9. 스크립트 프로그래밍

스크립트 프로그램

- 셸 프로그램
  - 명령어들을 다양한 형태로 결합하여 실행할 수 있도록 함
  - 일반적인 프로그램 작성에는 적합하지 않음
- 스크립트(script) 언어
  - 컴파일하지 않고 인터프리터 방식으로 실행
  - 컴파일을 하는 고급언어보다 사용이 편리함
  - (예) Perl, Python 등
- 스크립트 처리 기능이 포함된 유틸리티
  - awk – programmable 텍스트 처리
  - bc – calculator

9.1 awk – programmable 텍스트 처리

- awk
  - 주어진 condition을 만족하는 line에 대해서 지정된 동작(action)을 수행
  - 저자: Aho, Weinberger, Kernighan (AWK)
  - 사용형식
    $ awk '*command*' filename . . .
    $ awk –f *program* filename . . .

- awk program
  - 다음 형식의 명령어 리스트로 구성
    condition { action } // condition을 만족하는 행에 대해서 action 수행
    { action } // 조건 없음 – 모든 행에 대해서 action 수행
    condition // action 없음 - condition을 만족하는 모든 행을 출력
  - 한 줄의 명령어는 awk의 인수로 직접 지정가능
  - action에서 사용하는 구문은 C언어와 유사함

awk 내장 변수와 필드 접근

- 필드 – 각 행의 빈칸/탭으로 구분되는 단위
- 내장 변수

내장 변수	동작
$0	현재 행 전체
$*n*	현재 행의 n번째 필드 (n은 1이상의 정수)
NF	현재 행의 필드 개수
NR	현재 행의 번호
FS	필드 구분자
RS	행 구분자
OFS	출력 시의 필드 구분자
ORS	출력 시의 행 구분자
FILENAME	현재 처리 중인 파일이름

print 함수

- print : 출력 함수
  - 자동 줄 바꿈
  - 인수들을 컴마로 구분 → 인수 값들을 빈칸으로 구분하여 출력
  - 인수들을 빈칸으로 구분 → 인수 값들을 빈칸 없이 붙여서 출력

```
$ awk '{ print NR, NF, $0 }' song
$ awk '{ print NR $1 $NF }' song
        // NF는 필드 수, $NF는 마지막 필드 값
```


조건(condition)

- 조건의 형식

형태	동작을 수행하는 행
비교식	관계 연산자를 사용한 비교식을 만족하는 행
/패턴/	확장된 정규표현식 패턴과 일치하는 문자열이 포함된 행
조건1 && 조건2	두 조건을 동시에 만족하는 행
조건1 || 조건2	두 조건 중 적어도 하나를 만족하는 행
!조건	조건을 만족하지 않는 행
조건1, 조건2	(범위) 조건1을 만족하는 행부터 조건2를 만족하는 행까지의 행 조건2와 일치하는 행이 없으면 끝까지
BEGIN	입력을 읽기 전에 동작을 수행 (주로 초기화에 사용)
END	입력을 읽은 후에 동작을 수행 (주로 결과 출력에 사용)

  - 비교식은 C언어 형식과 유사
  - 패턴은 extended regular expression 사용

조건(condition) - 예

- 3행에서 6행사이의 행
  - $ awk ' NR>=3 && NR<=6 { print NR, $0 }' song
  - $ awk ' NR==3, NR==6 { print NR, $0 }' song
- /you/를 포함한 행
  - $ awk '/you/' song // no action – 행을 전부 출력
  - awk '/you/ { print FILENAME ":", $0 }' song // 파일이름 출력
- /you/를 포함한 행부터 /I/를 포함한 행까지
  - awk '/you/,/I/' song
- 60 이하의 성적(필드 $2, $3, $4)을 포함한 행
  - $ awk '$2 < 60 || $3 < 60 || $4 < 60' score

