

Übungsaufgaben zur Klausur

Die Aufgaben folgen keiner speziellen Anordnung, weder was die Schwierigkeit, noch was das Thema betrifft. Außerdem sind auch einige Aufgaben nicht im Stil typischer „Klausuraufgaben“, und dienen eher dazu ausgewählte Themen zu vertiefen.

AUFGABE 0 (Kurzfragen). a) Warum sind Turing-Maschinen im Zusammenhang mit dem Begriff der Berechenbarkeit so interessant? Was ist die Church-Turing-These?

- b) Was ist das Halteproblem? Zu welcher “Klasse” von Problemen gehört es?
- c) Programmiersprachen, in denen man (wenn unendlich viel Speicher zur Verfügung stünde) eine Turingmaschine emulieren kann, nennt man Turing-vollständig. Warum ist diese Eigenschaft interessant, bzw. was sagt das über die Programmiersprache aus?
- d) Was ist eine Referenz, wie wird sie deklariert und wozu benutzt man sie? Warum würde man Referenzen an Stelle von Zeigern benutzen? Wann können Referenzen Zeiger nicht ersetzen?
- e) Welche verschiedenen Anwendungsfälle haben Zeiger?
- f) Was versteht man unter einem Seiteneffekt einer Funktion oder Methode?
- g) Wann spricht man von Funktionen, wann von Methoden?
- h) Welchen Vorteil hat es, Speicher dynamisch zu allokatieren? Und wie geht das überhaupt?
- i) Was heißt es, dass C++ streng typgebunden ist? Und was versteht man unter statischer Typisierung?
- j) Was sind typische Eigenschaften funktionaler Programmierung? Welche Vor- und Nachteile hat funktionale Programmierung?
- k) Was ist ein Konstruktor? Was ist der Default-Konstruktor? Was ist der Copy-Konstruktor?
- l) Was ist der Destruktor?
- m) Was bedeutet Sichtbarkeit im Zusammenhang mit Klassen in C++? Welche verschiedenen “Sichtbarkeiten” gibt es in C++? Wofür werden sie

jeweils verwendet?

- n) Was ist Vererbung? Gebe ein Minimalbeispiel an.
- o) Erläutere grob die einzelnen Schritte die durchgeführt werden, um vom Quellcode zum fertigen Programm zu kommen.
- p) Was ist Moore's law? Ist es immer noch gültig? Welche Auswirkungen hat das auf die Softwareentwicklung?
- q) Was versteht man unter Overloading (deutsch: Überladen) von Funktionen?
- r) Was versteht man unter *Call by Value* und *Call by Reference*?
- s) Was versteht man unter der Signatur einer Funktion/ Methode?
- t) Was ist die Rule of Three?
- u) Was sind die Unterschiede zwischen einer *shallow copy* und einer *deep copy*?
- v) Mittels `using namespace std`; kann man alles aus dem Namensraum `std` "bekannt" machen. Welche Nachteile könnte das haben (bspw. in größeren Projekten)?
- w) Was ist eine `friend`-Class?
- x) Was macht das Schlüsselwort `inline` vor einer Funktionendeklaration?
- y) Was ist eine abstrakte Klasse?
- z) Was ist der Unterschied zwischen öffentlicher und privater Vererbung?
- aa) Was sind (rein) virtuelle Methoden?
- bb) Was ist der Unterschied zwischen Overloading (Überladen) und Overriding (Überschreiben)?
- cc) Wozu verwendet man das Schlüsselwort `template`?
- dd) Was ist der Unterschied zwischen `template <class T>` und `template <typename T>`?
- ee) Was ist RISC, was ist CISC?

- ff) Was ist dynamischer Polymorphismus?
- gg) Was ist statischer Polymorphismus?
- hh) Was ist ein vollständiger binärer Baum? Was ist der Unterschied zu einem Heap?
- ii) Wann ist ein Sortierverfahren stabil?

