Praktikum 4
Membaca Input File

Tujuan Pembelajaran

Mahasiswa dapat membaca dan memanipulasi data dari input file dalam bahasa pemrograman awk.

Dasar Teori

Pada program awk secara umum, semua input dibaca dari standard input (default-nya keyboard, tapi dapat juga menggunakan perintah *pipe* dari perintah lain) atau dari files yang namanya diberikan pada awk command line. Jika diberikan input file, awk membacanya secara urut, memproses semua data dari file sebelumnya ke file berikutnya.

Nama dari file yang sedang dibaca dapat ditemukan dalam built-in variable `FILENAME`. Input yang dibaca dalam unit disebut *records*, dan diproses dengan aturan pada program yang telah ditentukan, satu record dalam sekali baca. Default-nya, setiap satu record adalah satu baris. Setiap record secara otomatis dibagi kedalam potongan yang disebut *field*. Hal ini mempermudah program untuk bekerja pada bagian dari record. Record dalam file input ditentukan dengan *record separator* yang dapat diinisialisasi kedalam variable RS. Nilai default dari variabel RS adalah karakter *newline*, sehingga satu baris merupakan satu record. Record dibagi ke dalam field yang ditentukan dengan *field separator* yang diinisialisasikan dalam variabel FS. Nilai default FS adalah karakter *whitespace*.

Separator tersebut dapat berupa karakter, string, maupun regular expression. Separator dapat diubah dengan cara menginisialisasi ulang. Field dalam record ditunjuk dengan menggunakan notasi tanda dolar '$' diikuti nomor field. Nomor field dapat direpresentasikan dengan nilai angka (konstanta), variabel, maupun operasi aritmatika. Input memiliki beberapa properties yang tersimpan dalam built-in variable antara lain NR (*number of record*) menyatakan jumlah record dalam satu file yang telah dibaca, dan NF (*number of field*) menyatakan jumlah field yang ada dalam setiap record.

Pada kesempatan lainnya, perlu juga untuk menggunakan perintah "*getline*". Perintah tersebut sangat berguna, karena selain dapat memberikan input yang jelas dari sejumlah file, juga file yang digunakan tidak harus dinamai pada awk command line.

Percobaan 1: Bagaimana membagi input menjadi *record*.

Utilitas awk membagi input untuk awk program menjadi bagian *record* dan *fields*. Awk menyimpan jumlah *record* yang telah dibaca dari file input. Nilai tersebut disimpan dalam built-in variable yang disebut `FNR`. Variable tersebut nilainya di-reset menjadi nol ketika membaca file baru. Built-in variable lain adalah `NR` artinya total jumlah record yang telah dibaca dari semua file data. `NR` dimulai dari 0, tapi tidak reset menjadi 0 secara otomatis.

Record dipisahkan oleh sebuah karakter yang disebut *record separator* (`RS`). Default-nya, *record separator* adalah karakter *newline* (baris baru). Inilah mengapa *record* (defaultnya) berupa satu baris. Karakter lainnya dapat digunakan sebagai *record separator* dengan meng-*assign* suatu karakter ke built-in variable `RS`.

Seperti variable lainnya, nilai `RS` dapat diubah dalam awk program dengan operator *assignment* '='. Karakter *record separator* baru harus diapit oleh tanda petik, yang menandakan sebagai konstanta string. Sering kali waktu yang tepat untuk melakukan hal ini adalah saat awal eksekusi, sebelum mengambil input untuk diproses, sehingga record akan terbaca dengan separator yang benar. Untuk mengubah *record separator* dapat dilakukan dengan perintah berikut:

```
$ awk 'BEGIN { RS = "/" }
       { print $0 }' BBS-list
```

Pada command di atas, nilai record separator (`RS`) diubah sebelum membaca input. `RS = "/"` akan meberi nilai `RS` berupa string yang diawali dengan karakter slash (/). Dengan itu, maka record dari input file akan dipisahkan berdasarkan karakter slash. Pada baris pertama file BBS-list:

```
Aardvark    555-7685    1200/300    B
```

maka record yang diperoleh dari input tersebut dengan `RS = "/"` adalah:

```
aardvark    555-7685    1200
300   B
```

String 300 dianggap sebagai *record* baru karena sebelumnya diawali oleh karakter "/" yang merupakan *record separator*.

