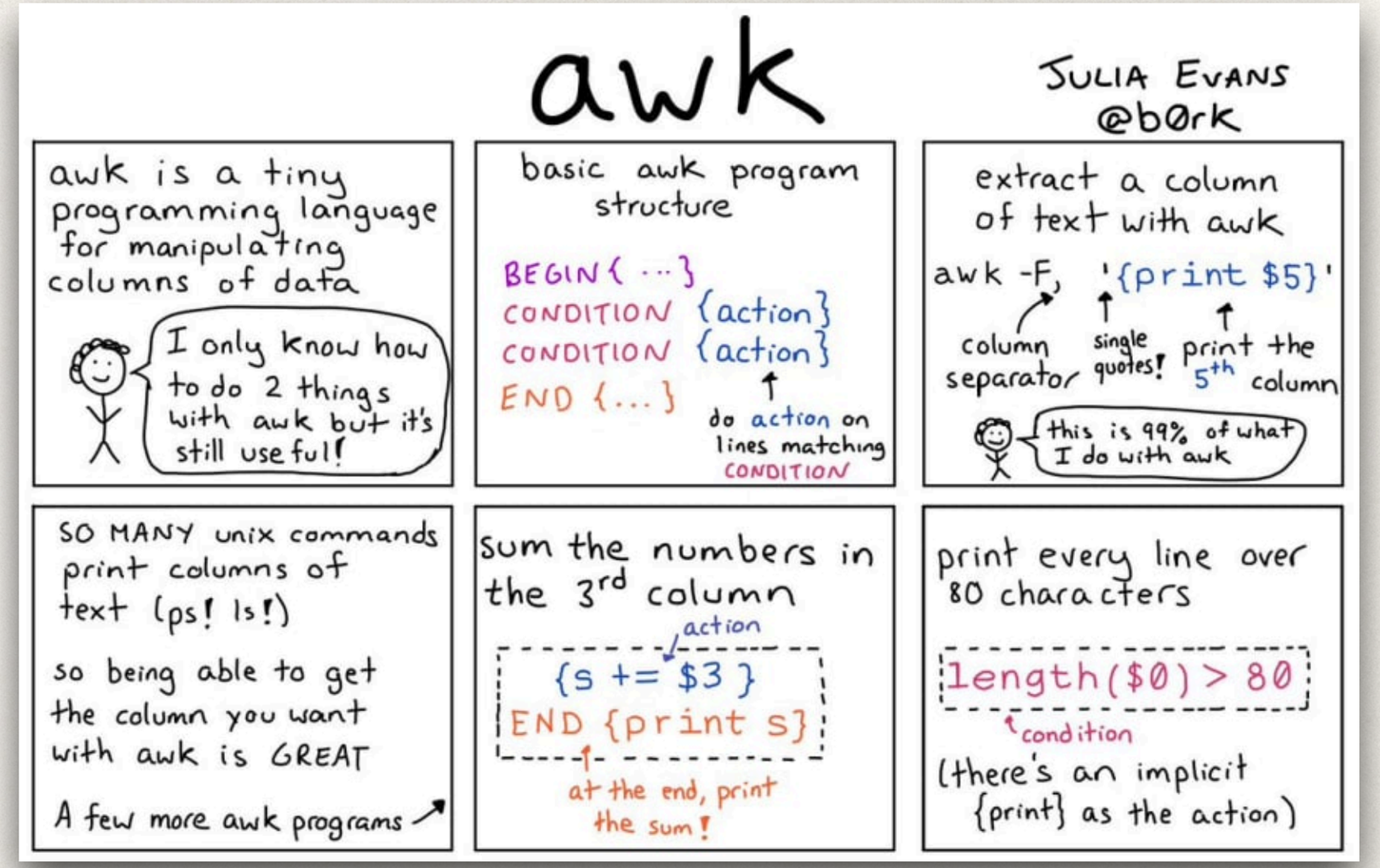


AWK

01418235 UNIX & Shell Programming

Chavalit Srisathapornphat

Outline

- Introduction
- awk variables and operations
- Expressions
- Control statements
- Associative Array
- Array Loop
- String Functions
- Math Functions
- User-defined Functions

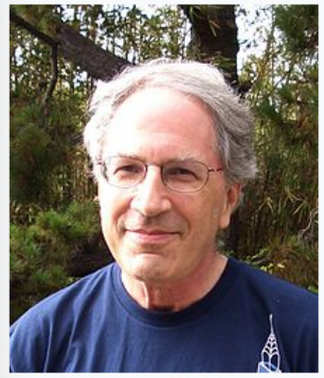
Introduction

- Authors (1977)
  - Alfred V. Aho,
  - Peter J. Weinberger, and
  - Brian W. Kernighan
- A powerful programming language
  - Disguised as a utility
- Behavior:
  - Like sed
    - reads the input file,
      line by line, and performs an action on a part of or on the entire line
  - Unlike sed
    - does not print the line unless specifically told to do so
- Execution
  - awk 'pattern{action}' input-file
  - awk –f scriptFile.awk input-file

Alfred Aho

Born	August 9, 1941 (age 77) Timmins, Ontario
Residence	United States
Nationality	Canadian American
Alma mater	University of Toronto Princeton University
Known for	Awk programming language Principles of Compiler Design Compilers: Principles, Techniques, and Tools Aho-Corasick algorithm