AUFGABE 1. Ist ein Polynom in der Form $p(x) = a_0 + a_1x + \dots + a_nx^n$ gegeben, so können einzelne Werte $p(\xi)$ mit Hilfe des *Horner-Schemas* berechnet werden:

$$\begin{aligned} b_n &= a_n, \\ b_k &= a_k + \xi b_{k+1} \text{ für } k = n-1, \dots, 0, \\ p(\xi) &= b_0 \end{aligned}$$

- Schreibe eine rekursive Funktion `double horner(double coeffs[], int deg, double x)`, die für ein Polynom vom Grad `deg`, dessen Koeffizienten im Array `coeffs` stehen, das Polynom an der Stelle `x` auswertet. Die Koeffizienten sollen dabei folgendermaßen in einem Array gespeichert werden:
Für $p(x) = 3 - 2x + 3x^2 - 4x^4$ ist `coeffs = { -4, 0, 3, -2, 3 }` (d.h. der höchste Koeffizient kommt zuerst). Außerdem ist `deg = 4` und bspw. $p(2)$ würde ausgewertet durch den Aufruf `horner(coeffs, 4, 2) = -981`.
- Schreibe nun die Funktion aus a) iterativ. Was ist die Zeitkomplexität in Abhängigkeit von `n` der iterativen Variante?
- * Was könnten Vorteile des Horner-Schemas sein? Warum wertet man das Polynom nicht einfach "normal" aus, indem man einfach ξ einsetzt?

AUFGABE 2. Schreibe eine Funktion, die ein Polynom der Form $p(x) = a_0 + a_1x + \dots + a_nx^n$ von der Standardeingabe liest und die Koeffizienten a_0, \dots, a_n extrahiert.

Bsp.: Eingabe: $2 + 2x - 3x^2 + 4x^3$, Ausgabe: `{ 2, 2, -3, 4 }`

Hinweise:

- Der Einfachheit halber, kann davon ausgegangen werden, dass alle Koeffizienten einstellig sind, also dass $|a_i| \leq 9$, für alle $i = 0, \dots, n$
- Ersetze zunächst alle '-' durch '+-' mit der Methode `std::string::replace` (um bspw. im String `s` alle 'a' durch 'b' zu ersetzen: `replace(s.begin(), s.end(), 'a', 'b')`)
- "Splitte" nun den String jeweils an den '+'. Du kannst annehmen, dass eine Funktion `std::vector<std::string> split(const std::string &s, char delimiter)` gegeben ist, die einen String `s` bei jedem Vorkommen von `delimiter` aufsplittet und die einzelnen Teile in einen `std::vector` schreibt (also bspw.: `split("abc+def+ghi+jkl", '+') == { "abc", "def", "ghi", "jkl" }`)
- Nun kannst du den im vorherigen Schritt erstellten Vektor durchlaufen und die Koeffizienten extrahieren. Hier musst du nur aufpassen, falls ein Koeffizient 0 ist und das dazugehörige Monom gar nicht in der

Eingabe vorkam (also bspw. $3 - 4x^2$, hier fehlt der Term $0x$, d.h. der Koeffizientenvektor ist hier $\{ 3, 0, -4 \}$)

AUFGABE 3. Schreibe eine Funktion `std::vector<std::string> split(const std::string &s, char delimiter)`, die einen String `s` bei jedem Vorkommen von `delimiter` aufsplittet und die einzelnen Teile in einen `std::vector` schreibt (also bspw. sollte das Ergebnis von `split("abc+def+ghi+jkl", '+')` das hier sein: `{"abc", "def", "ghi", "jkl" }`)

- Einen neuen `std::vector`, der `std::string`s enthält, legt man so an:
`std::vector<std::string> res;`
- Um `res` ein neues Element hinzuzufügen, verwende `res.push_back("test")`
- Die Funktion (siehe auch Übungsblatt 10)
`std::istream& getline(std::istream& is, std::string &s)`
akzeptiert auch noch einen dritten Parameter `char` `delimiter` und kann auch auf `std::stringstream`s operieren (um einen `std::stringstream` aus einem String zu erstellen, übergibt man einfach den jeweiligen String dem Konstruktor).

AUFGABE 4. Implementiere die Funktion `void reverse(std::string& s)`, die den String `s` "umdreht", also bspw. aus `Hallowelt tlewollaH` macht, auf die folgenden Arten:

- a) Implementiere die Funktion mit einer `for`-Schleife und der Funktion `std::swap(char &a, char &b)`, die den Wert von `a` und `b` tauscht. Die `for`-Schleife soll dabei *nicht* den ganzen String durchlaufen, sondern (falls `n` die Länge des Strings ist) nur von `i = 0` bis `i < n / 2`.
- b) Implementiere die Funktion rekursiv, die Signatur der Funktion ändert sich dabei zu: `std::string reverse(const std::string &s)` und die Funktion ändert nicht den ursprünglichen String sondern gibt das Ergebnis zurück.