Record separator juga dapat ditentukan melalui command line yaitu dengan menggunakan fitur *variable assignment* seperti pada perintah berikut:

```
$ awk '{ RS="/" ;print $0 }' BBS-list
```

Nilai RS tidak ditentukan dalam progam awk, melainkan melalui command line. Nilai RS di isi dengan karakter slash (/) sebelum memproses BBS-list. Dari hasil percobaan akan tampak kedua cara tersebut menghasilkan output yang berbeda. Disarankan untuk menggunakan karakter yang tidak biasa dipakai seperti slash (/) sebagai record separator karena akan membiasakan pengguna untuk teliti dalam beberapa kasus.

NF adalah variable built-in yang menyimpan jumlah field dalam record yang sedang dibaca. Nilai dari NF adalah jumlah dari field yang ada pada record tersebut. Jika ada satu field yang memiliki sebuah newline, awk akan memperlakukan *newline* seperti *whitespace*, dan akan mencetak '0' sebagai hasilnya. Seperti pada baris perintah dibawah ini:

```
$ echo | awk 'BEGIN { RS="x" } ; { print NF }'
$ echo x | awk 'BEGIN { RS="x" } ; { print NF }'
$ echo x a | awk 'BEGIN { RS="x" } ; { print NF }'
$ echo x a x | awk 'BEGIN { RS="x" } ; { print NF }'
$ echo x a x a b | awk 'BEGIN { RS="x" } ; { print NF }'
$ echo x a x a b x | awk 'BEGIN { RS="x" } ; { print NF }'
```

Perintah echo digunakan untuk menampilkan baris teks. Dari perintah echo di atas, jika argumen untuk instruksi echo tidak ada atau string kosong. Nilai nol yang keluar sebagai output tersebut menunjukkan nilai dari NF adalah 0, yang berarti bahwa tidak terdapat field. Jika terdapat satu record separator berarti ada 2 record. Satu field dengan field yang lainnya dipisahkan dengan *whitespace*.

Kalau sudah mencapai akhir dari input file akan mengakhiri input record saat ini, bahkan jika karakter yang terakhir dalam file tersebut bukanlah karakter di RS. String kosong "" (string tanpa karakter) memiliki arti khusus sebagai nilai dari RS. Ini berarti bahwa record dipisahkan oleh satu atau lebih baris kosong (*blank lines*) dan tidak ada yang lainnya. Jika Anda merubah nilai RS di tengah-tengah menjalankan awk, nilai yang baru tersebut akan digunakan untuk membatasi record berikutnya, tetapi record yang sedang atau sudah diproses tidak terpengaruh.

Setelah akhir dari record, awk akan memberikan nilai kepada variabel RT untuk teks input yang cocok dengan RS. Bila menggunakan gawk, nilai RS tidak terbatas pada string satu karakter. Hal ini dapat ditulis dengan regex (lihat Bab 3 *Regular Expressions*). Secara umum, setiap record berakhir pada string berikutnya yang cocok dengan regex, record berikutnya dimulai pada akhir string yang cocok. Ini adalah aturan umum yang berlaku pada kebanyakan kasus, di mana RS mengandung hanya satu *newline*. Maksudnya *record* berakhir pada awal pencocokan string berikutnya (baris baru berikutnya pada input), dan *record* berikutnya dimulai tepat setelah berakhirnya string tersebut (pada karakter pertama dari baris berikutnya). Baris baru akan muncul dikarenakan ada karakter yang cocok dengan RS, tetapi ini adalah *record* baru dan bukan bagian dari record yang lain.

Ketika RS berisi karakter tunggal, RT juga akan berisi karakter tunggal yang sama. Namun, ketika RS berisi regex, RT berisi input teks yang sebenarnya cocok dengan ekspresi reguler. Jika file input berakhir tanpa teks yang cocok RS, awk kan men-set RT berisi *null string*.