Peter Jay Weinberger



2009 portrait of Weinberger

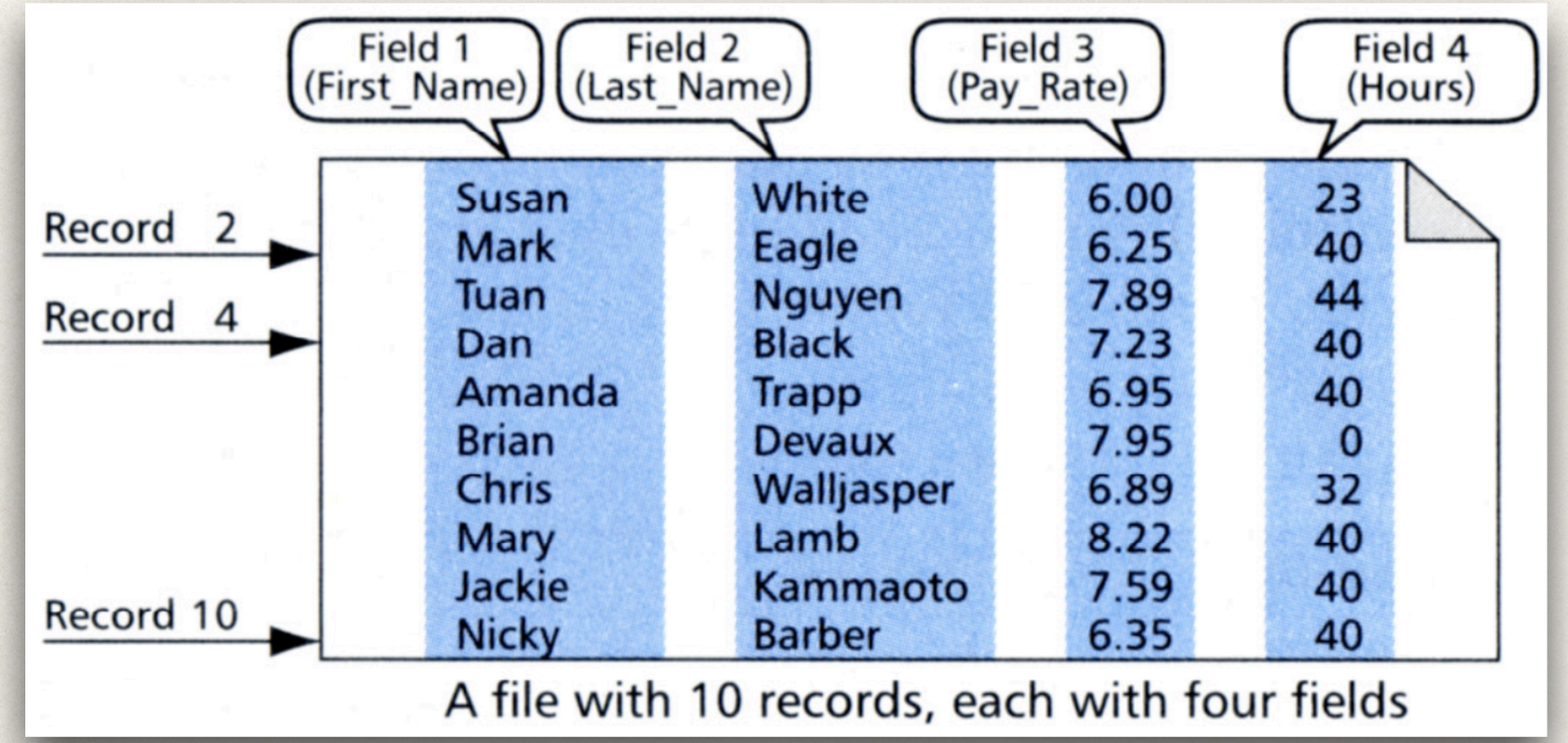
Born	August 6, 1942 (age 76)
Alma mater	University of California, Berkeley (Ph.D. 1969)
Known for	AWK (programming language)
Scientific career	
Fields	Number theory, computer science
Institutions	University of Michigan Bell Labs

Brian Kernighan

Brian Kernighan at Bell Labs in 2012 photographed by Ben Lowe

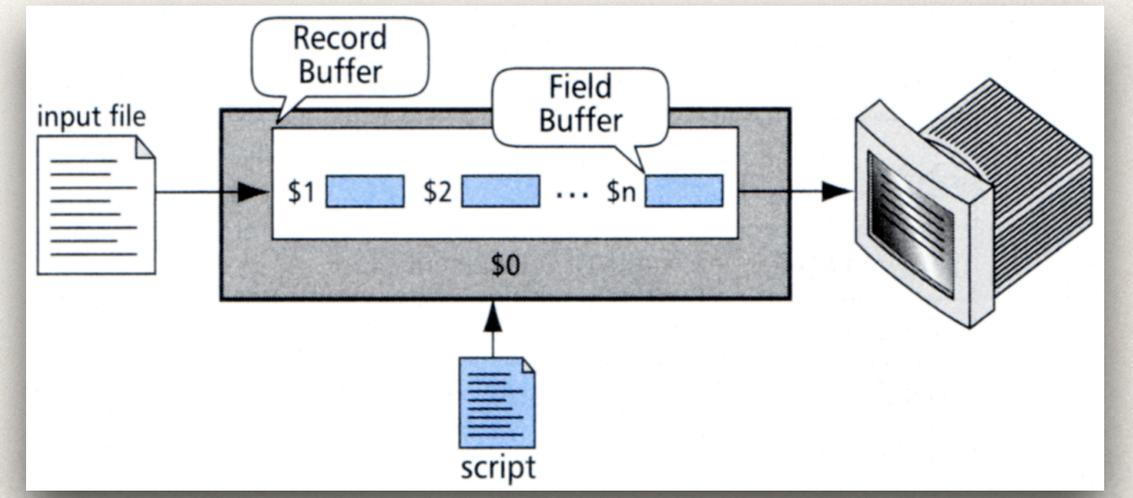
Born	Brian Wilson Kernighan January 1, 1942 (age 76)[1] Toronto, Ontario
Nationality	Canadian
Citizenship	Canada
Alma mater	University of Toronto Princeton University (PhD)
Known for	Unix AWK A Mathematical Programming Language (AMPL) Kernighan–Lin algorithm Lin–Kernighan heuristic *The C Programming Language* (book)[2]