AUFGABE 5. Rechne die folgenden Zahlen um:

- a) 11011101_2 ins Dezimalsystem
- b) 116_{10} ins Dualsystem

AUFGABE 6. Gegeben seien zwei Rechtecke in einem 2D-Koordinatensystem

(jeweils parallel zu den Koordinatenachsen). Diese können eindeutig durch Angabe der Koordinaten der linken oberen Ecke und der rechten unteren Ecke bestimmt werden:

```
class Rectangle {
public:
    double x1, y1; // linke obere Ecke
    double x2, y2; // rechte untere Ecke
}
```

Schreibe eine Methode `bool Rectangle::doesOverlap(const Rectangle other)`, die für ein Rechteck prüft, ob sich seine Fläche und die Fläche eines anderen Rechtecks überschneiden.

AUFGABE 7. a) Implementiere eine Klasse, die eine doppelt verkettete Liste repräsentiert und zumindest folgendes Interface erfüllt:

```
class DLL {
public:
    void addFirst(int value);
    int removeFirst();

    void print() const;
    void printReverse() const;

    ~DLL();

private:
    struct IntListItem {
        int value = 0;
        IntListItem *next = nullptr;
        IntListItem *prev = nullptr;
    };

    IntListItem *first = nullptr;
    IntListItem *last = nullptr;
    int count = 0;
};
```

Untenstehende main-Funktion soll folgenden Output erzeugen:

List: 4 -> 3 -> 2 -> 1

List (reversed): 1 -> 2 -> 3 -> 4

```
int main() {
    DLL list;
    list.addFirst(1);
    list.addFirst(2);
    list.addFirst(3);
    list.addFirst(4);

    list.print();
    list.printReverse();
}
```

```

    return 0;
}

```

- b) Welche Vorteile hat diese Variante gegenüber einfach verketteten Listen? Und gegenüber einem normalen Array? Welche Nachteile gibt es?

AUFGABE 8. Gegeben sei die Turingmaschine M mit den Endzuständen z_3 und z_5 , dem Startzustand z_0 und untenstehendem Programm.

Zustand	Eingabe	Ausgabe	Richtung	Folgez.
z_0	b	b	R	z_1
z_1	b	0	L	z_3
z_1	0	b	R	z_2
z_2	b	0	L	z_4
z_2	0	b	R	z_1
z_4	b	0	L	z_5

Die Turingmaschine berechnet die charakteristische Funktion $\chi_M : \mathbb{N} \rightarrow \{0, 1\}$ einer unbekannten Menge $M \subseteq \mathbb{N}$, d.h.

$$\chi_M(n) = \begin{cases} 1 & n \in M \\ 0 & n \notin M \end{cases}$$

- a) Ziel ist es, herauszufinden, um welche Menge M es sich handelt. Eingabe und Ausgabe werden dabei durch eine Unärdarstellung codiert: $n \in \mathbb{N}$ entspricht 0^{n+1} , d.h. $0 = 0$, $1 = 00$, $2 = 000$ etc. Die Ausgabe der Turingmaschine ist am Ende die Zahl, die rechts vom Schreib-Lese-Kopf steht.
- b) Entwerfe nun eine Turingmaschine, welche die charakteristische Funktion χ_{M^c} des Komplements $M^c = \mathbb{N} \setminus M$ berechnet, d.h. $\chi_{M^c}(n) = 1 - \chi_M(n)$.

AUFGABE 9. Du stößt in einem Programm auf folgenden Ausschnitt:

```

// Prüfe, ob der Eintrag an der Stelle i, j Null ist
if (std::abs(matrix[i][j]) < 1e-12) {
    ...
}

```

Erkläre, warum der Code und das Kommentar darüber (auch wenn nicht ideal formuliert) trotzdem Sinn ergeben.

AUFGABE 10. Formalisiere den Begriff Algorithmus. Welche speziellen Typen von Algorithmen gibt es und was zeichnet sie aus? Nenne jeweils ein Beispiel.

