## 2.5 Skripte/Mehrfachauswahl

### 2.5.1 Gesucht: Das Passende Programm zu jeder Datei

Das Shellskript `start.sh` soll zu einer Datei das passende Programm finden. Anschließend soll es dieses Programm starten und den Dateinamen als ersten Parameter hinzufügen. So soll `start.sh heute.py` die Befehlszeile `python heute.py` aufrufen.

### 2.5.2 Verzweigungen

Natürlich kann man das mit einer Verzweigung lösen:

startpy1.sh

```
1 #!/bin/bash
2 DATEI="$1"
3 if echo *.py | grep "$DATEI"
4 then
5    python3 "$DATEI"
6 else
7    echo "Kann $DATEI nicht starten"
8 fi
```

Das hat aber zwei Nachteile:

a) In der Verzweigung, wie sie hier benutzt wird, ist keine Mustererkennung eingebaut – stattdessen wird die Wildcard-Expansion der Shell benutzt. Und dann wird noch grep benutzt, um herauszufinden, ob die gesuchte Datei im Ergebnis der Expansion enthalten ist. Mit dem Befehl `[[` klappt es besser: `if [[ "$DATEI" == *.py ]]` usw.

b) Das Programm soll nicht nur Python-Programme starten, sondern auch PDF-Dateien öffnen, Bilder anzeigen, MP3-Dateien abspielen. Dann braucht man aber viele Verzweigungen. Das Skript wird so zu einer langen Kette von `if`-, `then`- und `elif`-Zeilen.

### 2.5.3 Mehrfachauswahl

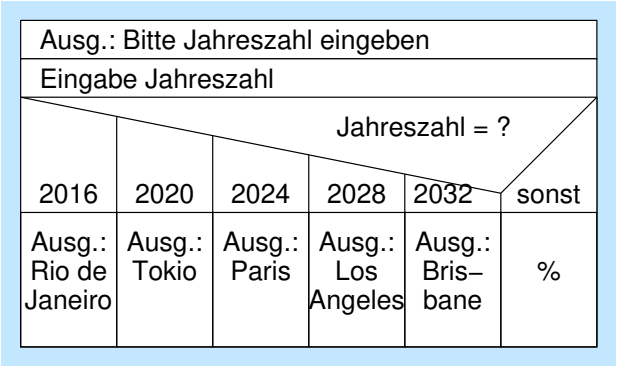


Abbildung 1: Struktogramm zur Mehrfachauswahl

Deshalb gibt es in vielen Programmiersprachen das Strukturelement *Mehrfachauswahl*: Ein Wort (oder allgemein eine Variable) wird nacheinander mit mehreren anderen Worten (Konstanten oder Variablen) verglichen. Sobald ein Vergleich passt, wird nur der eine Zweig ausgewählt, der zu diesem Vergleich gehört. Abbildung 1 zeigt das zugehörige Struktogramm.

Dieses Struktogramm hat in der Shell-Skriptsprache die Form:

olympia.sh

```
#!/bin/bash
echo -n "Bitte Jahreszahl eingeben: "
read JAHR
case "$JAHR" in
    2016) echo "Rio de Janeiro" ;;
    2020) echo "Tokio"          ;;
    2024) echo "Paris"          ;;
    2028) echo "Los Angeles"    ;;
    2032) echo "Brisbane"
          echo "ziemlich weit weg";;
esac
```

Falls der Inhalt von `$JAHR` auf das Muster `2016` passt, wird die Befehlsliste hinter `2016)` ausgeführt, und zwar bis zum *doppelten Semikolon*. Damit ist dieser Zweig beendet, und die Mehrfachauswahl wird sofort verlassen. Anders als bei C++, Java und verwandten Sprachen ist hier keine `break`-Anweisung nötig. Wie man am letzten Zweig sehen kann, darf so eine Befehlsliste auch mehr als einen Befehl umfassen. Nur am *doppelten Semikolon* ist Schluss.

### 2.5.4 Mehrfachauswahl und Muster

Wie man im folgenden Beispiel sieht, kann ein Muster auch Sternchen, Fragezeichen und eckige Klammern enthalten: Der Muster-Vergleich findet nach den Regeln der Wildcard-Expansion von Pfadnamen (*pattern matching* bzw. *filename globbing*) statt. Und zwar auch dann, wenn das Wort gar kein Dateiname ist, so wie hier:

ipadresse.sh

```
#!/bin/bash
echo -n "IP-Adresse eingeben: "
read IPADDR
case "$IPADDR" in
    127.*) echo "Localhost";;
    10.[1-3].*)  echo "Unser LAN";;
    19?.1*) echo "Kenn ich nicht";;
esac
```

Wem die Syntax des Befehls zu rustikal ist, der kann vor jedes Muster eine öffnende Klammer setzen:

```
case "$IPADDR" in
    (127.*) echo "Localhost";;
esac
```

### 2.5.5 Verknüpfung mehrerer Muster

Mehrere Muster können auch mit `|` oder-verknüpft werden:

editor1.sh

```
#!/bin/bash
echo "Wie heisst Ihr Editor?"
read EDITOR
case "$EDITOR" in
    joe|JOE)
        echo "Das war einfach:"
```

```
7         echo "Joe's own Editor";;
8     xemacs|emacs)
9         echo "extended-meta-alt-control-shift";;
10    vi|vim|elvis)
11        echo "vi oder so";;
12 esac
```

### 2.5.6 Default-Zweig

Ein Default-Zweig ist in der Shell-Skriptsprache nicht vorgesehen. Stattdessen findet sich in den meisten Mehrfachauswahlen in der Praxis *am Ende* ein Zweig, in dem auf das Muster $*$ geprüft wird:

editor2.sh

```
1 #!/bin/bash
2 echo "Wie heisst Ihr Editor?"
3 read EDITOR
4 case "${EDITOR}" in
5     nano) echo "Verwandt mit pico";;
6     pico) echo "Editor aus dem Email-Programm PINE.";;
7     *)    echo "${EDITOR} kenne ich nicht.";;
8 esac
```

Dieser Zweig passt auf jede Eingabe und wird somit dann benutzt, wenn vorher nichts gepasst hat.

### 2.5.7 Lösung (erweiterbar)

startpy2.sh

```
1 #!/bin/bash
2 DATEI="$1"
3 case "$DATEI" in
4     *.py) python3 "$DATEI";;
5     *)    echo "Kann $DATEI nicht starten";;
6 esac
```

Terminal

```
schueler@debian964:~$ echo 'print("Hello, World")' > neu.py
schueler@debian964:~$ startpy2.sh neu.py
Hello, World
schueler@debian964:~$ startpy2.sh neu.mp3
Kann neu.mp3 nicht starten
schueler@debian964:~$
```