Contoh berikut ini akan menggambarkan kedua fitur. Perintah berikut akan memberikan nilai RS dengan regex yang akan mencocokan baik *newline* atau serangkaian dari satu atau lebih huruf besar (uppercase) dengan pilihan *leading* dengan atau *trailing whitespace*:

```
$ echo record 1 AAAA record 2 BBBB record 3 |
> gawk 'BEGIN { RS = "\n|(* [[:upper:]]+ *)" }
> { print "Record =", $0, "and RT =", RT }'
= Record = record 1 and RT = AAAA
= Record = record 2 and RT = BBBB
= Record = record 3 and RT =
```

Bila pada percobaan sebelumnya RS bernilai sebuah karakter tunggal, maka RT juga bernilai karakter tunggal yang sama. Namun pada percobaan diatas nilai RS diberikan dengan regular expression. Maka terdapat kemungkinan RT dari setiap record akan berbeda satu sama lain. Nilai RS yang diberikan adalah "\n|(*[[:upper:]]+*)". Regular expression tersebut akan cocok dengan setiap karakter *newline* (\n) atau berapapun karakter huruf kapital (uppercase). Sehingga dari input 1 AAAA record 2 BBBB record 3, maka RT untuk record 1 adalah AAAA, record 2 adalah BBBB, dan record 3 adalah karakter *newline*. Karakter tersebut tidak tampak namun ada pada akhir dari input tersebut.

Suatu ketika Anda ingin menjadikan semua data pada input file sebagai *single record.* Satu-satunya cara adalah dengan memberikan nilai RS dengan nilai yang Anda tahu tidak ada pada input file. Hal ini akan sulit dilakukan secara umum, sehingga program selalu bekerja sembarangan ketika bekerja dengan input file. Anda mungkin berpikir bahwa untuk file teks, karakter null, yang terdiri dari karakter dengan semua bit sama dengan nol (zero), adalah nilai yang baik untuk digunakan untuk RS dalam hal ini:

```
BEGIN {RS = "\0"}  # whole file becomes one record?
```

Berikut ini program yang menunjukkan seluruh file menjadi satu record.

```
$ echo | awk 'BEGIN { RS = "\0"}; {print NF}'
$ echo a | awk 'BEGIN { RS = "\0"}; {print NF}'
$ echo a b | awk 'BEGIN { RS = "\0"}; {print NF}'
```

Awk pada kenyataannya menerima hal ini, dan menggunakan karakter nul untuk pemisah record. Namun, penggunaan ini tidak *portabel* (mungkin tidak berlaku) untuk implementasi awk lainnya. Awk lainnya menyimpan string secara internal sebagai *C-style* string. *C string* menggunakan karakter null sebagai terminator

string. Akibatnya, ini berarti bahwa RS = "\ 0" adalah sama seperti RS = "". Seperti pada program berikut ini dengan menggunakan `RS = ""`.

```
$ echo | awk 'BEGIN { RS = "" }; {print NF}'
$ echo c | awk 'BEGIN { RS = "" }; {print NF}'
$ echo c d | awk 'BEGIN { RS = "" }; {print NF}'
```

Cara terbaik untuk menjadikan seluruh file sebagai single record adalah dengan hanya membaca file, satu record dalam satu waktu, menggabungkan setiap record menjadi satu record di akhir.

Percobaan 2: Memeriksa *fields*

Ketika awk membaca *input record*, *record* secara otomatis diurai atau dipisahkan oleh utilitas awk menjadi potongan-potongan yang disebut *fields*. Secara default, *field* dipisahkan oleh *whitespace*, seperti kata-kata dalam satu baris. *Whitespace* dalam awk berarti setiap string dari satu atau lebih SPACEs, TABs, atau *newlines*, 2 karakter lain, seperti *formfeed*, *vertical tab*, dll, yang dianggap oleh spasi bahasa lain, tidak dianggap oleh spasi awk.