Fields and Records: Data file



A file with 10 records, each with four fields

awk buffer



awk Variables

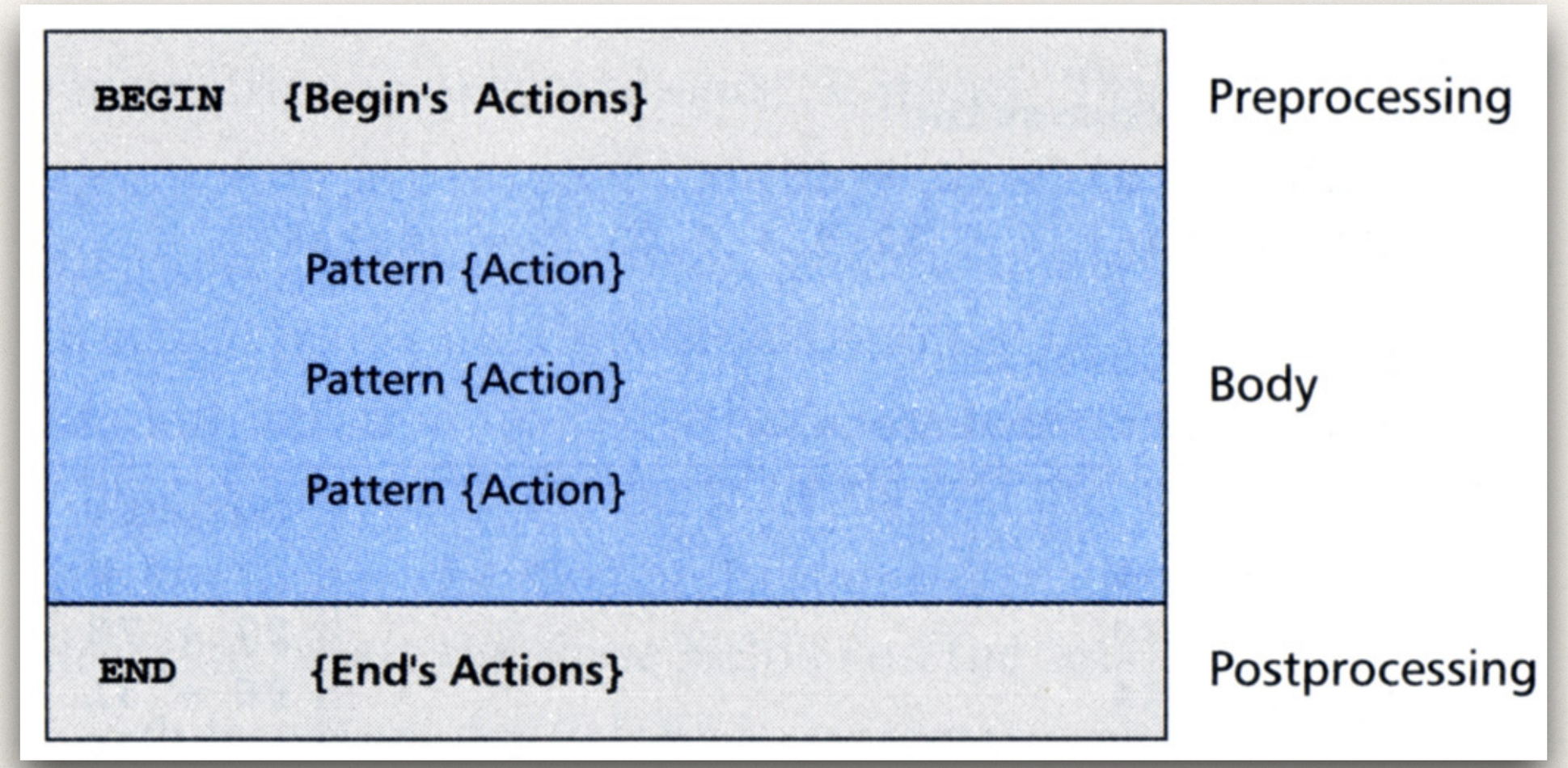
- User-defined variables
  - variable names
    - starts with letters, follows by letters/digits/underscores
  - do not need to be declared
  - initially created as strings
    - initialized to a null string
- awk built-in variables
  - Check out full list at
    - https://www.gnu.org/software/gawk/manual/gawk.html#Built_002din-Variables or
    - ftp://ftp.gnu.org/old-gnu/Manuals/gawk-3.0.3/html_chapter/gawk_11.html
  - awk's full manual can be found at
    - https://www.gnu.org/software/gawk/manual/

awk Variables: System Variables

Variable	Function	Default
FS	Input field separator	space or tab
RS	Input record separator	newline
OFS	Output field separator	space or tab
ORS	Output record separator	newline
NF[a]	Number of nonempty fields in current record	
NR[a]	Number of records read from all files	
FNR[a]	File number of records read—record number in current file	
FILENAME[a]	Name of the current file	
ARGC	Number of command-line arguments	
ARGV	Command-line argument array	
RLENGTH	Length of string matched by a built-in string function	
RSTART	Start of string matched by a built-in string function	

[a]Totally controlled by **awk.**

awk Scripts


BEGIN {Begin's Actions}
Pattern {Action}
Pattern {Action}
Pattern {Action}
END {End's Actions}
Preprocessing
Body
Postprocessing

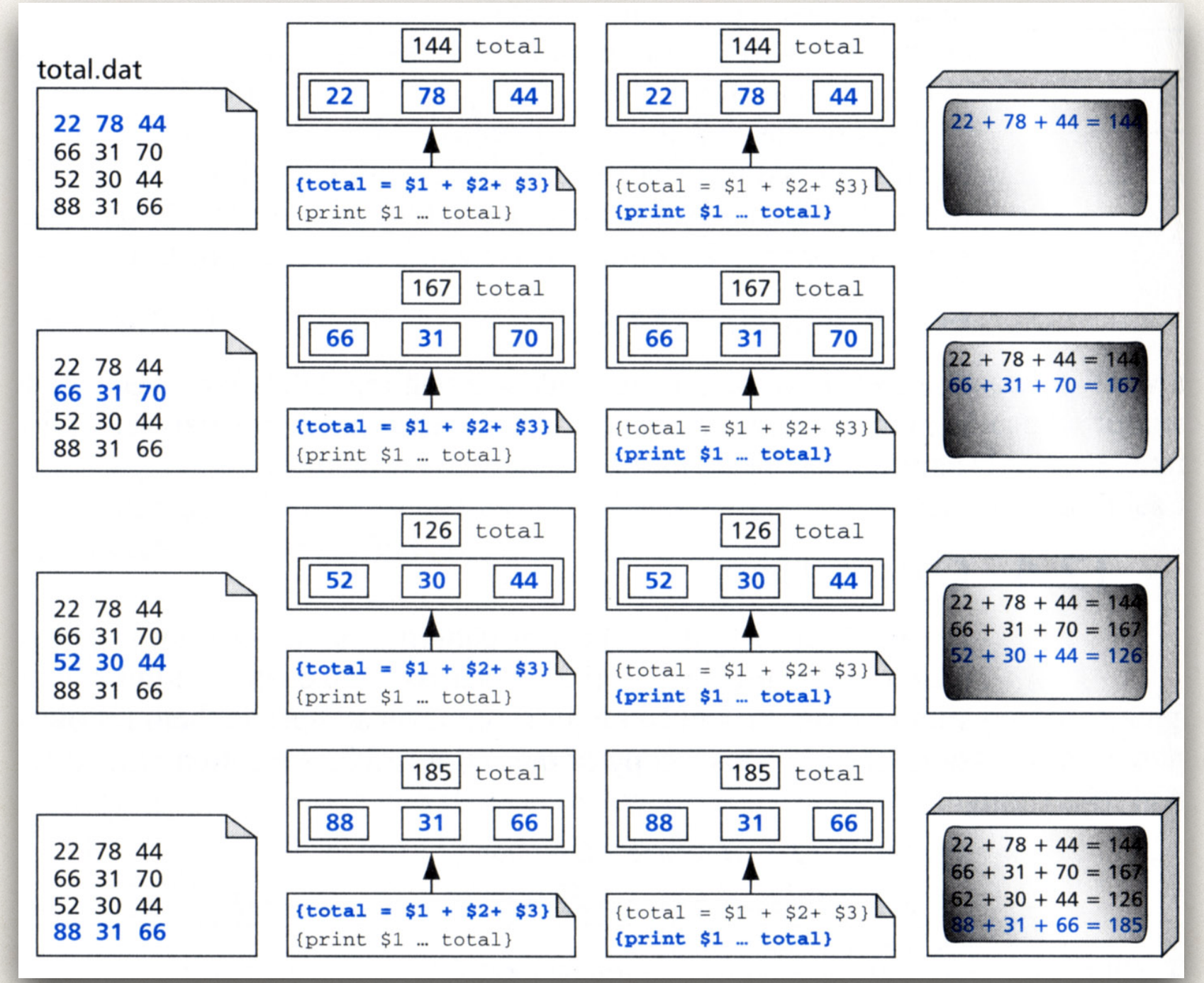
Operation: Example

- Command
  - awk –f total.awk total.dat
    - #Begin Processing
    - BEGIN {print "Print Totals"}
    - #Body Processing
    - {total = $1 + $2 + $3}
    - {print $1 " + " $2 " + " $3 " = " total}
    - #End Processing
    - END {print "End Totals”}
  - Input file (total.dat)
    - 22 78 44
    - 66 31 70
    - 52 30 44
    - 88 31 66

