einnahme leistungssteigender Psychopharmaka ist gestattet. Um ein Auslösen der Rauchmelder zu vermeiden, wird von zu allzu verbissenem Nachdenken abgeraten.
Viel Spaß!



Konsolen-Ausgabe, Eingabe, einfache Arithmetik

0. Mache dich mit den Befehlen der Kommandozeile, die wir heute gelernt haben, soweit vertraut, dass du sie benutzen kannst. Für den Editor sollte eine Zusammenfassung der Shortcuts in einem Ordner in eurem home-Verzeichnis auf den virtuellen Systemen zu finden sein.

Schreibe ein Programm,
1. das zwei ganze Zahlen einliest, Summe, Differenz, Produkt, Quotient und den Rest bei der Division berechnet und ausgibt.

```
("Ex1_YourName.cpp")
```

2. Variablenwerte von zwei Zahlen miteinander vertauscht, so dass n das ist, was m vorher war (und umgekehrt).

```
("Ex2_YourName.cpp")
```

3. das Temperaturen von °C in K und °F umrechnet.

```
("Ex3_YourName.cpp")
```

4. das aus Volumen, Konzentration und Molekulargewicht berechnet, wieviel du auf der Waage einwiegen musst, um eine Lösung dieses Volumens und mit dieser Konzentration herzustellen.

```
("Ex4_YourName.cpp")
```

5. das die Wellenlänge für einen Energieübergang $n \to m$ bei Wasserstoff mithilfe der Rydberg-Formel berechnet (Bohrsches Atommodell). Zur Erinnerung:

$$\frac{1}{\lambda} = R\left(\frac{1}{m^2} - \frac{1}{n^2}\right), R = 10967758,410 m^{-1}$$

Hinweis: Benutze für n und m den Datentyp `double`*, sonst wird es schwierig, die Brüche zu berechnen. Da wir noch keine Exponential kennen (und im vorliegenden Fall auch nicht verwenden würden), gilt* $m^2 = m \cdot m$*!*

`("Ex5_YourName.cpp")`

6. N hungrige Exemplare von *Paramecium caudatum* sind zufällig in einer Kultur mit *S. cerevisia*, die aus 10^{12} Zellen besteht, gelandet. Angenommen, jedes Paramecium frisst 15 Hefezellen in der Stunde, und jede nicht gefressene Zelle teilt sich alle 2 Stunden. Wieviele Hefezellen sind in Abhängigkeit von einem N nach 8 Stunden noch übrig? Was ist bei der Wahl der Datentypen für die Größe der Hefekultur und die Zahl der Paramecien zu beachten? (Für die fortgeschrittenen Biologen und Life Scientologen: Wir nehmen zusätzlich an, dass unsere Hefe sich ausschließlich asexuell durch Sprossung vermehrt und kein a/α-Mating stattfindet).

`("Ex6_YourName.cpp")`

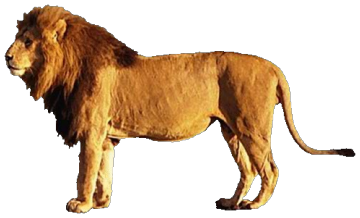
if-Abzweigungen, Zahlensysteme

7. Schreibe ein Programm, das 3D-Koordinaten (`double x`, `double y`, `double z`) darauf testet, in welchem Oktanten eines euklidischen Koordinatensystems sie liegen (Wenn das zu verwirrend ist, dann versuche erstmal ein 2-dimensionales Koordinatensystem mit 4 Quadranten).

`("Ex7_YourName.cpp")`

8. Entwickle einen Konverter, um das Bitmuster eines `unsigned char` auszugeben. Ob die bits vorwärts oder rückwärts ausgegeben werden, ist egal, vorwärts ist etwas schwieriger, ohne weitere Sprachkonstrukte zu verwenden.
(Hinweis: Mit einem `unsigned char` kann mit den Operatorengerechnet werden, als ob es sich dabei um ein 8-bit integer handelt.)

`("Ex8_YourName.cpp")`



9. Entscheide, ob eine Kugeloberfläche mit Radius r und Mittelpunkt (m_x, m_y, m_z), sowie eine Gerade mit offset- (o_x, o_y, o_z) und Richtungsvektor (d_x, d_y, d_z) Schnittpunkte haben und berechne sie gegebenenfalls.

`("Ex9_YourName.cpp")`

Schleifen, Gültigkeitsbereiche

10. Schreibe ein Programm, das ungerade Zahlen von 1 bis N ausgibt.

`("Ex10_YourName.cpp")`

11. Schreibe ein Programm das entscheidet, ob eine Zahl eine Primzahl ist.

`("Ex11_YourName.cpp")`

12. Wieviel Durchläufe haben jeweils die folgenden Schleifen?

```
int n = 100;
for(int i = 0 ; i < n; i = i + 1){
  // any source code
}

for(int i = 1 ; i <= n; i = i + 1){
  // any source code
}

for(int i = 0 ; i <= n; i = i + 1){
  // any source code
}

for(int i = 1 ; i < n; i = i + 1){
  // any source code
}

for(int i = 1 ; i < n; i = i + 1){
 i = i+1;
}

for(int i = 0 ; i < n; i = i + 1){
  if(i < 50)
    {
      i = i+1;
    }
}

for(int i = 0 ; i < n; i = i + 1){
  if(i < 51)
    {
      i = i+1;
    }
}

for(int i = 0 ; i < n; i = i + 1){
  if( (i % 3) == 0 )
    {
      i = i+1;
    }
}
```