AUFGABE 11. Ersetze im folgenden Programmausschnitt die `if-else`-Konstruktion durch eine einzige möglichst vereinfachte `if`-Anweisung:

```
if (true) {  
    if (x > 0) {  
        if (x%2 == 0)  
            std::cout << x;  
        } else {  
            if ((-x)%3 == 0)  
                std::cout << x;  
        }  
    }  
}
```

AUFGABE 12. a) Was macht der folgende Algorithmus?

```
void func(int a[], int b) {  
    for (int j = 1; j < b; j++)  
        for (int i = j; i > 0 && a[i-1] > a[i]; i--) {  
            int c = a[i];  
            a[i] = a[i-1];  
            a[i-1] = c;  
        }  
}
```

b) Welche Laufzeit hat der Algorithmus im schlechtesten Fall?

AUFGABE 13. a) Berechne die Zweierkomplementdarstellung der folgenden Zahlen:

i) -30

ii) 25

iii) 123

b) Rechne die folgenden Zahlen, die in Zweierkomplementdarstellung vorliegen, ins Dezimalsystem um:

i) 11111111

ii) 11100000

iii) 00000001

c) Welche Vorteile hat die Zweierkomplementdarstellung?

AUFGABE 14. Implementiere eine Klasse, die einen Stack modelliert. Der Typ der Werte auf dem Stack, soll durch templates allgemein gehalten werden, d.h. die Klassendefinition sieht folgendermaßen aus:

```
template <class T>
class Stack {
public:
    Stack(int capacity);
    ~Stack();
    bool push(T);
    T pop();

private:
    ...
};
```

AUFGABE 15. a) Erstelle eine Funktion `double mean(const std::vector<double>& v)`, die den Mittelwert $\mathbb{E}[v]$ aller Einträge in dem Vektor v zurückliefert:

$$\mathbb{E}[v] = \frac{1}{N} \sum_{i=1}^N v_i,$$

wobei N die Anzahl der Einträge im Vektor angibt.

b) Erstelle eine Funktion `double median(const std::vector<double>& v)`, die den Median aller Einträge in dem Vektor v zurückliefert. Um den Median zu berechnen, erstelle eine sortierte Kopie \tilde{v} von v und bestimme $M(v)$ als

$$M(v) = \begin{cases} \tilde{v}_{\frac{N+1}{2}} & N \text{ ungerade} \\ \frac{1}{2}(\tilde{v}_{\frac{N}{2}} + \tilde{v}_{\frac{N}{2}+1}) & N \text{ gerade} \end{cases}$$

wobei N die Anzahl der Einträge im Vektor angibt. Beachte die unterschiedlichen Indizierungsstrategien (1-basiert in der Formel oben, 0-basiert in C++), sowie den Spezialfall, dass der Vektor leer ist. Benutze zum Sortieren `std::sort(v.begin(), v.end())`.

AUFGABE 16. a) Schreibe eine Funktion `bool check_parentheses(std::string symbols)`, die für eine Zeichenkette `symbols` überprüft, ob diese genau so viele öffnende Klammern '(' wie schließende ')' enthält. Die Funktion soll auch `false` zurückgeben, wenn eine schließende Klammer vor einer öffnenden Klammer auftritt.

- i) Verwende zunächst einfach eine `int`-Variable um mitzuzählen.
 - ii) Verwende `std::stack<char>`.
- b) Kann man einen endlichen Automaten definieren, der dieselbe Aufgabe löst?

AUFGABE 17. Gegeben seien folgende Zahlen:

{5, 8, 2, 10, 1, 3, 2, 6}

- a) Baue aus diesen Zahlen einen Max-Heap (d.h. das größte Element steht in der Wurzel). Zeichne den vollständigen Baum nach jedem Schritt und erkläre kurz, wie das Element eingefügt wurde.
- b) Lösche nun die Wurzel aus dem Baum und erkläre die dabei auftretende Schritte.
- c) Wieviele "Ebenen" hat ein Heap (in Abhängigkeit der Anzahl der Elemente n)? Wie kann man daraus die Komplexität der Operation "Einfügen" ableiten?
- d) Erläutere kurz die Funktionsweise von Heapsort.

AUFGABE 18. Schreibe eine templatisierte Funktion `quicksort`, die einen `std::vector<T>` mittels dem Quicksort-Algorithmus sortiert. Das Pivot-Element soll dabei immer das "mittlere" Element der jeweiligen Liste sein (bei einer geraden Anzahl an Elemente soll das letzte Element der ersten Hälfte ausgewählt werden).