변수

- 사용자 정의 변수
  - 기본적으로 0 (null 문자열) 으로 초기화 됨 (별도로 초기화 필요없음)
  - 0이 아닌 초기값을 사용하는 경우 **BEGIN 조건**에서 초기화 수행
    (cf) awk의 인수에서도 변수 초기값 지정 가능
- awk는 조건을 만족하는 행에 대해서 반복하여 수행
  - 사실상 반복문으로 동작 – 계산 작업 가능
  - 계산 결과는 **END 조건**에서 출력
- 예 – 파일의 줄과 단어 수를 계수하는 프로그램

"wc.awk"

```
BEGIN { print "Scanning file .. " }
{ line++; word += NF }              //모든 행에 대해서 반복 수행
END { print "lines =", line, "words =", word }
```

$ awk -f wc.awk song

연산자

- 연산자
  - 비트연산자를 제외한 C 언어의 연산자 지원
  - 다음의 연산자 추가 지원

연산자	동작
^ 또는 **	거듭제곱
space	문자열 연결
< <= > >= != ==	비교 연산자는 숫자 뿐 만 아니라 문자열 비교도 수행함. 문자열에 대해서는 사전식 순서로 크기 비교
문자열 ~ /패턴/	정규표현식 매칭, 문자열에 패턴에 매칭되는 문자열이 포함되면 참
문자열 !~ /패턴/	정규표현식 매칭의 부정
/패턴/	$0 ~ /패턴/ 과 같음
첨자 in 배열	첨자의 배열 첨자 멤버쉽
^= **=	복합할당연산자의 산술연산에 거듭제곱도 포함

예 – 성적처리 프로그램

- 개인별, 과목별 평균 계산

"score.awk" // score파일 : 이름, 수학, 물리, 화학성적

```
BEGIN   { printf "Name\tMath\tPhysics\tChem\tAvg\n" }
        { count++;
          math += $2; phy += $3; chem += $4;
          avg = ($2 + $3 + $4)/3;
          printf "%s\t%d\t%d\t%d\t%.1f\n", $1, $2, $3, $4, avg;
        }
END     { if (count>0) {
                math /= count; phy /= count; chem /= count;
                printf "(Avg)\t%.1f\t%.1f\t%.1f\n", math, phy, chem;
          }
        }
```

$ awk –f score.awk score

- printf 함수 – C언어처럼 첫째 인수를 형식지정에 사용

예 – awk에서 추가된 연산자 사용

- 예

$ awk 'BEGIN {print 3^5, ("program"~/gra/), ("program">"string"); }'

- BEGIN 조건에 대한 action만으로 구성된 프로그램 작성하면 텍스트 파일 처리 없이 계산만 수행하는 프로그램 작성 가능

제어구문

- 제어 구문

```
if (조건식) 문장 [ else 문장 ]
while (조건식) 문장
for (식; 조건식; 식) 문장
break
continue
next            ... 현재 행에 대한 처리를 중단하고 다음 행으로 이동
exit            ... 현재 입력에 대한 처리를 중단함
```

- next, exit는 awk에서 추가된 구문

예 - 제어구문

- "score2.awk" - 각 사람의 최고 성적 출력

```
BEGIN   { print "Name\tMax_Score" }
        {
          max = $2
          for (i=3; i<=NF; i++) {
                if ($i > max) max = $i;
          }
          print $1 "\t" max;
        }
```

$1 : 이름
$2 … : 성적

$ awk –f score2.awk score

- "next.awk" – next, exit 사용

```
/you/   { next; }
/round/ { exit; }
        { print NR, $0; }
END     { print "End of processing" }
```

$ awk –f next.awk song

배열

- 연관 배열(associative array) 지원
  - 첨자로 임의의 문자열 사용 가능
    - a[3] a["3"] a["apple"]
  - (예)
    - arr[3] arr[03] arr["3"] - 서로 같음
    - arr["03"] – arr[03]과 다름
    - color["red"]와 color[red]는 다름 – red는 변수 값을 의미
- 배열에 대한 제어구문
  - for (변수 in 배열) { // 변수에 배열의 첨자 문자열을 순서대로 할당
    …
    }
  - if (첨자 in 배열) { // 첨자가 배열의 첨자이면 참
    …
    }
  - delete 배열[첨자] // 지정된 첨자의 배열원소 제거