Markiere in folgendem (sinnlosen) Programm an jeder {-Klammer, welche Variablen im Bereich jeweils gültig sind.

```
#include <iostream>
using namespace std;

int main(){
  int a;
  double x,y;
  {
    unsigned long long int b;
    int cat =  b % a;
    if(cat < b)
      {
        double z = x + y * b;
        cout <<  z;
      }
    else if(cat < a)
      { // cat, b, a, x, y
        char chemistry = 'c';
        bool badScientist;
        bool bioHazard = (badScientist = true);
        if(bioHazard && badScientist)
          {
            long long int bigExplosion = 50000000000;
            x += y;
            cout << x;
          }
        else
          {
            x = 6;
            double captainPicard = cat;
            /*
            {
            Influence?
            }
            */
          }
        bool unlucky = false;
      }
    else
      {
        a = (int) x - y;
      }
  }
}
```

Funktionen, Arrays

13. Nutze das kommende schöne Wetter und mache dich mit den Köstlichkeiten boarischer Biergartenkultur vertraut.

14. Schreibe ein Programm das alle Zahlen von 1 bis N aufzählt, aber

- für jede durch 7 teilbare Zahl „Brezn “
- für jede Zahl, die die Ziffer 7 (Dezimalsystem) enthält, „Obazda“
- für jede Schnapszahl „Bier “

statt der Zahl als Ausgabe liefert. Wenn mehrere der Bedingungen auftreten, sollten alle Ersatzwörter in der Ausgabe auftauchen. Implementiere für die Überprüfung folgende Funktionen:

```
bool isDivisableByN(int val, int n);
bool containsCipherN(int val, int n);
bool isRepdigit(int val);
```

Diese können dann folgendermaßen im Programm verwendet werden:

```
if(isDivisableByN(i, j) {
   //...
}
else {
   //...
}
```

Spoiler:

Bier kann mithilfe von abschneidenden Integerdivisionen durch 10 und mod(n, 10) gebraut werden

```
("Ex14_YourName.cpp")
```

15. Schreibe ein Programm, das auf einem double-Array $f(t)$ der Länge 1000 und mit der Zeiteinheit 1 ms, und den Werten (1.0, 2.0, 3.0, ... , 1000.0), die (reelle) Sinustransformation $S(\omega)$ von 1 - 500 kHz mit einer Abtastrate von 100 MHz berechnet:

$$S(\omega)) = \mathcal{SIN}(f(t)) = \sum_{i=0}^{999} f(t) \sin(2\pi\omega t_i)$$

Hinweis: Die Sinusfunktion ist in <`cmath`> *deklariert:*

```
double sinOfX = sin(x);
```

Warum wäre es falsch, einen größeren Frequenzbereich zu wählen? (Keine Programmieraufgabe ☺).

`("Ex15_YourName.cpp")`

16. Nehmen wir an, dass in einem 2D-Koordinatensystem, das nur ganzzahlige, positive Koordinaten kennt, weiße Punkte mit „true “und hellblaue Punkte mit „false“markiert werden. Baue ein solches Koordinatensystem mit der Länge 101 nach, färbe alle Punkte weiß und setze dann in die Mitte eine hellblaues Quadrat der Länge N ($N < 21$).
Alterntive Varianten:

- (einfach) Nimm eine bekannte Kantenlänge, z.B. 21.
- (schwierig) „Zeichne“ ein echtes baiuwarisches Rautenmuster aus 1 und 0).

`("Ex16_YourName.cpp")`

17. Entwerfe eine Liste von Funktionsprototypen, um eine Variante von “minesweeper auf der Kommandozeile” zu programmieren. Implementiere die Funktionen nicht, überlege dir aber welchen Datenyyp die Rückgabewerte und Argumente haben müssten.

18. Baue einen Konverter, der für positive Zahlen aus dem Dezimalsystem eine Darstellung in einem n-Zahlensystem ($n < 37$) ausgeben kann.

`("Ex18_YourName.cpp")`

Algorithmen

19. Bubblesort Baue eine Funktion

```
void printArray(double data[], unsigned int dataLength);
```

die die Ausgabe eines Arrays ermöglicht. Überlege dir dann ein Verfahren, um eine Funktion mit dem Prototyp

```
unsigned int sortArray(double data[], int dataLength, bool ascending);
```

zu implementieren. Die Funktion soll das array in auf- oder absteigender Reihenfolge (in Abhängigkeit von `ascending`) sortieren und ausgeben, wieviele elementare Tauschoperationen insgesamt nötig waren. Schreibe dazu eine Funktion

```
unsigned int maxSwapsOfBubbleSort(unsig dataLength);
```

Diese soll berechnen, wieviele Tauschoperationen eine Implementierung maximal benutzen muss, um ein array mit der Länge `dataLength` und unbekanntem Inhalt zu sortieren.

`("Ex18_YourName.cpp")`

20. Intervallhalbierungsverfahren
Die Gleichung $y = x2^x$ ist nicht analytisch nach x auflösbar, d.h. es gibt keinen eindeutigen Ausdruck $x = \ldots$, der durch Einsetzen von y einen Wert für x liefert. Anders ausgedrückt, existiert keine Umkehrfunktion für $f : x \longrightarrow f(x), f(x) = x2^x$. Da die Funktion streng monoton ist, kann allerdings durch geschicktes Ausprobieren dennoch einen Wert näherungsweise für ein gegebenes y_{input} berechnet werden.
Dazu werden eine obere Grenze für x_{max} Grenze für x_{min} definiert und das arithmetische Mittel $\bar{x} = \frac{x_{max}+x_{min}}{2}$ (d.h. die Mitte des Intervalls) in die Gleichung eingesetzt:

$$y_{test} = \bar{x} * 2^{\bar{x}}$$

Durch Vergleich des y_{test} mit y_{input} kann nun festgestellt werden, ob der geratene Wert y_{test} zu groß oder zu klein war. Dementsprechend wird nun die obere oder untere Grenze angepasst und auf den Wert der Mitte gesetzt. Dadurch schrumpft das Intervall und die gesuchte Zahl wird immer weiter eingeschränkt.
Aufgabe: Schreibe ein Programm, das 50 solche Iterationen durchführt, um bei obiger Gleichung x für Eingaben $0 \leq y \leq 10^8$ näherungsweise zu berechnen.

