# Strukturierte Programmierung Übung 3

Auszuarbeiten bis 05.11.20

## 1. Palindrom ( 2 + 2 + 2 Punkte )

Ein Palindrom ist eine Folge von Zeichen, die sowohl von vorne als auch von hinten gelesen, das gleiche Wort ergibt.

z.B.: OTTO, REITTIER, ROTOR, …

Entwickeln Sie einen Algorithmus, der für eine Folge von Zeichen bestimmt, ob es sich um ein Palindrom handelt oder nicht. Das zu untersuchende Wort wird als Eingabeparameter zur Verfügung gestellt. Das Testergebnis wird über einen Wahrheitswert zurückgeliefert.

a) Geben Sie den Algorithmus als Pseudocode an

b) Geben Sie den Algorithmus als Ablaufdiagramm an

c) Geben Sie den Algorithmus als Struktogramm dar

## 2. Häufigkeitsanalyse ( 5 Punkte )

Gegeben ist eine Zeichenfolge *text* der Länge ($n \geq 0$). Gesucht ist ein Algorithmus, der die Häufigkeit des Vorkommens (in Prozent) für die Vokale bestimmt, wobei zwischen Groß- und Kleinschreibung nicht unterschieden wird. Verwenden Sie für Ihren Algorithmus die folgende Schnittstelle:

`VowelOccurencies(↓text, ↑freqA, ↑freqE, ↑freqI, ↑freqO, ↑freqU)`

Geben Sie auch den Datentyp der Objekte *text*, *freqA*, *freqE*, *freqI*, *freqO* und *freqU* an. Überlegen Sie sich sinnvolle Testfälle und führen Sie entsprechende Schreibtischtests durch!

## 3. Russische Bauernmultiplikation ( 2 + 4 + 3 Punkte )

Die „Russische Bauernmultiplikation“ ist ein Verfahren zur Multiplikation zweier Natürlicher Zahlen. Informationen dazu finden Sie z.B. unter https://de.wikipedia.org/wiki/Russische_Bauernmultiplikation.