AUFGABE 19. Es seien zwei Algorithmen f, g gegeben, die beide die selbe Aufgabe lösen. Es ist bekannt, dass

$$f(n) = \mathcal{O}(n) \quad \text{und} \quad g(n) = \mathcal{O}(n^2)$$

- a) Diskutiere die folgende Aussage:

Der Algorithmus f ist schneller als g .

- b) Es wurde nun die Zeit in ms gemessen, die die beiden Algorithmen für verschiedene Eingabelängen n brauchten:
Offenbar ist g für all diese Eingabelängen schneller als f . Steht das im Widerspruch zu deiner Erklärung zur Aussage von a)?

Eingabelänge	Laufzeit von f	Laufzeit von g
10	1000000	100
100	10000000	10000
1000	100000000	1000000
10000	1000000000	100000000

- c) Nenne die (asymptotischen) Laufzeiten der in der Vorlesung besprochenen Sortieralgorithmen und sortiere diese.

AUFGABE 20. Ziel dieser Aufgabe ist es, in mehreren Schritten eine Funktion zu schreiben, die zwei `std::string`s darauf überprüft, ob eines ein Anagramm des anderen ist. Wir benutzen dazu die Eindeutigkeit der Primfaktorzerlegung, was auf einen cleveren – wenn auch ineffizienten – Trick führt. Angenommen wir haben jedem Buchstaben des Alphabets eindeutig eine Primzahl zugeordnet. Wir können dann jedem Buchstaben eines Wortes eine Primzahl zuordnen und anschließend das Produkt dieser Primzahlen bilden. Dann gilt: Ein Wort ist genau dann Anagramm eines anderen Wortes, wenn dieses Produkt für beide Wörter übereinstimmt. Gehe nun folgendermaßen vor:

- Schreibe eine Funktion `bool isPrime(unsigned int n)` die für die Zahl `n` prüft, ob diese eine Primzahl ist (bspw. durch einfaches Ausprobieren mithilfe des Modulo-Operators `%`).
- Schreibe eine Funktion `unsigned int findNextPrime(unsigned int n)`, die die kleinste Primzahl zurückgibt, die größer als `n` ist. Benutze die Funktion aus a).
- Schreibe eine Funktion `unsigned int getPrime(const char c)` die für einen Buchstaben mithilfe der Funktion aus b) eine Primzahl zurückgibt. Diese Primzahl soll eindeutig für diesen Buchstaben sein (Hinweis: `chars` sind eigentlich auch nur Zahlen, können also direkt nach `int` gecastet werden).

Diese Vorarbeit erlaubt es uns nun, eine Funktion `bool anagram(const std::string first, const std::string second)` zu schreiben, die die eingangs gestellte Aufgabe löst. Um die beiden `std::string`s Buchstabe für Buchstabe zu durchlaufen können *range-based* For-Schleifen verwendet werden: `for (auto c: string)`

AUFGABE 21. Alle Teilaufgaben dieser Aufgabe sind von der selben Art und können unabhängig voneinander gelöst werden. Ziel ist es jeweils, den Code auf Fehler zu überprüfen und die Fehler zu korrigieren. Dabei sind neben *syntaktischen* Fehlern (also solche, die zur Compilezeit vom Compiler erkannt werden) auch *semantische* Fehler gesucht. Letztere sind u.a. solche

Fehler, die erst zur Laufzeit zu Problemen führen (könnten). Aber auch Fehler, die man als „schlechten Stil“ bezeichnen könnte, sind gesucht. Am rechten Rand steht jeweils die Anzahl der Fehler.

Es wird unterschieden zwischen Programmen (der vollständige Code ist gegeben und muss so ausführbar sein) und Ausschnitten (nur Teile sind gegeben; fehlende Deklarationen von Variablen oder Funktionen sind *keine* Fehler).

a) Betrachte das folgende Programm: min. 5 Fehler

```
int main() {  
    array<int,10> zahlen = {0,1,2,3,4,5,6,7,8,9,10};  
    for (const int i = 0; i <= 10; i++) {  
        zahlen[i] = i * 2;  
    }  
}
```

b) Betrachte den folgenden Ausschnitt: min. 6 Fehler

```
...  
void summe(int m) {  
    return n * summe(n-1);  
}  
...  
  
// Berechne die Summe der Zahlen bis 10 und gebe sie aus  
std::cout >> "Summe = " summe(10);  
...  