`("Ex20_YourName.cpp")`

21. Naive Suche auf C-Strings
Ein C-String ist ein nullterminiertes array von chars, d.h. das letzte Element des arrays hat den Wert 0x00; C-Strings können folgendermaßen initialisiert werden:

```
char c[33] = "For the induction we used IPTG.\n";
```

Jede Position enthält dabei ein Zeichen:

Pos	c[0]	c[1]	c[2]	c[3]	...	c[29]	c[30]	c[31]	c[32]
char	'F'	'o'	'r'	' '	...	'G'	'.'	'\n'	'\0'
hex	0x46	0x6F	0x72	0x20	...	0x47	0x2E	0x0A	0x00

Dabei ist zu beachten, dass immer Platz für das byte 0x00 am Ende mitreserviert wird. Die Länge des arrays muss also immer mindestens $n + 1$ betragen, wenn der C-string n chars besitzt (inklusive Sonderzeichen wie '\n'

Aufgabe: Finde in der Gensequenz bei dem mitgeschickten Programmfragment die Startpositionen alle TATA-Boxen ("TATAA"), indem du die Zeichenfolge "TATAA" mit allen möglichen Positionen abgleichst.

```
("Ex21_YourName.txt")
```

22. Euler-Verfahren

Die Kinetik einer Reaktion $A \longrightarrow B+C$ wird, unter der Annahme, dass das Edukt vollständig reagiert, durch die Gleichung

$$v(t) = \frac{-d[A]}{dt} = k \cdot [A](t)$$

beschrieben. Will man den Verlauf der Konzentrationen berechnen, *ohne* dabei die Differentialgleichung durch Variablenseparation und Integration zu lösen, kann man dazu ein einfaches iteratives Verfahren anwenden:
Aus

$$-\frac{d[A]}{dt} = \frac{d[B]}{dt} = \frac{d[C]}{dt} = k \cdot [A](t)$$

kann durch einfaches Umstellen

$$d[A] = -k \cdot [A](t)dt \qquad \text{und} \qquad d[B] = k \cdot [A](t)dt$$

abgeleitet werden. In einer Näherung kann davon ausgegangen werden, dass die Konzentration von A sich in sehr kleinen Zeitabständen Δt praktisch konstant ist. Durch die Diskretierung des Zeitdifferentials folgt also in der Näherung

$$\Delta[A]_{\hat{t}} = -k \cdot [A]_{\hat{t}}\Delta t \qquad \text{und} \qquad \Delta[B]_{\hat{t}} = k \cdot [A]_{\hat{t}}\Delta t$$

für diskrete Zeiten $\hat{t}$ (Diskrete Zeiten sind z.B. 0 ms, 1 ms, 2 ms, 3 ms,... statt einer kontinuierlichen Zeitachse). Damit kann die Konzentration zu einem Zeitpunkt $\hat{t}$ wie folgt angegeben werden:

$$[A]_{\hat{t}} = [A]_{\hat{t}-1} - [A]_{\hat{t}-1} \cdot k \cdot \Delta t) = [A]_{\hat{t}-1} \cdot (1 - k \cdot \Delta t)$$

$$[B]_{\hat{t}} = [B]_{\hat{t}-1} + [A]_{\hat{t}-1} \cdot k \cdot \Delta t$$

Aufgabe: Berechne den Konzentrationsverlauf über 1 Minute von A und B, mit beliebigem k und Startkonzentration von A und Δt-Abständen von 50 ms. Kennt man also die Startkonzentration $[A]_{0ms}$, kann damit $[A]_{10ms}$ berechnet werden, damit dann $[A]_{20ms}$ usw. Mit $>$ kann die Ausgabe der Konsole in ein Textfile umgeleitet werden, d.h. alles was mit cout geschrieben wird, steht danach stattdessen in einer Datei

```
$./myProgram > outputFile.txt
```

Eine entsprechend formatierte Ausgabe kann von einer Tabellenkalkulation oder einem Plotter eingelesen werden (z.B. als csv):

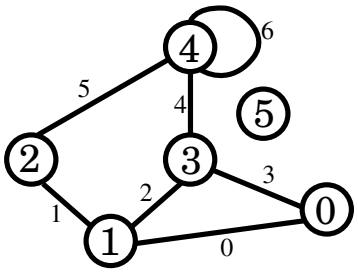
```
t,A,B
0,500,0
10,498.6534,0.13473
.............
```

Hinweis: Das Programm ist iterativ leichter verständlich, auch wenn für die Berechnung eine Rekursionsgleichung angegeben ist.

```
("Ex22_YourName.txt")
```

23. Graphen und Netzwerkanalyse

Ein Graph ist eine fundamentale Datenstruktur, die überall dort Anwendung findet, wo Beziehungen zwischen diskreten Einzelobjekten, Datensätzen oder anderen Entitäten untersucht werden. Formal handelt es sich bei einem Graph um ein Paar $G(V, E)$ mit einer Menge aus Knoten ("vertices") V und einer Menge aus Kanten E.
Hierbei können die Knoten beispielsweise Städte, Menschen, Publikationen, Wirtschaftssubjekte, Atome, Elektronikbauteile, Akupunkturpunkte usw. darstellen. Die Kanten symbolisieren dann dementsprechend Reiserouten, Bekanntschaften, Zitationen, Transaktionen, Bindungen, Leiterbahnen oder (für TCM-Esoteriker) Meridiane.