예 - 배열

- "arr.awk"

```
BEGIN {
        score["math"] = 50;
        score["physics"] = 80;
        score["chem"] = 90;
        score["project"] = "Pass";
        for (i in score) {
                print i "\t" score[i];
        }
      }
```

$ awk –f arr.awk

- "car.awk" – 회사별 차종 개수 계수

```
        { count[$1]++; }                    ... 회사이름을 첨자로 사용
END     {
          for (name in count) {
                printf "%-12s %d\n", name, count[name];
          }
        }
```

$ awk –f car.awk car

함수

- 내장 함수
  - 수학, 문자열, 난수발생, command 실행 함수 제공
  - 교과서 표 참조
- 사용자 정의 함수

```
function 함수이름 (인수, 인수, ...) {
        프로그램
}
```

- 예 – "func.awk" 교과서 참조
  - 내장함수와 사용자 정의 함수 모두 사용
  - 주석은 #로 시작

9.2 Perl

- Perl (Practical Extraction Report Language)
  - 텍스트 처리 편의와 보고서의 효과적 생성을 목적으로 만들어짐
  - C언어, shell과 유사한 구문 - 일반적인 프로그램 작성이 편리함
  - UNIX/Linux 기능 접근 용이
- Perl 프로그램 실행 방법 – 3가지
  1. $ perl
     프로그램 입력
     ^D
  2. $ perl program
  3. ① 셸 프로그램과 같이 1행에 perl 프로그램 경로 지정

```
#!/usr/bin/perl
print "Hello World\n";
```

  ② $ chmod +x hello.pl // 실행 허가권
  ③ $ ./hello.pl // 실행

자료와 변수

- 기본 자료
  - 숫자 – 정수/실수형 구분 없음.
    - 0x로 시작(16진수), 0으로 시작(8진수), 0b로 시작(2진수)
    - 읽기편의를 위한 _ (예) 0b1101_0001
    - 부동소수점수 : 1.35, 7.25e10
  - 문자열 – 문자/문자열 구분 없음.
    - “ ” 또는 ‘ ‘로 표기
    - \ escape 문자 : \n, \r, \t, \f, \b, \007, \x7f, \cD (control-D)
- 변수
  - 변수 이름에 항상 $로 시작 – 읽기, 저장에 모두 사용

```
$a = $a + 20
print “\$a = $a\n”          // $a는 변수값 출력, \$는 $ 출력
```


연산자

- 연산자
  - C언어 연산자 대부분 제공
- 추가 제공 연산자

연산자	동작
**	거듭제곱 (실수 연산도 가능)
.	문자열 연결 연산자 (예) "abc"."xy"는 "abcxy"와 같음)
x	문자열 반복 연산자 ("ab" x 5은 “abababababab”과 같음)
eq ne lt gt le ge	문자열 비교
=~ !~	정규표현식 매칭 ($a =~ /hat/와 $a !~ /hat/는 변수 $a에 패턴 hat가 포함되어 있는 지와 포함되어 있지 않은 지 검사)
<=>	두 숫자 비교 (크기에 따라서 결과가 -1, 0, 1 중 하나임)
cmp	두 문자열 비교 (크기에 따라서 결과가 -1, 0, 1 중 하나임)
not and or xor	논리 연산자 (우선순위가 = 보다 낮음)

제어 구문

- 제어구문
  - if – else, while, for, do – while 구문 – C언어와 같은 형식
  - if 와 반복문에 대한 문장이 1개이어도 반드시 { }를 사용함
- 추가 제공 제어구문
  - if – **elsif** … else
  - until
  - do – until
  - foreach // 배열 원소에 대한 반복 처리에 유용
  - last : C의 break와 같은 의미
  - next : C의 continue와 같은 의미

배열과 리스트

- 배열

```
@arr = (1,2,3,4,5)              // 리스트 : ( , , … ) 형태의 값
@arr = (1..5)                   // 범위 연산자  사용
@arr = (-1, 1..5, 10)
@arr = (100, "apple", 3.5);     // 여러 유형의 자료가 섞임
```