Tujuan dari *fields* ini adalah untuk membuatnya lebih mudah bagi Anda untuk merujuk (*refers*) bagian-bagian dari *records*. Anda tidak harus menggunakannya-Anda dapat beroperasi pada seluruh *records* jika Anda ingin-tapi *fields* adalah apa yang membuat program awk sederhana begitu *powerful*.

Dalam program awk, notasi dolar-sign ('$') digunakan untuk menunjuk suatu *field*. Notasi $ diikuti oleh angka yang menunjukkan nomor *field*. Jadi $1 artinya field pertama, $2 artinya field kedua, dan seterusnya. Contohnya ada baris input record seperti dibawah ini:

```
This seems like a pretty nice example.
```

Field pertama ($1) adalah "This", field kedua ($2) adalah "seems", field ketiga ($3) adalah "like", field keempat ($4) adalah "a", field kelima ($5) adalah "pretty", field keenam ($6) adalah "nice", field ketujuh ($7) adalah "example." (karena karakter 'e' dan '.' tidak dipisahkan spasi maka dianggap satu field).

`NF` adalah variabel built-in yang nilainya adalah jumlah field dalam record yang aktif. Awk secara otomatis mengupdate nilai `NF` setiap kali membaca record. Tidak peduli berapa banyak field yang ada, field terakhir dalam record dapat diwakili oleh `$NF`. Jadi, `$NF` adalah sama dengan field ketujuh ($7) yaitu "example." seperti pada contoh diatas. Jika Anda ingin mencoba untuk mengacu (*refer*s) field yang terakhir (seperti $8 pada contoh diatas, ketika *record* hanya memiliki 7 *record*), Anda akan mendapatkan string kosong. (Jika digunakan dalam operasi numerik, Anda akan mendapatkan nol.)

Penggunaan $0 digunakan untuk merepresentasikan keseluruhan input record, digunakan saat Anda sedang tidak membutuhkan field yang spesifik. Berikut ini contohnya percobaan dengan perintah `print $0`:

```
$ awk '$1 ~ /foo/{ print $0 }' BBS-list
> fooey     555-1234    2400/1200/300     B
> foot      555-6699    1200/300          B
> macfoo    555-6480    1200/300          A
> sabafoo   555-2127    1200/300          C
```

Tampak ouputnya adalah hasil cetak dari keseluruhan field satu record yang mengandung string "foo". Hal tersebut menunjukkan bahwa $0 bukan menunjuk field ke-0 melainkan menunjuk semua field pada satu record. Coba bandingkan dengan program berikut ini:

```
$ awk '/foo/ { print $1,$NF }' BBS-list
> fooey     B
> foot      B
> macfoo    A
> sabafoo   C
```

Pada percobaan diatas, $1 menunjuk field ke-1, sehingga pencarian untuk regex 'foo' di lakukan pada field ke-1. `NF` menyimpan jumlah field dari record yang sedang dibaca. Satu record dalam input file BBS-list terdiri dari 4 field, maka nilai `NF` adalah 4. Sehingga $`NF` = $4, atau menunjuk field terakhir dari record terbut. Dengan perintah `print $1,$NF`, maka record yang cocok akan dicetak field pertama dan dan terakhir.

Percobaan 3: *Nonconstant Field Numbers*

Awk menyimpan jumlah field setiap record dalam built-in variabel `NF`. Jumlah field (`NF`) tidak dapat menjadi konstanta. Setiap ekspresi dalam bahasa awk dapat digunakan setelah karakter '$' untuk menyatakan field keberapa. Nilai dari ekspresi tersebut menentukan jumlah field. Jika nilainya adalah sebuah string, bukan nomor, akan diubah ke nomor. Perhatikan contoh berikut ini:

```
$ awk '{ print $NF }' BBS-list
$ awk '{ print $NR }' BBS-list
```