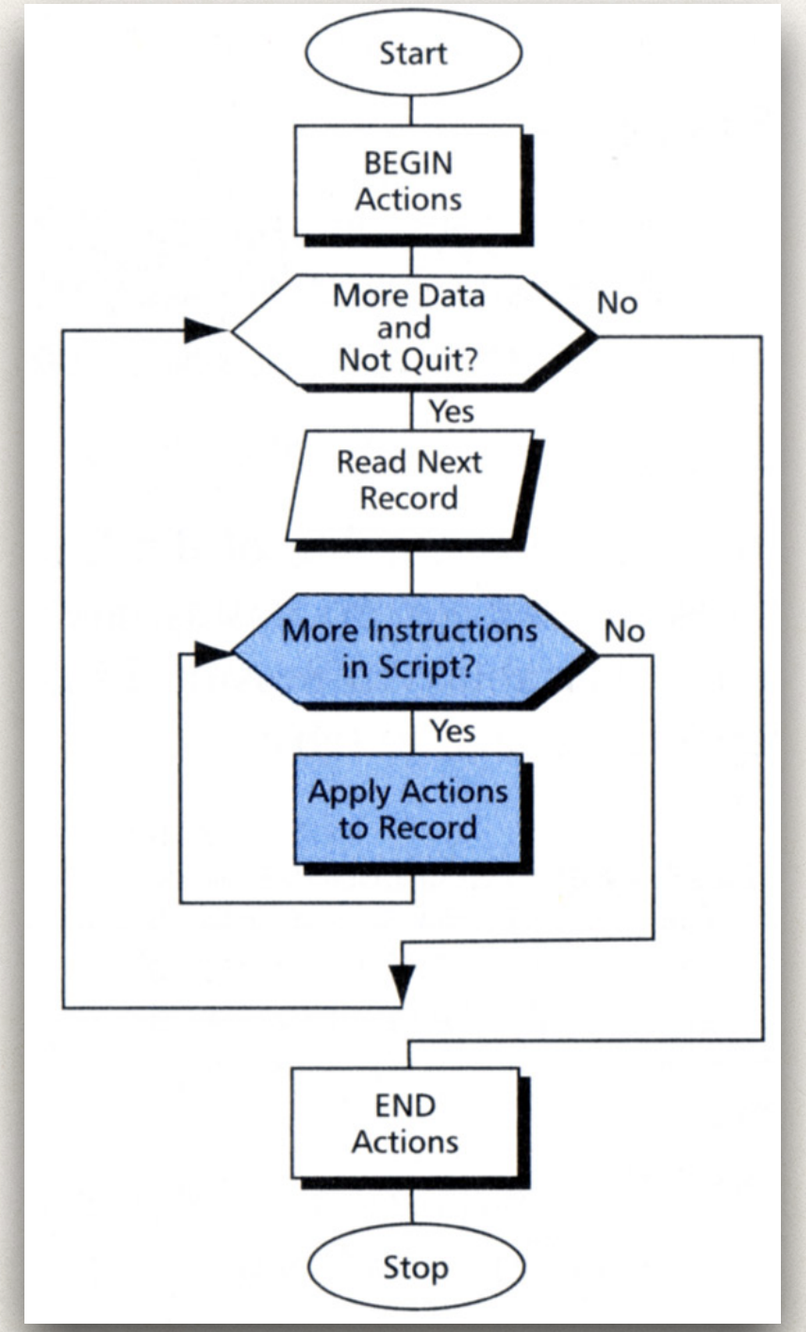
Output

```
Print Totals
22 + 78 + 44 = 144
66 + 31 + 70 = 167
52 + 30 + 44 = 126
88 + 31 + 66 = 185
End Totals
```