a) Erklären Sie das Verfahren

b) Entwickeln Sie einen Algorithmus für das Verfahren als Pseudocode

c) Stellen Sie Ihre Lösung als Ablaufdiagramm dar

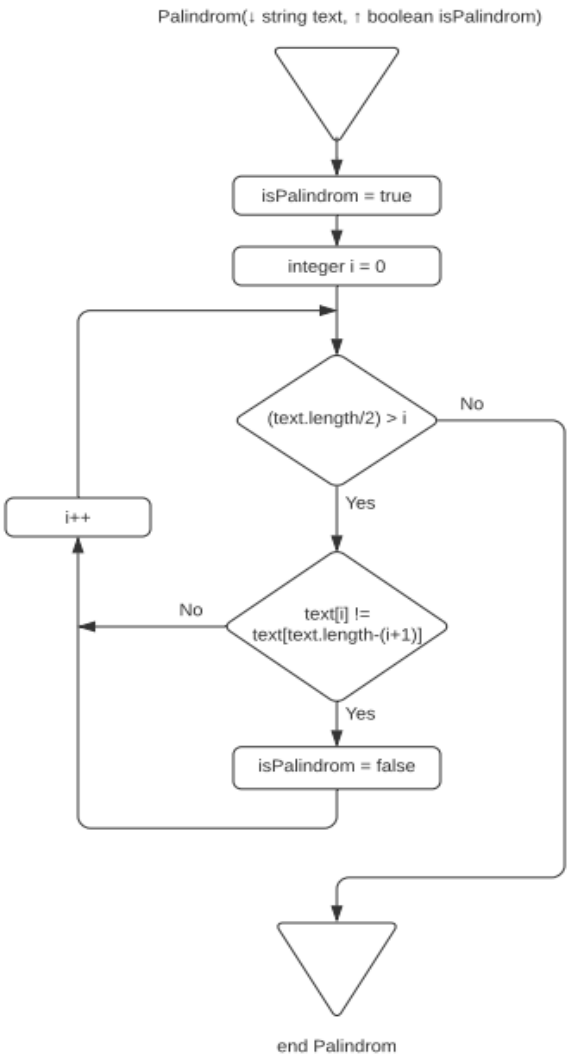
1 Palindrom

a) Pseudo-Code Alogrithmus

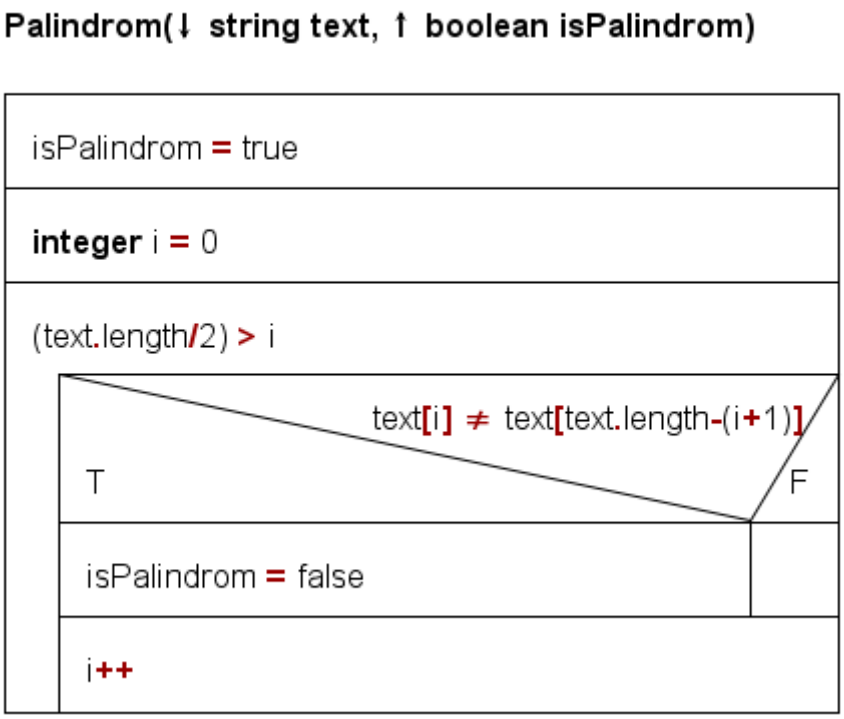
```
palindrom(↓string text, ↑boolean isPalindrom) {
    isPalindrom = true
    integer i = 0

    while((text.length/2) > i)
        if(text[i] != text[text.length-(i+1)])
            isPalindrom = false
        end if
        i++
    end while
}
```

b) Ablaufdiagramm



c) Struktogramm (programm used: Structorizer)




Schreibtischtests

| Annahme: text = "OTTO" | isPalindrom | i | (text.length / 2) > i | text[i] != text[text.length-(i+1)] |
|---|---|---|---|---|
| | TRUE | 0 | TRUE | FALSE |
| | TRUE | 1 | TRUE | FALSE |
| | TRUE | 2 | FALSE | |
| | | | | |
| Annahme: text = "PLAN" | TRUE | 0 | TRUE | TRUE |
| | FALSE | | | |
| | | | | |
| Annahme: text = "" \|\| text = " " \|\| text = "a" | TRUE | 0 | FLASE | |
| | | | | |
| Annahme: text = "ROTOR" | TRUE | 0 | TRUE | FALSE |
| | TRUE | 1 | TRUE | FALSE |
| | TRUE | 2 | FALSE | FALSE |

## 2 Häufigkeitsanalyse

PseudoCode Algorithmus:

```
VowelOccurencies(↓string text, ↑integer freqA, ↑integer freqE, ↑integer freqI, ↑integer freqO, ↑integer freqU) {
    freqA = 0
    freqE = 0
    freqI = 0
    freqO = 0
    freqU = 0

    if (text.length > 0)
        integer i = 0

        while(i < text.length) do
            character currChar = text[i]

            if (currChar == 'A' || currChar == 'a')
                freqA++
            else if (currChar == 'E' || currChar == 'e')
                freqE++
            else if (currChar == 'I' || currChar == 'i')
                freqI++
            else if (currChar == 'O' || currChar == 'o')
                freqO++
            else if (currChar == 'U' || currChar == 'u')
                freqU++

            i++
        end while

        freqA = (freqA * 100) / text.length
        freqE = (freqE * 100) / text.length
        freqI = (freqI * 100) / text.length
        freqO = (freqO * 100) / text.length
        freqU = (freqU * 100) / text.length
    end if
}
```