```

c) Betrachte das folgende Programm: min. 5 Fehler

```
int main() {  
    const int x = std::cin >> x;  
    // Pruefe ob x gerade  
    if (x % 2 = 0)  
        cout << "x ist gerade";  
        x = x / 2;  
    else  
        cout << "x ist ungerade";  
}
```

d) Betrachte den folgenden Ausschnitt:

min. 5 Fehler

```
...
// Suche Maximum von arr
void max(int arr[], int x);
{
    int temp = arr[0]
    for(i = 0; i <= arr.size(); i++) {
        if( arr[i] < temp) {
            temp = arr[j];
        }
    }
    return temp;
}
```

e) Betrachte das folgende Programm:

min. 6 Fehler

```
template <class T>
class A{
    A(T a){
        b = a;
    }
private:
    void print() {
        std::cout << b;
    }
};

int main() {
    A a(3);
    a.print();
}
```

AUFGABE 22. Gegeben sei ein aufsteigend sortiertes Array von ganzen Zahlen `int[n]` `a` der Länge `n` und eine ganze Zahl `int` `x`. Es darf angenommen werden, dass `a` nicht leer ist.

Die folgenden Funktionen implementieren eine *binäre Suche* von `x` in `a` als Funktion `find(a,n,x)`.

```
int find(int a[], int left, int right, int x) {
    int mid = (left + right) / 2;

    if (left > right || x == a[mid])
        return mid;

    if (x < a[mid])
        return find(a, left, mid-1, x);
    else
        return find(a, mid+1, right, x);
}

int find(int a[], int n, int x) {
```

```

    return find(a, 0, n-1, x);
}

```

- Was berechnet `find(a,n,x)`? Welche Bedeutung hat der Rückgabewert?
- Was ist die Komplexität des Algorithmus in Abhängigkeit von n ? Es genügt den Vergleich `x == a[mid]` zu betrachten (d. h. es soll herausgefunden werden, wie oft dieser Vergleich in Abhängigkeit von n durchgeführt wird).
- Was wird zurückgegeben, wenn der Algorithmus "erfolglos" endet? Ändere den Algorithmus so, dass in diesem Fall -1 zurückgegeben wird.
- Schreibe eine iterative (d. h. nicht-rekursive) Version des Algorithmus.

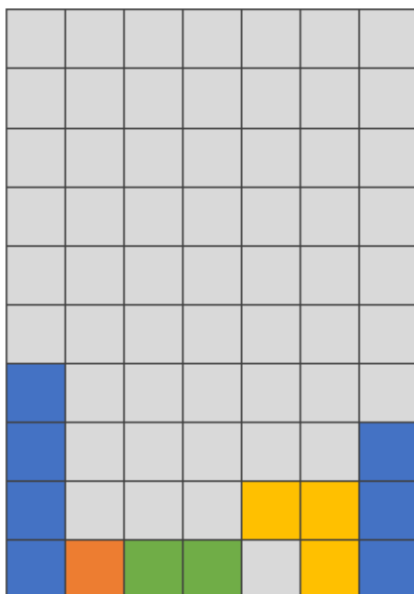
AUFGABE 23. In dieser Aufgabe soll eine Methode implementiert werden, die Teil einer Klasse innerhalb eines Tetris-Programms sein könnte. Die Klasse soll dabei das Spielfeld repräsentieren und zwar als zweidimensionales Array (d. h. als Matrix). Dazu nehmen wir an, dass eine Matrix-Klasse existiert, die bspw. so benutzt wird:

```
Matrix<int,10,7> grid; // 10x7 Matrix mit Integer-Werten
```

Zugriff auf die Matrixelement sei mit eckigen Klammern möglich, d. h. um das Element in der 5. Zeile und 4. Spalte zu verändern, kann man schreiben:

```
spielfeld[4][3] = 2;
```

Für das folgende Spielfeld enthält die Variable `grid` dann folgende Werte (die Werte codieren die Farben)



```

grid = {
    { 0, 0, 0, 0, 0, 0, 0 },
    { 0, 0, 0, 0, 0, 0, 0 },
    { 0, 0, 0, 0, 0, 0, 0 },
    { 0, 0, 0, 0, 0, 0, 0 },
    { 0, 0, 0, 0, 0, 0, 0 },
    { 0, 0, 0, 0, 0, 0, 0 },
    { 1, 0, 0, 0, 0, 0, 0 },
    { 1, 0, 0, 0, 0, 0, 1 },
    { 1, 0, 0, 0, 4, 4, 1 },
    { 1, 2, 3, 3, 0, 4, 1 }
};

```

Die Aufgabe ist nun eine Methode zu schreiben, die die unterste Zeile löscht, falls sie "voll" ist.