Um einen Graphen direkt objektorientiert darzustellen, werden weitere Sprachkonstrukte benötigt. Daher bilden wir die Information über die Topologie des Graphen aus dem Bild in einer Adjazenzmatrix $\mathbf{A(G)}$ ab. Dabei handelt es sich um eine "Knoten-mal-Knoten"-Matrix der Form, in der angegeben wird, ob zwei Knoten mit einer Kante verbunden sind.
$\mathbf{B}(G)$

$$\mathbf{A}(G) = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad \mathbf{B}(G) = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Das Konzept des Graphen kann um diverse Eigenschaften erweitert werden (gerichtete Kanten, Farbkanten, Hypergraphen usw.).
Aufgaben: Erstelle einen Graphen mit n Knoten ($n = 200$) als Adjazenzmatrix. Definiere

eine Wahrscheinlichkeit $0 < p < 1$, mit der entschieden wird, ob ein Zufallsgenerator eine 1 oder 0 setzt. Ermittle empirisch, wie hoch die Wahrscheinlichkeit q ist, dass jeder Knoten von einem anderen Knoten aus über die Kanten erreich werden kann (d.h. das alle Knoten miteinander zumindest indirekt miteinander verbunden sind) und plotte $q(p)$.

```
("Ex23_YourName.txt")
```

Pointer, Arrays

24. Pointerarithmetik

Betrachte folgenden Ausschnitte aus einem Programm:

1. ```
   int start = 0;
   int *startAdress = &start;
   cout << startAdress << "\n"; // delivers 0x10000000
   startAdress += 0x100;
   cout << startAdress << "\n"; // 1
   for (int i = 0; i < 16; i++) startAdress += 0x100;
   cout << startAdress << "\n"; // 2
  --startAdress;
   cout << startAdress << "\n"; // 3
   ```

   Welche Werte liefert cout bei 1, 2, und 3, unter der Annahme, dass beim ersten cout die Ausgabe im Kommentar erscheint?

2. ```
   unsigned int startValues[5] = {0x0, 0x0, 0x0000FFFF, 0x0, 0x0};
   unsigned int *ptr = startValues + 2;
   cout << *(ptr) << "\n"; // 1
   char *strangePointer = (char*) ptr;
   strangePointer++;
   cout << *((unsigned int*) strangePointer) << "\n"; // 2
   strangePointer = strangePointer - 2;
   cout << *((unsigned int*) strangePointer) << "\n";  // 3
   ```

   Welche Werte liefert jeweils cout und warum?

```
("Ex24_YourName.txt")
```

25. Bootstrapping

Durch einen extraterrestrischen EMP-Angriff im Jahre 2377 n.Chr. wurden sämtlichen informationsverarbeitenden Systeme auf der Erde zerstört. Der Inhalt sämtlicher Festplatten und Speicher sind unwiderruflich verloren. Lediglich in den Tiefen eines Militärbunkers der Weltregierung finden sich noch ein paar antike Bauteile, mit denen ein paar Prototypen erfolgreich rekonstruiert werden können. Nach dem ein minimales Betriebssystem in Maschinensprache etabliert wurde, soll als nächstes ein erster C-Compiler entwickelt werden, um die Informationstechnik der Zivilisation wieder aufzubauen. Dabei stellt die Menschheit fest, dass bei der Compilerkonstruktion um ein Henne-Ei Problem handelt, da der Compilercode selbst ja noch nicht kompiliert werden kann.

Zunächst wird also nur ein Teil eines Sprachstandards als Binärcode bzw. Assembler entwickelt. Mit dieser minimalen Hochsprache kann dann ein Compiler gebaut werden, der schon das ein oder andere Sprachelement mehr kennt. Dieser Zyklus wird fortgeführt, bis der Compiler alle Eigenschaften des Sprachstandards verarbeiten kann. In der Entwicklugsphase stehen dem Entwickler also nur begrenzte Sprachmittel zur Verfügung. **Aufgabe:** Implementiere eine Funktion

```
char charAt(char *cArr, uint cArrSize, uint pos);
```

Die aus einem char-Array ein bestimmtes Element an der Position `pos` liest, ohne dabei den Zugriffsoperator `[]` zu verwenden. Baue dann damit eine Funktion

```
void reverseString(char *cString)\\
```

die die Zeichenfolge eines nullterminierten C-String umdreht (*in-place*, d.h. direkt auf dem Speicher, der durch den Pointer designiert wird), dabei aber keine Funktionen aus einer anderen Bibliothek verwendet. Überlege dir zunächst, wie du die Länge des C-Strings ermitteln kannst, wenn sie nicht in der Signatur spezifiert wurde.

```
("Ex25_YourName.cpp")
```

26. Pointer auf pointer

1. ```
   void strangeFunction(int **ptrToA);
   ```

   Welcher Ausdruck liefert in der Funktion die Adresse der Variablen a, wenn die Funktion mit dem übergebenen Parameter aufgerufen wird?

2. ```
   int array[1000];
   ```

   Welche zwei Ausdrücke liefern die Adresse des hundertsten integers im array?

3. ```
   int x = 0;
   int *ptrX = &x;
   int **ptrPtrX = (int**) ptr;
   **ptrPtrX = 50;
   ```

   Was geht hier nach unsauberem Casten garantiert schief?

4. ```
   #include <iostream>
   using namespace std;

   void function(int** const & strangeVar);

   int main(){
     int b = 500;
     int *ptr = &b;
     int **ptrPtr = &ptr;
     function(ptrPtr);
   }
   ```

```
void function(int** const & strangeVar){
  int value = ???
  int *adress = ???
  return;
}
```

Was muss auf der rechten Seite der assignments in der Funktion stehen, um den Wert bzw. die Adresse von a zu erhalten?

```
("Ex26_YourName.txt")
```