- 배열 원소

```
@arr[0]  또는  $arr[0]          // 첫째 원소
@arr[4] 또는 $arr[4]            // 다섯째 원소
$#arr                           // 배열의 마지막 첨자
@arr[5] = 200                   // 배열 크기 자동 확장
```

- 배열의 스칼라 연산

```
@arr2 = @arr                    // 배열 복사
($a, $b, $c) = (100, 200, 300)
$c = scalar(@arr)               // scalar 함수 : 배열 크기 반환
$s = @arr                       // scalar 연산 시에 자동적으로 scalar 변환
$a = @arr + @arr2
```


명령어 행 인수, 배열에 대한 연산 및 제어구문

- foreach 문

```
foreach $n (1, 3, 5, 7) {
        print $n, "\n";
}
foreach (1. 10) {
        print $_, "\n";                 // $_는 기본 변수
}
```

- 명령어 행 인수 : 배열 변수 @ARGV

```
foreach $arg (@ARGV) …
```

- 배열에 대한 함수
  - sort, reverse,
  - push, pop – 스택 연산
  - shift, unshift – 큐 연산

함수

- perl 함수 정의

```
sub 함수이름 {
    프로그램 ..
}
```

- perl 함수 호출

```
함수이름(인수 …)
```


파일 입출력

- 파일 핸들
  - STDIN, STDOUT, STDERR – 기본적으로 오픈되는 파일핸들
  - 일반적인 파일은 open 함수 이용하여 파일핸들 지정

```
open(handle, filename)
close(handle)
```

- 파일 입출력

```
print handle  "text"            // 파일 출력
$line = <handle>                // 줄 단위 입력
@array = <handle>               // 파일 끝까지 줄 단위 입력
```

- 파일 연산
  - test 명령어와 유사한 연산

정규표현식 연산

- 정규표현식 연산자

연산자	동작
str =~ /regex/	문자열에 정규표현식 패턴이 포함되면 참
str !~ /regex/	문자열에 정규표현식 패턴이 포함되지 않으면 참
변수 =~ s/regex/str/	변수의 문자열에서 정규표현식 패턴에 매칭되는 첫째 문자열을 지정된 문자열로 치환.

- 패턴매칭 문자열 추출

```
if ($s =~ /(.*)#(.*)#/) {
        print "$1\n";              ... # 앞의 문자열
        print "$2\n";              ... # 사이의 문자열
}
```


연관 배열 - hash

- 연관배열 – 문자열을 첨자로 사용 가능

```
%age = ("Kim" => 20, "Park" => 25, "Choi"=> 33, Lee => 50);
$a = $age{“Kim”}
$age{“Ahn”} = 100              // 추가
$age{“Kim”} =80                // 수정
```

- 연관 배열에 대한 함수

함수	동작
keys	해시 원소의 키들로 구성된 배열을 반환
values	해시 원소의 값들로 구성된 배열을 반환
each	해쉬 원소의 (키, 값) 쌍의 배열을 호출할 때마다 순서대로 반환
exists	지정된 원소가 존재하는 지 검사
delete	해시에서 지정된 원소를 제거

- 연관배열 %ENV – 환경변수들을 저장 (변수 이름이 첨자)

내장 함수

- 주요 내장 함수

함수	동작
chop(s)	문자열 s의 마지막 문자를 제거하여 반환
chomp(s)	문자열 s 마지막의 \n문자를 제거하여 반환
eval(s)	문자열 s로 나타낸 Perl 프로그램 문장을 실행
index(s, t, i)	문자열 s에서 주어진 위치 i부터 검색하여 특정 문자열 t가 포함된 위치를 반환. 위치가 생략되면 처음부터 검색.
split(/re/, s)	문자열 s를 정규표현식 re으로 지정된 구분자로 분리한 리스트 값 반환
substr(s, i, len)	문자열 s의 위치 i부터 길이 len의 부분 문자열 반환. 길이가 생략되면 문자열 끝까지 반환
length(s)	문자열 s의 길이를 반환
sysem(s)	문자열 s로 나타낸 명령어를 셸에서 실행