`NR` adalah variable built-in yang menyimpan jumlah record yang telah dibaca. Sehingga untuk pembacaan suatu input file yang terdiri dari lebih dari satu record, nilai `NR` akan berubah (bertambah) dari pembacaan record pertama sampai terakhir. Perintah `awk "{print $NR}"` akan mencetak field dengan nomor yang ditunjuk oleh nilai `NR`. Untuk record pertama nilai `NR` adalah 1, sehingga untuk record pertama yang dicetak adalah field pertama. Untuk record ke-2 maka nilai `NR` adalah 2 sehingga field yang dicetak adalah field ke-2. Dan seterusnya. Setelah record ke-4 yakni record ke-5 dan setelahnya, tidak tercetak

kosong, karena jumlah field dalam satu record pada file input tersebut adalah 4. Hal ini juga menunjukkan bahwa notasi $ dapat diikuti oleh nilai variable.

```
$ awk '{ print $(2*2) }' BBS-list
```

Pada perintah tersebut, $(2*2) = $4, maka yang output yang tercetak dengan program awk di atas adalah field ke-4. Hal ini menunjukkan bahwa awk mengevaluasi ekspresi (2*2) dan menggunakan nilainya sebagai nomor field kan dicetak. Operasi perkalian tersebut diapit oleh tanda kurung agar operasi perkalian tersebut dikerjakan lebih dulu sebelum memprosesnya dengan notasi $. Jika yang dimasukkan di field numbernya adalah 0, maka yang dimaksudkan adalah seluruh kolom. Jadi kalau misalnya ada $(2-2) artinya sama dengan $0. Field number yang negatif tidak diijinkan.

Percobaan 4: Merubah isi dari sebuah *field*

Isi dari suatu field (dalam sudut pandang awk) dapat diubah dengan program awk, namun awk tidak mengubah file input (asli), melainkan hanya mengubah nilai yang diterima saja.

```
$ awk '{ nboxes = $3 ; $3 = $3 - 10
>        print nboxes, $3 }' inventory-shipped
= 25 15
= 32 22
= 24 14
```

Mula-mula program menyimpan nilai awal dari field-3 dalam variabel nboxes. Kemudian operasi $3 = $3 – 10, akan mengisi field-3 dengan nilai awal field-3 dikurangi 10. Perintah print nboxes, $3 untuk mencetak nilai awal field-3 dan nilai field-3 baru. Untuk record pertama nilai awal field-3 adalah 25, sehingga nboxes bernilai 25. Kemudian field-3 diisi dengan hasil operasi 25-10=15, sehingga field-3 sekarang berisi 15. Maka hasil output (print) dari record pertama adalah nboxes=25 dan $3=15.

String dari karakter diatas dikonversi secara langsung oleh awk menjadi “number” agar komputer dapat melakukan operasi aritmetika. Hasil dari opeasi diatas adalah “number” juga, kemudian dikonversi kembali menjadi “string of character” lalu menjadi field.

Ketika nilai dari field berubah (dalam sudut pandang awk), teks dari input record dihitung ulang agar field yang baru terisi dengan field sebelumnya. Dengan kata lain, $0 untuk mencerminkan perubahan field. Contohnya pada program berikut ini mencetak salinan dari file input, dengan nilai kolom ke-2 dikurangi 10 dari setiap barisnya:

```
$ awk '{ $2 = $2 - 10; print $0 }' inventory-shipped
```

```
= Jan 3 25 15 115
= Feb 5 32 24 226
= Mar 5 24 34 228
```

Program di atas mengisi field-2 dengan hasil dari operasi pengurangan nilai awal field-2 oleh 10. Kemudian mencetak record dengan printah `print $0`, sebagaimana telah diungkapkan bahwa $0 akan menunjuk pada semua field/satu record. Hal ini juga dimungkinkan juga untuk menetapkan isi dari field yang berada di luar jangkauan (*out of range*).

```
$ awk '{ $6 = ($5 + $4 + $3 + $2)
>        print $6 }' inventory-shipped
= 168
= 297
= 301
```

Anda baru saja membuat $6, yang nilainya adalah jumlah dari field $2, $3, $4, dan $5. Tanda '+' adalah operator penjumlahan. Untuk file "inventory-shipped", $6 merepresentasikan jumlah/total paket yang dikirim pada bulan tertentu.