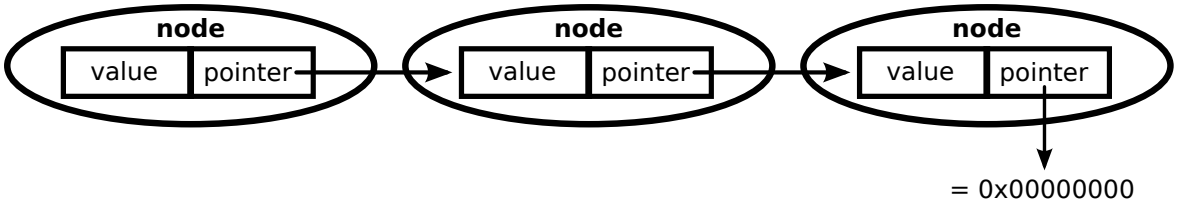
27. Einfach verkettete Listen

Eine einfach verkettete Liste ist eine häufig verwendete Datenstruktur. Sie kann z.B. benutzt werden, um ein dynamisches Array zu implementieren, d.h. ein Array, bei dem die Länge zur Laufzeit des Programms verändert werden kann. Eine einfache Implementierung für eine Liste von integern wird durch `structs` ermöglicht:

```
struct node{
  int value;
  node *next;
};

typedef linkedList node;
```

Jeder `node` beinhaltet hier eine Variable des Arraytyps (in diesem Fall integer), sowie einen Pointer, der auf das nächste Element zeigt, d.h. die Speicheradresse des nächsten `node` enthält. Dieser hat wieder einen Wert und einen pointer auf das nächste Element usw. Das letzte Element wird durch einen null-Pointer gekennzeichnet, d.h. der Wert dieser Pointervariablen ist 0x00000000. Diese Zusammenfassung führt dazu, dass die einzelnen Elemente der List nicht dicht gepackt im Speicher liegen müssen, da über die Speicheradresseninformation auf jedes Element zugegriffen werden kann. Daher können aus dieser Liste Elemente beliebig entfernt oder hinzugefügt werden. Die Listenelemente sollen alle auf dem Heap angelegt werden.



Aufgabe: Implementiere eine verkettete Liste mit `chars` als Typ `charList`, eine Funktion

```
void append(linkedList charList, char item);
```

die eine Liste um ein character verlängert und eine Funktion

```
char removeLastItem(charList);
```

die das letzte `char` von der Liste wieder entfernt und zurückgibt.

```
("Ex27_YourName.cpp")
```

28. Perzeptrone

```
("Ex27_YourName.cpp")
```

Klassen, templates, STL, weiter Algorithmen

28. csv-Dateien und Standardbibliothek

Templates bieten die Möglichkeit, eine Funktion oder unabhängig von einem bestimmten Datentyp zu implementieren. Dieser wird dann vom Compiler zur Laufzeit eingefügt, und zwar mit dem jeweiligen Datentyp an der verwendeten Stelle, d.h. der verwendete Algorithmus ist unabhängig vom Datentyp verwendbar. Eine template-Klasse mit dem template-Typ `T` wird z.B. folgendermaßen realisiert:

```
template <typename T>
class myTempClass {
   public:
   myTempClass(const T &init) : (init) {}
   T getValue() const;
   void setValue(const T &val) {this->value = val;}
   private:
   T value;
};
```

Wird diese Klasse nun irgendwo im Quelltext verwendet, muss `T` an der entsprechenden Stelle spezifiert werden. Dies geschieht ebenfalls durch die Angabe eines Datentyps in spitzen Klammern:

```
int main(){
myTempClass<int> intClass(100);
//.......
intClass.setValue(5);
}
```

Besonders bietet sich diese Konstruktion für Containerklassen an, d.h. Klassen, die außschließlich dafür gedacht sind, viele Instanzen des gleichen oder verschiedener Datentyp(s/en) zusammenzufassen. Solche Als einfachste Beispiel sei hier `std::vector>` genannt. Diese Klasse wird im Prinzip wie ein array eines bestimmten Typs verwendet. D.h. für Aggregate des gleichen Datentyps in fester Ordnung, numerischen Indices und möglichst fixer Größe. Die Länge eines `std::vector` kann variiert werden, allerdings ist der dazugehörige Aufwand linear, da die Verlängerung das Kopieren des gesamten Vektors zur Folge haben kann. Dafür ist garantiert, dass die eigentlichen Daten kompakt im Speicher liegen, was beispielsweise Kompatibilität zu C-Arrays ermöglicht. `std::vector<double>` kann als Objekt wie ein

array vom Typ `double` verwendet werden, ist aber in der Handhabung etwas komfortabler, da einige zusätzlichen Funktionen bereitgestellt werden. Einige einfache Beispiele für den Umgang mit den Funktionen von `std::vector`.

```
unsigned int size = 100;
vector<double> *v = new vector<double>(size); // create new vector<double>
double tenthElement = v->at(10);              // read value from vector
(*v)[10] = 140.0;                                // write element to vector
unsigned int vLength = v->size();              // get number of elements
v->push_back(500.0)                            // append element, thereby
                                              // increasing the length of v by 1
delete v;                                     // free memory
```

Die STL (Stanadard Template Library) ist ein Bestandteil der C++-Bibliothek und verwendet durchgehend das template-Konstrukt.
Ein sinnvoll einsetzbares Programm sollte in der Lage sein, alle zu verarbeitenden Daten aus Dateien zu lesen. Viele naturwissenschafliche Messdaten können im csv-Format als Tabelle abgespeichert werden. Der typische Aufbau besteht dabei aus einer Kopfzeile mit n Einträgen, die von einem Trennzeichen (“,”, “\t”, “;”)getrennt werden. In den nächsten Zeilen folgen dann jeweils die Daten. Während für das Lesen die C++-Standardbibliothek (oder andere Bibliotheken) recht einfache Funktionen bereitstellt, ist bereits die Interpretation und Umwandlung eines solch einfach aufgebauten Formates keine triviale Aufgabe mehr. Daher übernimmt diese Aufgabe hier die mitgesendete Klasse `CsvParser`. Diese Klasse aber ist unter keinen Umständen als Beispiel für saubere Programmierung zu verstehen. Unter anderem verwendet diese Klasse die beiden Klassen `std::string` und `std::vector` der C++-Standardbibliothek. `std::string` repräsentiert Zeichenketten als Objekte. Der `CsvParser` verwendet den `typedef vectorMatrix`, der aus Vektoren eine Matrix aufbaut:

```
typedef vector<vector<double> > vectorMatrix;
```