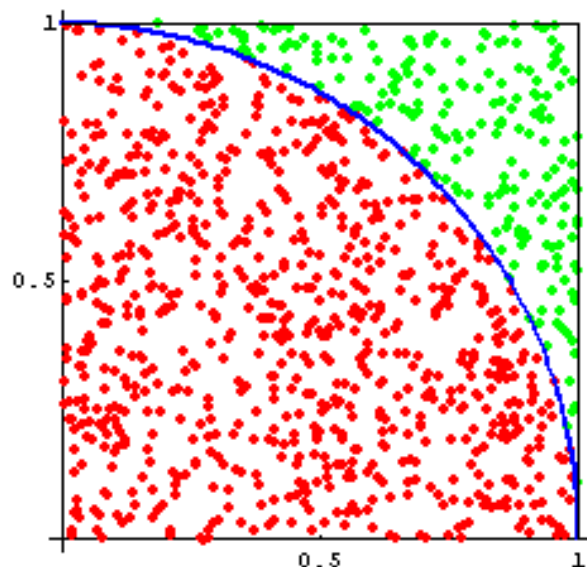
- a) Schreibe eine Methode `bool is_last_line_complete()`, die prüft, ob die letzte Zeile des Grids gelöscht werden kann, d. h. prüft, ob diese Zeile nur nicht-null Einträge enthält. Die Größe des Spielfeldes kann dabei mit `grid.rows()` bzw. `grid.cols()` abgefragt werden.
- b) Schreibe nun eine Methode, die die letzte Zeile löscht. Dabei sollen einfach alle Zeilen darüber nach unten verschoben werden und die erste Zeile des Spielfeldes mit Nullen gefüllt werden.

AUFGABE 24. Diese Aufgabe beschäftigt sich mit sog. *Monte Carlo Integration*. Ziel ist es die Zahl π zu approximieren. Die Methode basiert auf folgender Idee:

Generiert man n zufällige Punkte in einem Quadrat der Länge l und zählt dann die Punkte, die in diesem Quadrat in einem Kreis liegen, der den Durchmesser l hat, dann gilt

$$\lim_{n \rightarrow \infty} \frac{A_{\text{Kreis}}}{A_{\text{Quadrat}}} = \pi, \quad (1)$$

wobei A die Anzahl der Punkte innerhalb des Kreises bzw. des Quadrates angibt. In dieser Aufgabe soll eine Funktion geschrieben werden, die für gegebenes n den Quotienten in (1) berechnet. Als Quadrat wählen wir das Quadrat mit den Randpunkten $(-1, -1), (-1, 1), (1, 1), (1, -1)$. Allerdings betrachten wir nur ein Viertel dieses Quadrats und multiplizieren später das Ergebnis einfach mit 4, siehe Abbildung:



Ziel ist es nun eine Funktion `double pi_mc(int n)` zu schreiben, die nach dem obigen Prinzip π approximiert. Gehe dazu folgendermaßen vor:

- Erzeuge n zufällige Punkte im Einheitsquadrat (wie in der Abbildung oben). Es kann davon ausgegangen werden, dass eine Funktion `randu(int a, int b)` existiert, die gleichverteilte Zufallszahlen im Intervall $[a, b]$ erzeugt.
- Zähle die Punkte, die innerhalb des Viertelkreises liegen (in der Abbildung rot). Hinweis: Der Kreis hat Radius 1. Der Abstand zweier Punkte $p_1 = (x_1, y_1)$ und $p_2 = (x_2, y_2)$ ist

$$d(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

- Teile am Ende die Anzahl der Punkte im Kreis durch die Anzahl der insgesamt erzeugten Punkte und multipliziere das Ergebnis noch mit vier (s.o.).