Membuat field baru artinya adlah mengubah salinan (*copy*) internal pada awk yaitu pada input record saat ini yang direpresentasikan dengan $0. Jadi, jika Anda mecoba untuk '`print $0`' setelah menambahkan field, record akan mencetak field yang baru juga. Jumlah FS (*field separator*) juga akan menyesuaikan antara field yang baru dengan dan field yang sudah ada sebelumnya. Cobalah program berikut ini:

```
$ awk '{ $6 = ($5 + $4 + $3 + $2)
>        print $0 }' inventory-shipped
= Jan 13 25 15 115 168
= Feb 15 32 24 226 297
= Mar 15 24 34 228 301
```

Perhitungan ulang ini mempengaruhi dan dipengaruhi oleh NF (*number of field*). Misalnya nilai NF adalah jumlah field pada data yang Anda masukkan. Format yang sebenarnya dari $0 juga dipengaruhi oleh beberapa hal, misalnya oleh OFS (*output field separator*) yang digunakan untuk memisahkan fields yang nantinya akan dibahas pada bab selanjutnya. Catatan, bagaimanapun juga field yang "out-of-range" tidak mengubah nilai dari $0 atau NF. Mengacu pada field yang "out-of-range" hanya akan menghasilkan string kosong. Contohnya dapat dilihat pada program berikut ini:

```
$ awk '{ if($(NF+1) != "") print "can not happen"; else print
"everything is normal" }' BBS-list
= everything is normal
= everything is normal
= everything is normal
= ...
```

Hasil program di atas selalu menghasilkan output “everything is normal” karena $(NF+1) dipastikan melebihi range dari jumlah field. Maka dari itu, field ke-(NF+1) tidak ada/null.

Penting untuk dicatat bahwa ketika memberikan nilai pada field akan merubah nilai $0 tetapi tidak mengubah nilai dari NF (*number of fields*), bahkan ketika Anda mengisikan string kosong (empty string) pada field. Berikut ini adalah contoh program yang menggunakan OFS (*output field separator*) yang digunakan untuk memisahkan fields.

```
$ echo a b c d | awk '{ OFS = ":"; $2 = ""
> print $0; print NF }'
= a::c:d
= 4
```

Input dari program di atas adalah dari echo, yaitu: “a b c d” sedangkan OFS (output field seperator) yang sebelumnya SPACE diganti dengan “:” sehingga ketika di cetak, field dipisahkan dengan “:”. Kemudian $2 = “” maksudnya memberikan nilai field-2 berupa string kosong. Perintah “print $0” digunakan untuk mengetahui hasil perubahan, dan “print NF” untuk mengetahui jumlah field. Hasil output dari program diatas adalah “a::c:d” setelah a, terdapat dua titik dua, yang menujukkan bahwa ada field diantara dua titik tersebut. Dan itu dibuktikan dengan nilai NF = 4, yang berarti bahwa terdapat 4 field. Hal ini menunjukkan bahwa suatu field dapat berisi nilai kosong (*empty value*).

Bagaimana jika Anda ingin menambah dengan field yang baru. Program awk berikut ini akan membuat field baru :

```
$ echo a b c d | awk '{ OFS = ":"; $2 = ""; $6 = "new"
> print $0; print NF }'
= a::c:d::new
= 6
```

Field yang baru dimasukkan tersebut adalah field ke-6 ($6) dari input awal yang berjumlah 4 field. Pada hasil output program tersebut terdapat dua buah karakter titik dua setelah d, yang menunjukkan bahwa adanya field ke-5 ($5) di sana, hanya saja nilainya adalah kosong (empty value). Program awk secara otomatis membuat field ke-5 dan memberinya nilai kosong, sehingga jumlah field menjadi 6 yang ditunjukkan dengan nilai NF (*number of field*).

Bagaimana jika Anda ingin merubah nilai NF (*number of field*). Program dibawah ini akan menurunkan nilai NF kemudian menampilkan isi dari $0.

```
$ echo a b c d e f | awk '{ print "NF =", NF;
> NF = 3; print $0 }'
= NF = 6
= a b c
```

Jumlah field (NF) input program di atas adalah 6, yaitu: “a b c d e f”. Pada program tersebut nilai NF diubah menjadi 3. Hasil output hanya menampilkan field ke-1 sampai field ke-3. Hal ini menunjukkan bahwa perubahan nilai NF dapat mengubah pula jumlah field. Jika nilai NF diubah menjadi lebih kecil dari nilai NF awal, maka akan menghilangkan field setelahnya.