Schreibtischtests:

| Annahme: "Hallo mein Name ist Marco" \| length = 25 | freqA | freqE | freqI | freqO | freqU | text.length > 0 | i | i < text.length | currChar |
|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | TRUE | 0 | TRUE | H |
| | 0 | 0 | 0 | 0 | 0 | | 1 | TRUE | a |
| | 1 | 0 | 0 | 0 | 0 | | 2 | TRUE | l |
| | 1 | 0 | 0 | 0 | 0 | | 3 | TRUE | l |
| | 1 | 0 | 0 | 0 | 0 | | 4 | TRUE | o |
| | 1 | 0 | 0 | 1 | 0 | | 5 | TRUE | White Space |
| | 1 | 0 | 0 | 1 | 0 | | 6 | TRUE | m |
| | 1 | 0 | 0 | 1 | 0 | | 7 | TRUE | e |
| | 1 | 1 | 0 | 1 | 0 | | 8 | TRUE | i |
| | 1 | 1 | 1 | 1 | 0 | | 9 | TRUE | n |
| | 1 | 1 | 1 | 1 | 0 | | 10 | TRUE | White Space |
| | 1 | 1 | 1 | 1 | 0 | | 11 | TRUE | N |
| | 1 | 1 | 1 | 1 | 0 | | 12 | TRUE | a |
| | 2 | 1 | 1 | 1 | 0 | | 13 | TRUE | m |
| | 2 | 1 | 1 | 1 | 0 | | 14 | TRUE | e |
| | 2 | 2 | 1 | 1 | 0 | | 15 | TRUE | White Space |
| | 2 | 2 | 1 | 1 | 0 | | 16 | TRUE | i |
| | 2 | 2 | 2 | 1 | 0 | | 17 | TRUE | s |
| | 2 | 2 | 2 | 1 | 0 | | 18 | TRUE | t |
| | 2 | 2 | 2 | 1 | 0 | | 19 | TRUE | White Space |
| | 2 | 2 | 2 | 1 | 0 | | 20 | TRUE | M |
| | 2 | 2 | 2 | 1 | 0 | | 21 | TRUE | a |
| | 3 | 2 | 2 | 1 | 0 | | 22 | TRUE | r |
| | 3 | 2 | 2 | 1 | 0 | | 23 | TRUE | c |
| | 3 | 2 | 2 | 1 | 0 | | 24 | TRUE | o |
| | 3 | 2 | 2 | 2 | 0 | | 25 | FALSE | |
| | 12 | 8 | 8 | 8 | 0 | | | | |
| | | | | | | | | | |
| Annahme: "" \| length = 0 | 0 | 0 | 0 | 0 | 0 | FLASE | | | |
| | | | | | | | | | |
| Annahme: "aeiou" \| length = 5 | 0 | 0 | 0 | 0 | 0 | TRUE | 0 | TRUE | a |
| | 1 | 0 | 0 | 0 | 0 | | 1 | TRUE | e |
| | 1 | 1 | 0 | 0 | 0 | | 2 | TRUE | i |
| | 1 | 1 | 1 | 0 | 0 | | 3 | TRUE | o |
| | 1 | 1 | 1 | 1 | 0 | | 4 | TRUE | u |
| | 1 | 1 | 1 | 1 | 1 | | 5 | FALSE | |
| | 20 | 20 | 20 | 20 | 20 | | | | |

3 Russische Bauernmultiplikation

a) Erklärung zum Verfahren

Die Russische Bauernmultiplikation ist ein Multiplikationsverfahren indem der Multiplikator solange halbiert wird bis er 1 erreicht. Der Multiplikant zudem wird immer mit sich selber addiert. Der Multiplikant wird in jedem verlauf, falls der Multiplikator ungerade ist, zu dem endergebnis addiert.

Beispiel: (aus wikipedia)

| Multiplikator | Multiplikand | zu addieren |
|---|---|---|
| 27 | 82 | 82 |
| 13 | 164 | 164 |
| 6 | 328 | ~~328~~ |
| 3 | 656 | 656 |
| 1 | 1312 | 1312 |
| **Produkt:** | | **2214** |

Auf der rechten Seite sieht man, dass nur die Multiplikands zum ergebnis addiert werden wo we der Multiplikator ungerade war.