Operation: Example

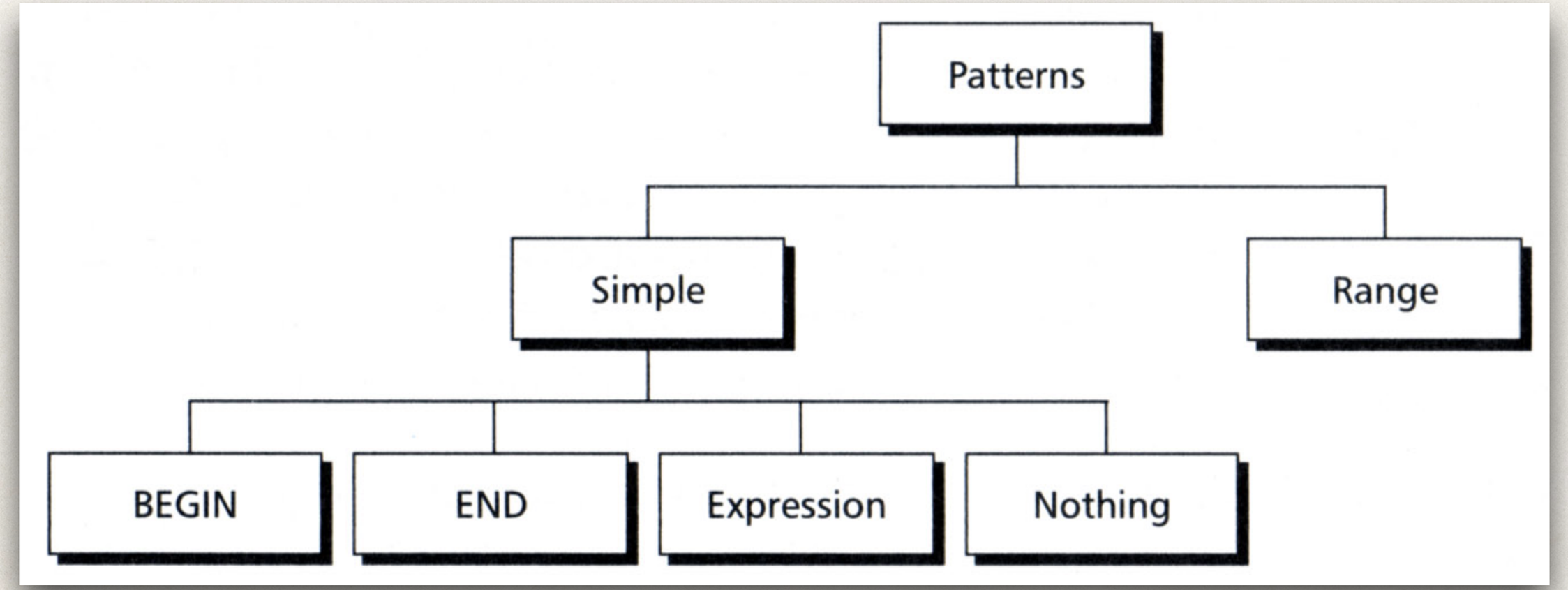


awk: Operation



Patterns

- Pattern identifies which records in the file are to receive an action

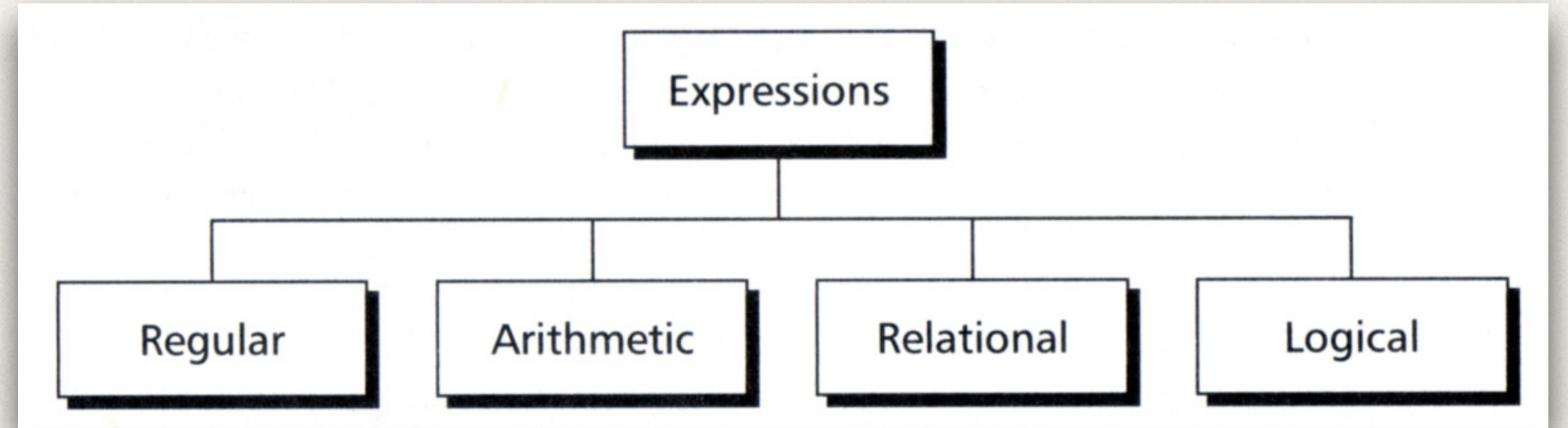


BEGIN and END

- BEGIN
  - BEGIN is true at the beginning of the input file, before the first record is read
  - Used to initialize the script before processing any data

- END
  - END is used at the conclusion of the script
  - Both are optional and can be placed anywhere in a script file

Expressions



Regular Expression

- The same as RE defined in egrep
- RE in awk
  - enclosed in /slashes/
  - requires one of two operators
    - `~` match operator
    - `!~` not-match operator
  - No operator -> RE could match anywhere in the input line
- Examples
  - `$0 ~ /^A.*B$/`
  - `$3 !~ /^ /`
  - `$4 !~ /bird/`

Arithmetic Expressions

- Match the record if the result of arithmetic expression is nonzero (true)
- Example: `$3 - $4 { print }`

Operator	Example	Explanation
`* / % ^`	`a^2`	Variable a is raised to power 2 (a^2).
`++`	`++a a++`	Adds 1 to a.
`--`	`--a a--`	Subtracts 1 from a.
`+ -`	`a + b, a - b`	Adds or subtracts two values.
`+`	`+a`	Unary plus: Value is unchanged.
`-`	`-a`	Unary minus: Value is complemented.
`=`	`a = 0`	a is assigned the value 0.
`*=`	`x *= y`	The equivalent of `x = x * y`; x is assigned the product of `x * y`.
`/=`	`x /= y`	The equivalent of `x = x / y`; x is assigned the quotient of `x / y`.
`%=`	`x %= y`	The equivalent of `x = x % y` where '%' is the modulo operator; x is assigned the modulus of `x / y`.
`+=`	`x += 5`	The equivalent of `x = x + 5`; x is assigned the sum of x and 5.
`-=`	`x -= 5`	The equivalent of `x = x - 5`; x is assigned the difference `(x - 5)`.

Relational Expressions

- Comparing a string with a number
  - The number is converted to a string first
- Force converting a string to be numeric
  - by adding 0 to it
- Force converting a numeric to be string
  - by appending a null string ("") to it

Operator	Explanation
<	Less than
<=	Less than or equal
==	Equal
!=	Not equal
>	Greater than
>=	Greater than or equal

Relational Expressions: Example

```
$ awk '$2 == "computers" {print}' sales.dat
```

Input:

```
1       clothing        3141
1       computers       9161
1       textbooks       21312
2       clothing        3252
2       computers       12321
2       supplies        2242
2       textbooks       15462
```

Output:

```
1       computers       9161
2       computers       12321
```

Logical Expressions and Nothing

- Logical expressions
  - logical operators are used to combine two or more expressions
    - ! expr
    - expr1 && expr2
    - expr1 || expr2
- Nothing
  - with no pattern, awk applies action to every input line

Logical Expressions: Example

```
$ awk '$2 == "computers" && $3 > 10000 {print}' sales.dat
```

Input:

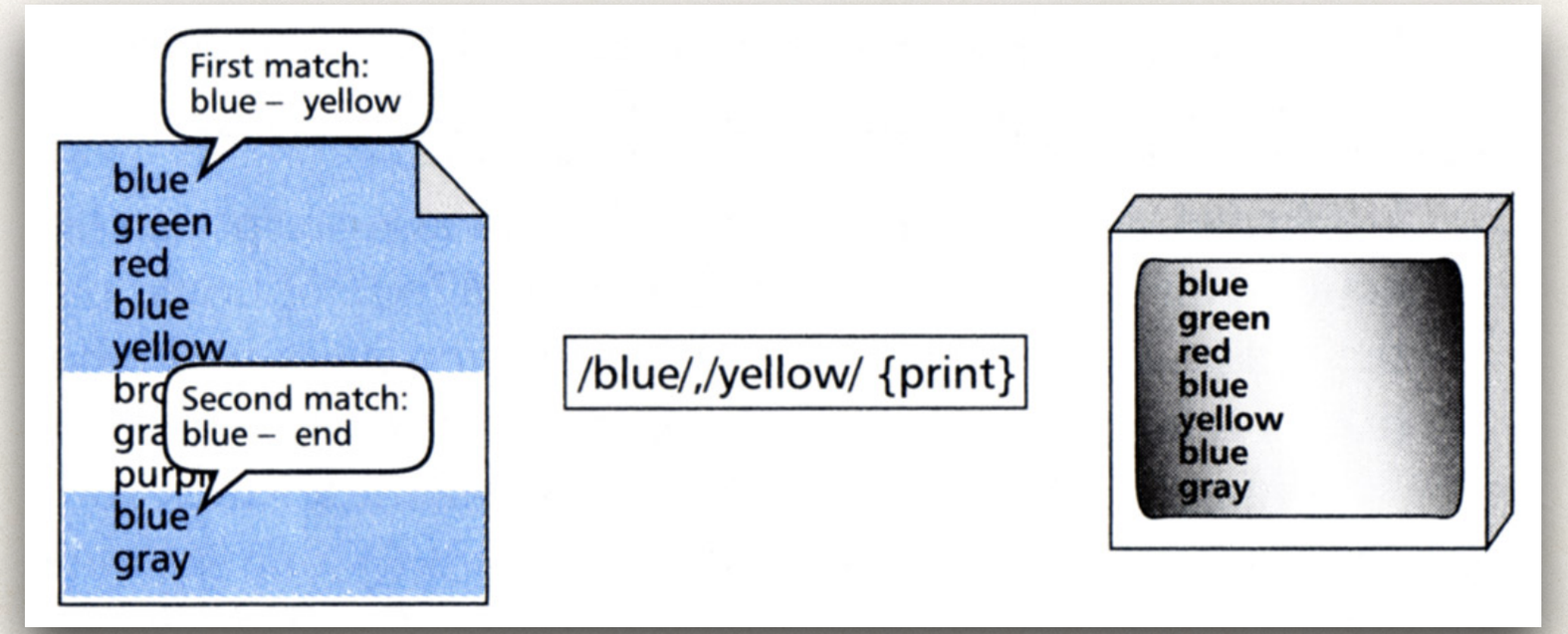
```
1       clothing        3141
1       computers       9161
1       textbooks       21312
2       clothing        3252
2       computers       12321
2       supplies        2242
2       textbooks       15462
```

Output:

```
2       computers       12321
```

Range Patterns

- A range pattern is associated with a range of records or lines
- Format
  - start-pattern, end-pattern
- Example
  - `awk 'NR == 8, NR == 13 {print NR, $0}' file1`



Actions

- Introduction
  - Similar to instructions/statements in programming languages (C)
  - But only act when the pattern is true
  - All C capabilities are incorporated 😄 / 😭
- Action
  - One or more statements associated with a pattern
  - Action statements must be enclosed in a pair of braces
  - Multiple statements are separated by
    - newline, semicolon, or a pair of braces (block)

Actions

```
pattern {statement}
```

(a) One Statement Action

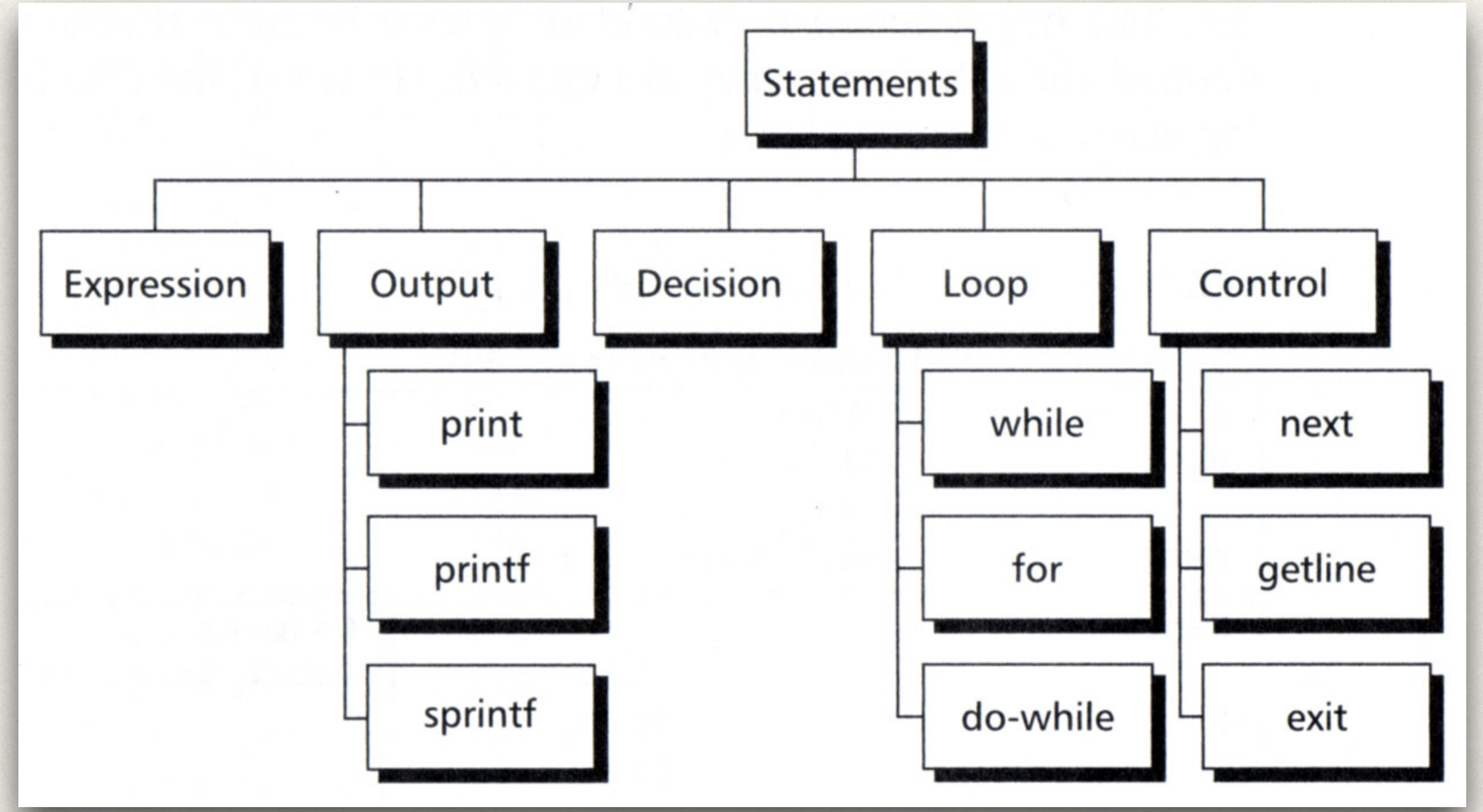
```
pattern {statement1; statement2; statement3}
```