Penting untuk diketahui bahwa $0 adalah “*full record*” (semua isi dari record) yang dibaca dari input. Termasuk juga karakter whitespace yang memisahkan fields. Jadi bukan masalah yang aneh jika Anda mencoba untuk merubah field separator pada sebuah record sederhana dengan mengubah FS dan OFS nya, dan coba jalankan perintah “print $0” untuk menampilkan record yang telah dimodifikasi. Tetapi hal ini tidak akan bekerja jika tidak ada perubahan pada record itu sendiri. Malahan, Anda harus memaksa record untuk *rebuilt.*

Percobaan 5: Bagaimana fields dapat dipisahkan

Field separator (FS) atau pemisah kolom (biasanya berupa karakter tunggal atau sebuah regex) digunakan untuk memecah record menjadi beberapa field yang terpisah. Program awk dapat melakukannya dengan cara memindai (*scanning*) rangkaian karakter yang cocok dengan separator, kemudian field itu sendiri merupakan teks yang berada diantara separator tersebut.

Contoh berikut ini menggunakan simbol bintang (*) untuk mewakili SPACE pada output. Jika pemisah kolom adalah “oo”, maka baris berikut:

```
moo goo gai pan
```

dibagi menjadi tiga field: ‘m’, ‘*g’ dan ‘*gai*pan’. Perhatikan spasi yang ada pada field ke-2 dan ke-3.

Pemisah kolom diwakili oleh *built-in variabel* FS. Nilai dari FS dapat berubah dengan program awk dengan operator assignment (=). Waktu yang tepat untuk melakukan ini adalah pada awal eksekusi sebelum input diproses, sehingga record pertama dibaca dengan pemisah kolom yang tepat. Untuk melakukan ini lakukan dengan perintah BEGIN, contohnya kita akan menjadikan nilai dari FS adalah karakter koma (,).

```
$ awk ‘BEGIN { FS="," }; { print $2 }’
> John Q. Smith, 29 Oak St., Walamazoo, MI 42139
> John Q. Smith, LXIX, 29 Oak St., Walamazoo, MI 42139
```

Pada program di atas didefinisikan untuk nilai field separarator FS=”,” sehingga record akan dipecah kedalam field berdasarkan string “,”. Dari input yaitu:

```
John Q. Smith, 29 Oak St., Walamazoo, MI 42139
```

Maka akan dipecah ke dalam field menjadi:

```
John Q. Smith     29 Oak St.  Walamazoo MI      42139
```

Jika ditambahkan string LXIX pada field ke-2 maka string tersebut yang akan tampil. Kesimpulannya untuk memilih karakter sebagai FS dan meletakkan data dengan hati-hati adalah penting untuk mencegah terjadinya kekeliruan pembacaan. Jika data tidak dalam bentuk yang mudah untuk proses, mungkin Anda bisa memodifikasinya terlebih dahulu program awk yang Anda buat.

Ada beberapa aturan main di dalam penggunaan Field Separator (FS):

a. Whitespace secara otomatis sebagai FS

Field biasanya dipisahkan oleh *whitespace* (spasi, TAB, dan baris baru) tetapi bukan dengan *single space* (spasi tunggal). Dua spasi berturut-turut pada sebuah baris tidak membatasi field yang kosong. Nilai default dari FS (pemisah field) adalah string yang berisi spasi tunggal " ". Jika awk melihat hal yang demikian dengan cara yang biasa, setiap karakter spasi akan memisahkan field, sehingga dua spasi berturut-turut akan diartikan bahwa ada sebuah field kosong di antara keduanya. Hal ini tidak akan terjadi di awk, karena satu, dua atau lebih spasi dianggap sebagai satu nilai FS untuk menentukan batas field secara default.

