The AWK programming language (1977)

Awk is a general purpose programming language which is designed for processing data. The name awk is derived from the surnames of its authors: Alfred **A**ho, Peter **W**einberger, and Brian **K**ernighan. (This tutorial follows closely chapter 1 of "The AWK Programming Language" from the same authors.)

Getting started

First, we create the data file hiwi.dat using emacs, vi or any other editor.

name	hourly wage in Euro	working hours per week
Daniel	8.20	10
Dorothea	7.95	24
Peter	5.05	0
Claudine	9.05	12
Nils	13.55	40

Let's look at the following line

awk	'$3>0	{print $1,$2*$3}'	hiwi.dat
	pattern	action	input file

Awk processes data line by line and executes the command in brackets if the condition in the pattern is fulfilled.

$1	entry in first column in current output line
$2	entry in second column in current output line
…	
$0	complete line, e.g., {print $0} prints the current entry
NR	number of lines read so far
NF	number of fields in current output statement

The awk statement above translate into "Take input file hiwi.dat, go through it line by line, if the number of hours per week is greater than 0, print the name and the total pay".

If we want to save the result to a file t1, we type
awk '$3>0 {print $1,$2*$3}' hiwi.dat > t1
We can also write an awk program hiwi.awk which only consists of the action statement, and invoke it with awk –f hiwi.awk hiwi.dat.

Exercises:

1. Print the complete data set

A: awk '{print $0}' hiwi.dat

2. Print the hourly wage for entry 4 and 5 and save the result in t1

A: awk 'NR>3 {print $2}' hiwi.dat > t1

3. Write and run an awk program called vacation.awk which prints the name of the people who didn't work this week

A: $3==0 {print $1}
awk –f vacation.awk hiwi.dat

Output and operators

Print several fields {print $1,$2}
Print text {print "Hello world!"} (quotation marks are also used when an expression is compared to a string)

Fancier output can be generated with the printf statement which is similar to printf in **C**:
awk '{printf("total pay for %s is %.2f\n", $1, $2*$3)}' hiwi.dat

Awk also uses similar operators:
<,>,<=,>=,==,!=,
NOT !
AND &&
OR ||

Example: comparison by text
awk '$1=="Claudine" {print $0}' hiwi.dat

Exercises:
4. Reproduce the output of the printf statement above with the print command
A: awk '{print "total pay for",$1,"is",$2*$3}' hiwi.dat

5. Write an awk statement which produces the following output:

1	Daniel	8.20	10
2	Dorothea	7.95	24
3	Peter	5.05	0

A: awk 'NR<=3 {print NR,$0}' hiwi.dat

BEGIN and END

BEGIN is a special pattern which is invoked before the first line is read.
END is a special pattern which is invoked after the last line is read.

Example:
awk 'BEGIN {a=0} $2>10 {a++} END {print a}' hiwi.dat

Exercises:
6. What does this line do?
A: Counts the number of people who earn more than 10 Euro.

7. Write an awk statement which calculates the overall total pay.
A: awk 'BEGIN {a=0} {a+=$2*$3} END {print a}' hiwi.dat

8. Write and run an awk program called average_rate.awk which calculates the average hourly wage paid this week.
A:
```
BEGIN   {a=0; b=0}
        {a+=$2*$3; b+=$3}
END     {print a/b}
```
awk –f average_rate.awk hiwi.dat

(At this point you should realize why you need to learn this language!)

Control-flow statements and arrays

if/else, while, for, and arrays work like in **C**.

Example: print data in reverse order

```
        {a[NR]=$0}
END     {i=NR;
        while (i>0) {print a[i];i--;}
        }
```

Exercise:
9. Obtain the same result using a for-statement
A:

```
        {a[NR]=$0}
END     {for(i=NR;i>0;i--) print a[i]}
```