(b) Multiple Statements Separated by Semicolons

```
pattern
{
    statement1
    statement2
    statement3
}
```

(c) Multiple Statements Separated by Newlines

Basic awk Statements



Statements: Expression and Output

- Expression
  - Same as in patterns
  - The value of expression is ignore
  - Examples
    - {$2 = 6}, {total += ($6 + 4)}
- Output
  - print
    - print – print the whole record
    - print $1, $2 – print specific values
  - printf and sprintf
    - Similar to those in C

Statements: Example

```
$ awk -f totalSales.awk sales1.dat
```

```
# total Sales script
BEGIN {total = 0}
      {total += $3}
END   {print "Total Sales", total}
```

Input:

```
1       clothing        3141
1       computers       9161
1       textbooks       21312
2       clothing        3252
2       computers       12321
2       supplies        2242
2       textbooks       15462
```

Output:

```
Total Sales 66891
```

Statements: Example

```
$ awk '{print}' sales2.dat
```

```
Output:
1       clothing        3141
1       computers       9161
1       software        3141
1       supplies        2131
1       textbooks       21312
1       sporting        0
2       clothing        3252
2       computers       12321
2       software        3252
2       supplies        2242
2       textbooks       22452
2       sporting        2345
3       clothing        3363
3       computers       13431
3       software        3363
3       supplies        2353
3       textbooks       23553
3       sporting        4554
```

Statements: Examples

```
$ awk 'BEGIN {OFS = "\t"}; {print $1, $2, $3}' sales2.dat | head -5
Output:
1       clothing        3141
1       computers       9161
1       software        3141
1       supplies        2131
1       textbooks       21312
```

```
$ awk '{print $1, $2, $3}' sales2.dat | head -5
Output:
1 clothing 3141
1 computers 9161
1 software 3141
1 supplies 2131
1 textbooks 21312
```

Statements: Examples

```
$ awk '{printf("%2d %-12s $%9.2f\n", $1, $2, $3)}' sales2.dat | head -5
```

```
Output:
 1 clothing     $  3141.00
 1 computers    $  9161.00
 1 software     $  3141.00
 1 supplies     $  2131.00
 1 textbooks    $ 21312.00
```

```
$ awk -f sprintf.awk sales2.dat
```

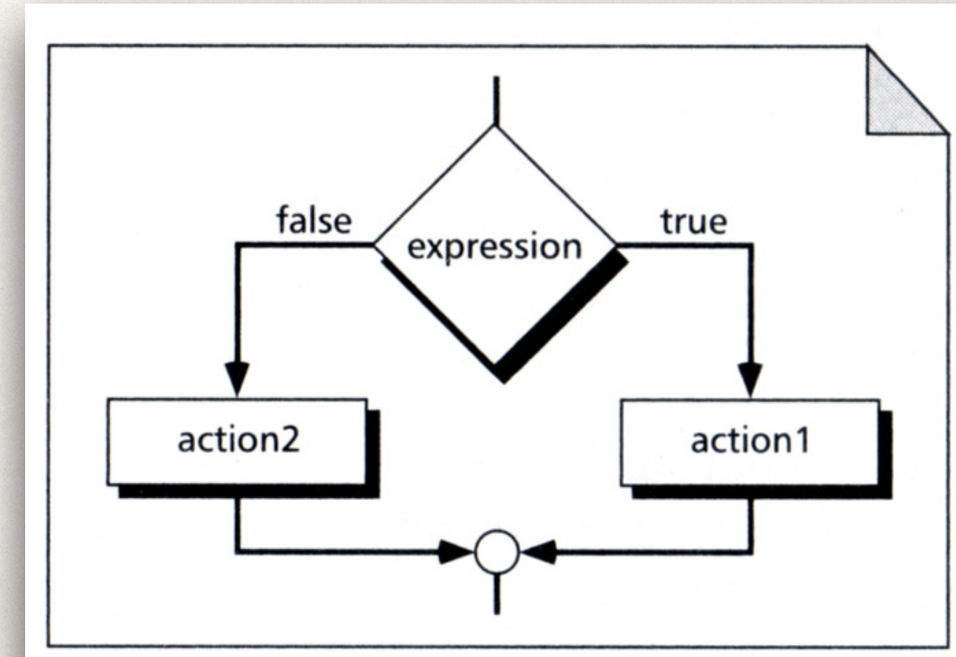
```
# sprintf.awk script
# Demonstrate sprintf command
NR == 1 {
        str = sprintf("%2d %-12s $%9.2f\n", $1, $2, $3)
        len = length(str)
        print len " " str
       }
```

```
Input:
1       clothing        3141
1       computers       9161
1       textbooks       21312
2       clothing        3252
2       computers       12321
2       supplies        2242
2       textbooks       15462
```

```
Output:
27  1 clothing     $  3141.00
```

Decision Statements

- if-else
  - Similar to C




```
if  ( expression )
    action1
else
    action2
```

Decision Statements: Example

```
$ awk -f compSalesAvg.awk sales1.dat
```

```
# compSalesAvg.awk
{
 if ($2 == "computers")
    {
     compSales += $3
     numMons++
    } # if computers true
} # end if

 END {
     if (compSales / numMons > 10000)
          print "Good sales in computers: $", compSales
      else
          print "Time for a pep-talk: $", compSales
     } # END
```

Input:

```
1       clothing        3141
1       computers       9161
1       textbooks       21312
2       clothing        3252
2       computers       12321
2       supplies        2242
2       textbooks       15462
```

Output:

```
Good sales in computers: $ 21482
```

Control Actions: next

- Next
  - Stop processing the current record
  - Start processing the next record from the beginning of the script

```
$ awk -f averageSales.awk sales1.dat
# averageSales script
 $3 == 0 {next}
 # Have non-zero sale
 {
  total += $3
  count++
 } # Non-zero sales
 END {avrg = total / count
      printf("Total sales    : $%9.2f\n", total)
      printf("Number of sales: %7d\n",    count)
      printf("Average sales  : $%9.2f\n", avrg)
     } # END
```

Input:

```
1       clothing        3141
1       computers       9161
1       textbooks       21312
2       clothing        3252
2       computers       12321
2       supplies        2242
2       textbooks       15462
```

Output:

```
Total sales    : $66891.00
Number of sales:       7
Average sales  : $ 9555.86
```