Wie man sieht, können Templates auch geschachtelt werden. Auf das Element (5, 10) kann dann z.B. folgendermaßen zugegriffen werden, da der erste Zugriffsoperator jeweils ein Element vom Typ `std::vector<double>`

```
double value = matrix.at(5).at(10);
```

Oder auch kürzer:

```
double value = matrix[5][10];
```

In `main()` befindet sich ein Beispiel, wie diese Klasse nun verwendet werden kann:

- Anlegen eines Parser-Objekt: Dem Konstruktor muss ein std::string mit dem absoluten Dateipfad übergeben werden, ein char als Trennzeichen und ein Zeichen für den Dezimalpunkt.
- Lesen der Datei und Umwandlung der Dateidaten in einen vector mit den Kopfzeileneinträgen und eine Datenmatrix

- Zugriff auf die gelesenen Daten durch eine get-Funktionen, die einen pointer liefern.
- Lesende Operationen auf den zugänglich gemachten Daten
- Aufräumen des Speichers, sobald die Daten nicht mehr benötigt werden.

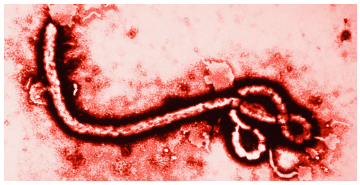
Aufgabe: Erstelle mithilfe dieser Klasse ein Programm, das ein csv-File liest (z.B. das Mitgeschickte), und aus jeder Zeile und jeder Spalte jeweils das arithmetische Mittel berechnet sowie die Matrix transponiert, d.h. Zeilen und Spalten vertauscht.

`("Ex28_YourName.cpp")`

29. Objektmodellierung

Objekte können ebenfalls eine Klassenvariable darstellen. Durch das Zuweisen von Variablenwerten können Eigenschaften, durch die Implementierung der Funktionen können abgebildet werden. **Aufgabe:** Modelliere ein beliebiges Tier aus seinen wichtigsten Körperteilen und den dazugehörigen -funktionen. Schreibe dazu jeweils nur die Definition der Klassen und mit den dazugehörigen Deklarationen der Funktionen, ohne letztere zu implementieren.

`("Ex29_YourName.cpp")`



30. Exploration von gerichteten Graphen und Reduktion

Wie du weißt, kann man Tic-Tac-Toe nur verlieren, wenn der Gegner einen Fehler macht. Wenn jeder perfekt spielt, geht das Spiel auf einem 3x3-Feld immer untentschieden aus. Wie sieht das aber aus, wenn man in einem 3-dimensionalen Raum spielt?
Schnell wird klar, dass der Spieler, der die Position (1,1,1) also den Schwerpunkt des Würfels besetzt einen riesigen Vorteil hat, da er zum einen die meisten direkten potentiellen direkten Gewinnmöglichkeiten besetzt, zum Anderen aber auch die meisten Gelegenheiten für Doppeldrohungen aufbauen kann. Wenn also beim ersten die Mitte besetzt wird, kann eine perfekte Strategie entwickelt werden, die immer zum Sieg führt. Auch beim Schach gibt in bestimmten Stellungen Zugkombinationen, bei denen der Gegner keine Chance mehr hat, und in jedem Fall Matt gesetzt wird, viele Schachprobleme haben zur Lösung eine maximale Zuganzahl, z.B. Matt in max. 3 Zügen. In Spielen mit vollständig bekannter Information (und ohne Zufallseinfluss) wie Schach, Go, Shogi, Dame, Mühle etc. spricht man von einem "forcierten Sieg". Aufgrund der hohen kombinatorischen Möglichkeiten ist bis heute allerdings nicht bekannt, ob solche Strategieen bei den jeweiligen Turnierspielen existieren. Prinzipiell wäre es z.B. beim Schach denkbar, dass solch eine forcierte Strategie nur unter Ausnutzung des Zugzwangs zustande kommen kann. Daher ist bis heute z.B. auch nicht bekannt, ob Weiß als Beginner tatsächlich einen Vorteil haben muss, auch wenn dies für alle bisher tatsächlich gespielten Partien als wahrscheinlich gilt. **Aufgabe:** Untersuche "3x3x3-Sogo" darauf, ob für den anziehenden oder den zweiten Spieler jeweils eine forcierte Strategie besteht, das Sogo wird im Prinzip wie 3D-Tic-Tac-Toe gespielt. Der Unterschied besteht darin, dass die "Schwerkraft" die Anzahl der Zugmöglichkeiten derart einschränkt, dass in die z-Dimension nur Türme gebaut werden dürfen, aber keine Steine in die Luft gesetzt. D.h. dass z.B. die Position (0,0,1) erst besetzt werden kann, sobald auch auf Position (0,0,0) ein Stein gesetzt ist. Während es also bei Tic-Tac-Toe rein kombinatorisch 9! (bzw. 27! für 3D) Trajektorien

gibt (abzüglich derer, die schon vor Vollbesetzung zu einem Sieg führen), sind es bei "Sogo" bedeutend weniger. Gleichzeitig ist es aber wesentlich schwieriger, die strategisch wichtigen Mittelpunkte zu erhalten. Bei einem handelsüblichen 4x4x4-Feld haben daher bei Gelegenheitsspielern trotzdem beide eine gewisse Siegchance. Ein Spielstand lässt sich natürlich am Einfachsten mit einem nxnxn-Tensor darstellen, z.B.

