1. Näherungsweise Berechnung der Quadratwurzel

Das Heron-verfahren beschreibt die schrittweise Verfeinerung einer Wurzel zahl. Dabei wird als erstes ein Schätzwert hergenommen und dieser anschließend verfeinert. Der verfeinerte Wert wird mit jedem durchlauf noch weiter verfeinert.

Wurzel	5	0	39876	39876
EPS	0.00002	0.0000000006	0.00004	1
Ergebnis	2.2360679775	0.0000000000	199.6897593769	199.6897598700

2. Suche nach einer Teilzeichenkette

Es wurden zwei Algorithmen implementiert die dieses Problem lösen. Eine Bruce Force Methode und den Boyer-Moore-Horspool Algorithmus. Bei der Bruce Force Variante wird jeder Charakter in der Ausgangs Zeichenkette mit der Teilzeichenkette abgeglichen. Bei kompletter Übereinstimmung wird ein Match ausgegeben.

Der Boyer-Moore-Horspool Algorithmus legt als erstes eine Sprungtabelle der Teilzeichenkette an. Dabei gilt die Formel für den Sprung „Länge – Pos – 1“. Bei gleichen Buchstaben wird der Sprung Wert des Charakters überschrieben. Wenn die Sprungtabelle angelegt ist wird die Teilzeichenkette von rechts nach links Charakter für Charakter an der Zeichenkette überprüft. Bei einem Missmatch wird eine Verschiebung der Teilzeichenkette nach der Sprungtabelle vorgenommen. Falls der Charakter nicht in der Tabelle vorkommt wird automatisch die Länge der Teilzeichenkette gesprungen. Wenn alle Zeichen mit der Teilzeichenkette gleich sind wird ein Match ausgegeben und die Länge der Zeichenkette gesprungen.

```
char string[] = "Fischers Fritz fischt frische Fische, frische Fische fischt
Fischers Fritz";
char pattern[] = "isch";
```

```
Console:

Hier hab ich was gefunden:
 1  16  24  31  40  47  54  61
Ich habe 8 Matches gefunden!
```

3. Schrittweise Verfeinerung: Verschlüsselung

Der Ausgangstext wird an die erste Funktion übergeben. Es wird überprüft ob das kommende Wort das 5te ist. Falls nein wird dieses nicht verschlüsselt. In einer Unterfunktion wird das erste Wort gesucht und zurückgegeben. Das erste Sonderzeichen (‚.‘ ‚,‘ , ‘) ist Teil des Wortes. Ein Zeiger zeigt nun auf die letzte Position und wird ebenfalls zurückgegeben. Anschließend wird das Wort verschlüsselt (falls nicht 5tes Wort) anhand des zugrundeliegendem Algorithmus (siehe Aufgabenstellung) und wird wieder zurückgegeben. Nun wird dieses verschlüsselte Wort an den verschlüsselten String angehängt.

```
encodestring(↓String text_plain, ↑String text_encoded) {
    Const Integer C_enc_unen = 5     //Das wie vielte Wort wird übersprungen

    Integer enc_wordcount = 0
    Integer text_pos = 0
    String text_partencode = ""

    While(text_pos < text_plain.length()){
        if (enc_wordcount >= C_enc_unen) {
            text_partencode += findWordinStr(text_plain, text_pos)
        } else {
            text_partencode += encodeWord(findWordinStr(text_plain, text_pos))
        }
    }
    text_encoded = text_partencode
}


encodeWord(↓String word_plain, ↑String word_encr) {
    Integer word_pos = 0
    Integer word_length = word_plain.length()
    Switch(word_plain[word_length-1]) {
        case ',':
            word_encr = '@'
            --word_length
        Break

        case '.':
            word_encr = '#'
            --word_length
        Break

        case ' ':
            word_encr = RandChar()
            --word_length
        Break
    }
    word_plain.SubString(0,word_length)
    Switch (word_plain) {
        case "heute":
            word_encr = "nretsO" + word_encr
        break;

        case "Bahnhof":
            word_encr = "402U" + word_encr
        break;

        case "alle":
            word_encr = "hci" + word_encr
        break;

        default:
            while (word_pos < word_length) {
                word_encr = word_plain[word_pos] + word_encr
                ++word_pos
            }
    }
}


findWordinStr(↓String text, ↓↑Integer pos, ↑String word) {
    boolean add = TRUE
    while(pos < text.length() && text[pos] != ',' && text[pos] != '.' && text[pos] !
= ' ') {
        word += text[pos]
        ++pos
    }
```

```
    while(pos < text.length() && (text[pos] == ',' || text[pos] == '.' || text[pos] == ' ')) {
        if (add == TRUE) {
            word += text[pos]
            add = FALSE
        }
        ++pos
    }
}


RandChar(↑char rnd_char) {
    Switch (rand() % 3) {
        Case 0:
            rnd_char = '?'
        Break;

        Case 1:
            rnd_char = '%'
        Break;

        Case 2:
            rnd_char = '&'
        Break;
    }
}
```