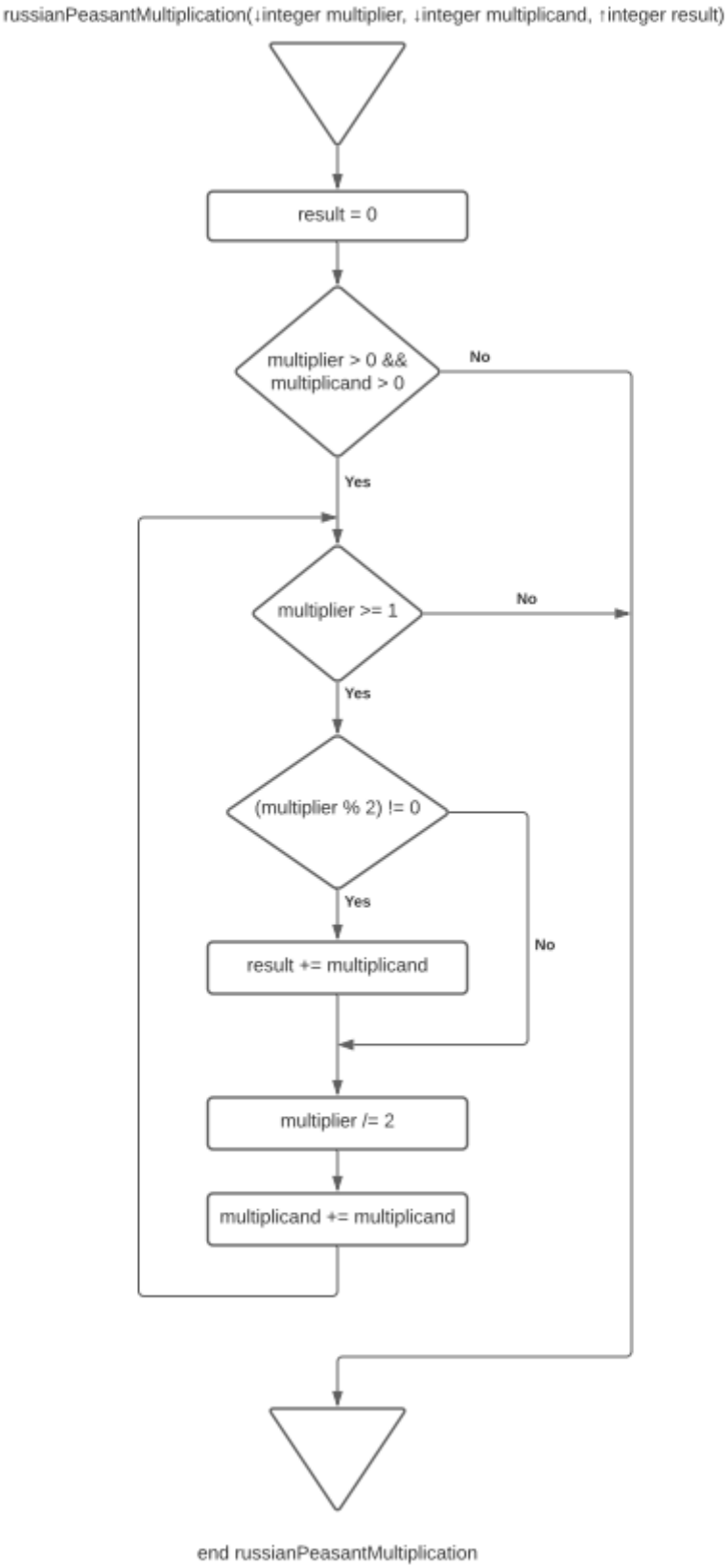
b) Pseudo-Code Alogrithmus

```
russianPeasantMultiplication(↓integer multiplier, ↓integer multiplicand, ↑integer result) {
    result = 0

    if (multiplier > 0 && multiplicand > 0)
        while (multiplier >= 1)
            if ((multiplier % 2) != 0)
                result += multiplicand
            end if

            multiplier /= 2
            multiplicand += multiplicand
        end while
    end if
}
```

c) Ablaufdiagramm



Schreibtischtests:

| Annahme: 2,2 | multiplier | multiplicand | result | multiplier > 0 && multiplicand > 0 | multiplier >= 1 | (multiplier % 2) != 0 |
|---|---|---|---|---|---|---|
| | 2 | 2 | 0 | TRUE | TRUE | FALSE |
| | 1 | 4 | 0 | | TRUE | TRUE |
| | 0 | 8 | 4 | | FALSE | |
| | | | | | | |
| Annahme: 4, 3 | 4 | 3 | 0 | TRUE | TRUE | FALSE |
| | 2 | 6 | 0 | | TRUE | FALSE |
| | 1 | 12 | 0 | | TRUE | TRUE |
| | 0 | 24 | 12 | | FALSE | |
| | | | | | | |
| Annahme: 0, 2 \|\| 0, 0 \|\| -3, 2 \|\| 3, 0 | 0 | 2 | 0 | FALSE | | |
| | | | | | | |
| Annahme: 7, 1 | 7 | 1 | 0 | TRUE | TRUE | TRUE |
| | 3 | 2 | 1 | | TRUE | TRUE |
| | 1 | 4 | 3 | | TRUE | TRUE |
| | 0 | 8 | 7 | | FALSE | |
| | | | | | | |