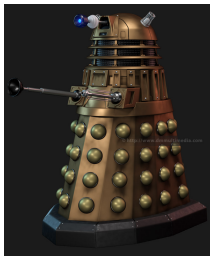
$$\begin{pmatrix} -1 & 1 & -1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Die kombinatorische Anzahl möglicher Spielstände nimmt auch bei Sogo mit jedem Zug zu. Allerdings kann man intuitiv sehen, dass viele dieser kombinatorischen Möglichkeiten lediglich Abbildung einer Symmetrieoperation eines anderen Spiels sind (d.h. es kommt auf das gleiche heraus, ob die Ecke links oben oder rechts unten). Dies bedeutet, dass die Anzahl der möglichen Zustände für einen bestimmten Spielfortschritt zwar anwächst, einige dieser Zustände aber so zusammengefasst werden können, dass keine Information verloren geht (vgl. Abbildung xxx). Um die Anzahl der Zustände also gering zu halten, bietet es sich an, in jeder Runde alle Symmetriegruppen. Dies bedeutet auch, dass die Trajektorieren nicht auseinanderlaufen müssen. Durch diese Operation wird also aus dem Baum ein *gerichteter* Graph. Konkret muss also in jedes Abbild der möglichen Symmetrieoperationen darauf untersucht werden, ob sie mit einer anderen Konfiguration übereinstimmen. Ist dies der Fall, können diese beiden Situationen zusammengefasst werden und es muss nur noch von einer der beiden weitergerechnet werden, ohne dass ein Informationsverlust über alle theoretisch möglichen Situationen auftritt. Insgesamt gibt es bis zu 8 symmetrische Möglichkeiten für einen Spielzustand:

Identität	Spiegelung
90°-Drehung	Spiegelung + 90°-Drehung
180°-Drehung	Spiegelung + 180°-Drehung
270°-Drehung	Spiegelung + 270°-Drehung

Da es maximal 27 Züge gibt, kann also in 27 Schritten die Topoologie eines Graphen erstellt werden. Da jedoch nie eine Kante zwischen einem Zustand des n-ten und $n+k$-ten Zugs ($k \neq 1$) bestehen wird, kann der Graph in 27 Teilgraphen, die je einen Zug repräsentieren, zerlegt werden. Überlege dir anschließend, wie du diesen Graphen verwenden kannst, um eine Antwort auf die Fragestellung zu finden.

`("Ex30_YourName.cpp")`



30. Binärbäume & Parser

Um von der Kommandozeile direkt in einen `std::string` zu lesen, bietet sich die Funktion `getline` an:

```
#include <iostream>
#include <string>
#include <sstream>
```

```
using std::cin;
using std::string;
using std::cout;
using std::endl;

int main(){
   std::string s;
   getline(cin, s);
   cout << string << endl;
}
```

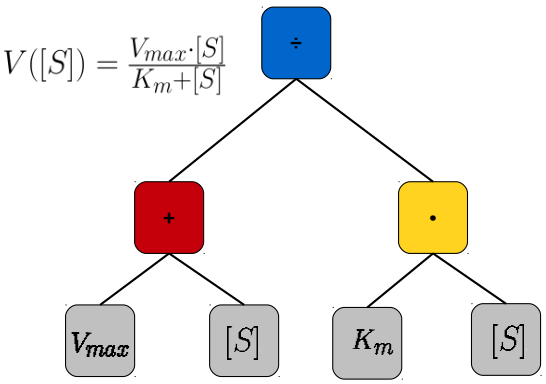
In dieser Aufgabe soll ein Taschenrechner implementiert werden, der folgende Ausdrücke unter Verwendung der 4 Grundrechenarten als Kommandozeileneingabe auslesen und berechnen kann:

```
123.5 + 5 * 2 - 35 / 7
```

Das Problem besteht nun darin, dass die Operatorpräzedenz und die damit Handlungssequenz dynamisch zur Laufzeit hergeleitet werden muss. D.h. der Programmierer kennt nicht die Reihenfolge, in der die Rechnungen ausgeführt werden und kann daher nur Regeln beschreiben, nach denen das Programm die Sequenz
in der die Rechnungen explizit durchgeführt werden sollen. Kommen mehrere gleichrangige Rechnungen vor, dann ist es auch egal, in welcher Reihenfolge die Rechnungen abgearbeitet werden. Da die verwendeten mathematischen Ausdrücke nur 2 Argumente verwenden, kann der Ausdruck z.B. als Binärbaum dargestellt werden. Dabei handelt es sich um eine Datenstruktur, die ein wenig an die bereits behandelte Liste hat. Jeder "node" entspricht dabei einer durchzuführenden Rechenoperation und zeigt dabei entweder auf einen anderen "node", d.h. die Operation verwendet das Zwischenergebnis einer anderen Rechnung. Eine Operation kann auch festgelegte Werte aus dem Ausdruck verwenden.
Beispiel Michaelis-Menten-Kinetik:

$$v([S]) = \frac{v_{max} \cdot [S]}{K_m + [S]},$$

Diese Formel beschreibt für eine Reihe von Enzymen in guter Näherung die Umsatzgeschwindigkeit v in Abhängigkeit von der Substratkonzentration $v[S]$ und verwendet dazu die zwei Konstanten K_m und v_{max}. Der entsprechende Parserbaum sähe folgendermaßen aus:



Wird der Baum traversiert, d.h. sukzessive von unten nach oben abgearbeitet, ergibt sich an der root das Ergebnis der Gleichung. Die Traversierung repräsentiert also genau das Vorgehen, wenn man versuchen würde, die Formel mit gegebenen Werten von Hand zu berechen:

- Multipliziere v_{max} und $[S]$.
- Addiere K_m und $[S]$.
- Teile das Ergebnis des ersten Schrittes durch das des Zweiten.