Control Actions: getline

- getline
  - Stop processing the current record, read the next record
  - Start processing the next record with the rest of the script
  - Note
    - Input can be directed to $0 – by default
    - Input can be directed to a separate variable
    - Returns a value that can be evaluated
      1 – if record is succesfully read
      0 – end of file
      -1 – read error
    - Input can be read from another file
  - Syntax
    - getline variable < other-input-file
      Ex: getline var < “filename”

Control Actions: getline - Example

```
$ awk -f exchange.awk 5lines.dat
# exchange.awk script
# exchanges lines 2 by 2
{
 if ((getline evenLine) == 1)
     {
      print evenLine
      print $0
     } # if getline
 else
      print $0
}
```

Input:	Output:
line 1	line 2
line 2	line 1
line 3	line 4
line 4	line 3
line 5	line 5

Control Actions: exit

- exit
  - Terminate the script and start executing the end statements
  - If exit is in the end pattern, the script exits immediately
  - Should be used only in an error condition
  - Return the value specified

```
$ awk -f salesZeroDiv.awk zeroSales.dat
# salesZeroDiv.awk script
# with compare complemented for test

$3 == 0 {next}

{total += $3
 print $1 $2 $3
 count++
} # Not zero sales

END     {
         if (count == 0)
            {
              printf("No sales to average\n")
              exit 100
            }  # end if
         avrg = total / count
         printf("Total sales    : $%9.2f\n", total)
         printf("Number of sales: %7d\n",    count)
         printf("Average sales  : $%9.2f\n", avrg)
        }  # end END block
```

```
Input:
1       clothing        0
1       computers       0
1       textbooks       0
```

```
Output:
No sales to average
```

Loops

- All 3 of C loops are available in awk
  - while
  - for
  - do-while

Loops: while

```
$ awk -f stuWhile.awk students.dat
```

```
# stuWhile.awk script
{
 total = 0
 count = 0
 i     = 2

 while (i <= NF)
     {
      total += $i

      count++
      i++
     } # while

# test for zero divide
 if (count > 0)
    {
     avrg = total/count
     print ($1, avrg)
    } # zero divide test
} # body
```

Input:

```
1234    87      83      91      89
2345    71      78      83      81
3456    93      97      89      91
4567    81      82      79      89
5678    78      86      81      79
```

Output:

```
1234 87.5
2345 78.25
3456 92.5
4567 82.75
5678 81
```

Loops: for

```
# for loop example
{
 total = 0
 count = 0
 for (i = 2; i <= NF; i++)
     {
      total += $i
      count++
     } # for
} # end of student scores
# test for zero divide
count > 0 {
            avrg = total/count
            print ($1, avrg)
           } # zero divide test
# end
```

Loops: do-while

```
# do while example
{
 total = 0
 count = 0
 i     = 2
} # initialization

 NF > 1 {
        do
            {
             total += $i
             count++
             i++
            } # do body
        while (i <= NF)

        avrg = total/count
        print ($1, avrg)
       } # NF > 1
# end script
```

Associative Arrays

- awk arrays are known as associative arrays
  - awk uses strings for array indexes
  - Index entry is associated with the array element
  - Example: an array A
    - Index: "Rob", "George", "Jan", "Jone"
    - Data: 23, 19, 20, 20
    - A["Rob"] = 23, A["George"] = 19, ...
    - awk –f awk02-ex01.awk name-age.dat

Associative Arrays

- Design constraints
  - Index must be unique, but data may be duplicated
  - No ordering imposed on the indexes
  - Array index cannot be sorted
  - Data values in the array can be sorted

Array Loops 1

- for...in
  - Format
    - for (index_variable in array_name)
  - Example
    - awk –f awk02-ex02.awk 5lines.dat
    - awk –f awk02-ex03.awk sales1.dat

Array Loops 2

- Check whether an index exists
  - if ("magazine" in deptSales)
    - Return true if magazine is an index
    - if ($2 in deptSales)
    - Example

```
awk –f awk02-ex04.awk <one-column-textfile>
```

- Delete array entry
  - delete array_name[index]
    - Example

```
awk –f awk02-ex05.awk sales1.dat
```

String Functions 1

- Length
  - Return the number of characters in the string parameter
  - Format
    - length (string)
  - Example
    - awk –f awk02-ex06.awk <textfile>
- Index
  - Returns the first position of a substring within a string, or 0 (zero) if not found
  - Format
    - index (string, substring)
  - Example
    - awk –f awk02-ex07.awk

String Functions 2

- Substring
  - Extract a substring from a string
  - Format
    - substr (string, position)
    - substr (string, position, length)
  - Example
    - awk –f awk02-ex08.awk
    - awk –f awk02-ex09.awk
- Split
  - Copy fields in a string into an array
  - Format
    - split (string, array)
    - split (string, array, field_separator)
  - Example
    - awk –f awk02-ex10.awk phoneBook.dat

String Functions 3

- Substitution
  - Substitute one string value for another
  - Returns true (1) if success, otherwise false (0)
  - Format
    - sub (regexp, replacement_string, in_string)
      If omitted, in_string = $0
  - Example
    - awk –f awk02-ex11.awk upx-1.24.README

- Global substitution
  - gsub (regexp, replacement_string, in_string)
  - Returns the number of substitutions made

String Functions 4

- Match
  - Returns the starting position of the matching expression in the line, or 0 if no matching
  - Set 2 system variables
    - RSTART – starting position
    - RLENGTH – length of the matching text string
  - Format
    - Startpos = match (string, regexp)
  - Example
    - awk –f awk02-ex12.awk /usr/doc/Cygwin/upx-1.24.README

Math Functions

- rand()
- srand(seed)
- cos(x)
- exp(x)
- log(x)
- sin(x)
- sqrt(x)
- More → http://www.gnu.org/software/gawk/manual/gawk.html#Numeric-Functions

User-defined Functions

- Format

  function name(parameter-list)

  { code }

- Example
  - awk –f awk02ex13.awk larger.dat

- More details and reference
  - http://www.gnu.org/software/gawk/manual/gawk.html
  - https://www.linuxlinks.com/best-free-books-to-learn-about-awk

Awk Command-line Arguments

- ARGC – argument count
  - not including options to awk
- ARGV – array of cmdline argument
  - index starts from 0 to (ARGC-1)
- Demonstration
  - cmd_argument.awk
  - awk -f cmd_argument.awk a b c d e f 'uppercase G'

```
BEGIN{
   print "ARGC=" ARGC;
   for (i=0; i<ARGC; i++) {
      printf "command line argument #%d = '%s'\n",i,ARGV[i];
   }
}
```

Awk Command-line Variables

- Option -v
  - used to send values to variables inside awk script
- Demonstration
  - awk -f cmd_variable.awk -v a=500 -v b="clock rate" a b c d

```
BEGIN{
    print a;
    print b;
    print "ARGC=" ARGC;
    for (i=0; i<ARGC; i++) {
       printf "command line argument #%d = '%s'\n",i,ARGV[i];
    }
}
```