Will man den Parserbaum für eine gegebene Menge von Operatoren aufstellen, bietet es sich zunächst an, die Baumstruktur analog zu der Liste zu implementieren. Ein `node` kann beispielsweise folgendermaßen konstruiert werden:

```
class ParserTreeNode {
public:
  ParserTreeNode(const double val);
  ParserTreeNode(const char op, ParserTreeNode *firstChild, ParserTreeNode *secondChild)
  // .......
 double result();

private:
  const bool isLeaf;
  union nodeData
  {
     char op;      // is used if node operation
     double val;   // is used if node is leaf and has a fixed value
  };

  nodeData data;
  ParserTreeNode * firstChild;
  ParserTreeNode * secondChild;
};
```

Anschließend sollte der Ausdruck, d.h. mit der niedrigsten Operatorpräzedenz angefangen werden und parallel der Baum von der Wurzel her aufgebaut werden. Ist man bei einema atomaren Ausdruck angekommen, d.h einer Zahl angekommen, wird der `node` als `leaf` markiert, und in der `union` statt einem Operator eine Konstante kodiert. Am besten werden gleich zwei Konstruktoren für jeweils der beiden Funktionsweisen des `node` angelegt. Die Funktion `result()` liefert rekursiv das Ergebnis der jeweiligen Operation, d.h. Aufrufen durch die Wurzel liefert das Ergebnis des gesamten Ausdrucks. Ist der `node` ein Operator, muss auf die beiden Referenzen ("child nodes") zurückgegriffen werden und das Ergebnis (gegebenenfalls berechnet werden). Bei einem `leaf` wird der Konstantenwert direkt zurückgegeben.
Im Grunde handelt es sich bei dem Programm um einen Interpreter für eine eigene kleine Programmiersprache aus 4 Operatoren. Tatsächlich verwenden auch richtige Compiler eine Art

Parserbaum, um den Quelltext des Programmierers auszuwerten und aus den Anweisungen eine Handlungssequenz (in Form von Maschinensprache) abzuleiten.

`("Ex30_YourName.cpp")`

31. Formale Sprachen:
Sei $\mathcal{S}$ eine rekursiv definierte Sprache über dem Alphabet $\Sigma = \{A, B, C, *, +\}$ mit den atomaren Termen A, B, C.
Rekursionsregeln:

1. Ist ϕ ein gültiger Ausdruck in $\mathcal{S}$, dann ist auch $*\phi$ gültig.
2. Sind ϕ und ψ gültige Ausdrücke, dann ist auch $\phi + \psi$ gültig.

Ausdrücke, die nicht unter Anwendung dieser Regeln gebildet werden können, sind nicht gültig. Beispiele:

- C
- $C + B$
- $A + *B$
- $*****A$
- $**B + **C + ***B$

sind gültige Ausdrücke in $\mathcal{S}$.

- ABC
- $AB*$
- $A++C$

sind keine gültigen Ausdrücke in $\mathcal{S}$.
Aufgabe: Entwickle ein Programm, das testet, ob es sich bei dem Inhalt eines `char []` um einen gültigen Ausdruck handelt.
Hinweise zum Aufbau:

- Mit dieser zusätzlichen Zeile kann für `unsigned int` der wesentlich kürzere Name `uint` definiert werden:

```
//..
using namespace std;
typedef unsigned int uint;
//...
```

- Header, die solche nützlichen Definitionen enthalten, sind leider nicht immer im C++-Standard enthalten und werden daher durch viele systemspezifische header ergänzt.

- Ein rekursiver Ansatz sollte deutlich leichter sein als der iterative Weg.
- Die Array-Größe kann als globale Konstante festgelegt werden, da variablen-abhängige Array-Längen nicht durch den C++-Standard unterstützt werden (aber bei `g++` als Erweiterung mit drin sind.)

  ```
  //..
  using namespace std;
  typedef unsigned int uint;
  const uint strSize = 100;
  //...function declarations
  ```

  Die Implementierung folgender, oder ähnlicher Hilfsfunktionen könnte helfen, das Problem zu lösen und gleichzeitig einen lesbaren Quellcode zu erzeugen:

```
/*   Returns if a given term contains a character
 */
bool contains(char term[], uint termSize, char c);
```

```
 /*   Returns if the term contains other characters
  *   than allowed (i.e other chars than 'A', 'B', 'C', '+' or '*')
  */
bool containsOnlyValidLetters(char term[], uint termSize);

 /*   Delivers a substring from "input" from "first" two "
  *   The delivered string is well terminated with 0x00. However, its
  *   allocated space is equal to the size of the input string.
  *   Returns if extraction was possible within the size parameters.
  *   Example: "ABCDEFGHIJK\0" => first = 1, last = 4 ====> "BCDE\0"
  */
bool substring(char input[], char result[], uint &resultSize, uint first, uint last);

 /*   Returns if the given substring is valid
  *   by checking the recursion rules
  *   Recursive function.
  */
bool isValidTerm(char term[], uint termSize);

 /*   Returns if the given term is atomic
  */
bool isAtomicTerm(char term[], uint termSize);
```

Insgesamt sieht der Ansatz also so aus (für Ausdrück mit maximal 99 Zeichen):

```
#include <iostream>
using namespace std;
typedef unsigned int uint;
const uint strSize = 100;

bool contains(char term[], uint termSize, char c);
bool containsOnlyValidLetters(char term[], uint termSize);
bool substring(char input[], char result[], uint &resultSize, uint first, uint last);
bool isValidTerm(char term[], uint termSize);
bool isAtomicTerm(char term[], uint termSize);

int main(){
  main
}

("Ex31_YourName.cpp")
```

32. Von Stapeln und Haufen

```
("Ex32_YourName.cpp")
```

33. Graphische Oberflächen mit QT:

```
("Ex32_YourName.cpp")
```

33. OpenGL:

```
("Ex34_YourName.cpp")
```

35. Neuronale Netzwerke:

```
("Ex35_YourName.cpp")
```