Jika FS adalah karakter tunggal lainnya seperti koma "," maka setiap kemunculan karakter tersebut akan memisahkan dua field. Dua karakter tersebut jika berturut-turut, misalnya koma-koma “,,” akan membatasi field kosong. Jika karakter terjadi pada awal atau akhir baris, itu juga akan menjadi tanda yang membatasi field kosong. Karakter spasi adalah satu-satunya karakter yang tidak mengikuti aturan ini.

```
$ echo ' 1 2 3 ' | awk ‘{print $1‟ ”$2‟ ”$3}’
$ echo ‘ 1 2 3 ’ | awk ‘BEGIN {FS=" "}
>                           {print $1” ”$2” ”$3}’
```

b. Menggunakan regex sebagai FS

Kalau sebelumnya kita membahas penggunaan karakter tunggal atau string sederhana sebagai nilai FS. Sekarang kita membahas nilai FS sebagai string yang berisi regex. Dalam hal ini, regex akan mencari ekspresi yang cocok pada setiap recordnya untuk memisahkannya menjadi fields. Contohnya :

```
FS = ", \t"
```

membuat setiap baris input yang terdiri dari koma (,) diikuti dengan spasi ( ) dan TAB ( ) akan menjadi pemisah field.

Sebagai contoh penggunaan regex adalah *single space* (spasi tunggal) yang dipakai untuk memisahkan field seperti menggunakan koma. FS dapat diatur dengan menggunakan "[ ]" (kurung siku kiri, spasi, kurung siku kanan). Regex ini akan cocok dengan *single space*, tidak ada yang lainnya.

Perbedaan penting antara dua kondisi yaitu FS=" "' (spasi tunggal) dan FS = "[\ t \ n]+" (regex yang cocok untuk satu atau lebih spasi, TAB, atau baris). Untuk kedua nilai FS tersebut, field yang dipisahkan oleh karakter spasi, TAB, dan/atau *newline* (baris baru). Namun, ketika nilai FS adalah " " (spasi tunggal), awk akan mencari spasi dari record, kemudian memutuskan mana field-nya. Misalnya, program berikut ini akan mencetak 'b' Karena field separator default adalah *whitespace*. diatas (Catatan: perhatikan karakter spasi sebelum karakter 'a' dan setelah karakter 'd')

```
$ echo ' a b c d ' | awk '{ print; $2 = $2; print }'
= b
```

Tetapi program berikut ini akan mencetak 'a'.

```
$ echo ' a b c d ' | awk 'BEGIN { FS = "[ \t\n]+" }
>                         { print $2 }'
= a
```

Pada contoh diatas nilai FS diganti dengan "[\t\n]+" yang merupakan regex yang cocok dengan setiap karakter spasi, TAB atau newline. Pada input tersebut, sebelum karakter a terdapat spasi, sehingga dengan itu awk menentukan adanya field pertama dan bernilai null (field kosong). Oleh karena itu filed ke-2 adalah 'a'.

Perubahan whitespace juga terlihat pada saat rekomputasi (pembentukan ulang) $0, seperti pada contoh berikut ini:

```
$ echo '    a b c d' | awk '{ print; $2 = $2; print }'
=    a b c d
= a b c d
```

Perintah print yang pertama mencetak record sebagaimana yang terbaca dari input, dengan menyertakan whitespace secara utuh. Assignment $2 ke $2 mengakibatkan pembentukan ulang $0 dengan menggabungkan $1 beserta $NF bersamaan, dipisahkan dengan nilai OFD. Karena white space yang mendahului diabaikan ketika menemukan $1, sehingga spasi bukan merupakan bagian dari $0 yang baru. Oleh sebab itu, perintah print yang terakhir mencetak nilai $0 yang baru tanpa ada karakter whitespace diawal.

Tambahan yang harus diperhatikan untuk penggunaan regex sebagai FS adalah penggunaan operator '^' sebagai standar POSIX untuk mengawali record atau

mengawali field, hal ini berbeda-beda pada setiap versi awk. Cobalah di awk, gawk atau nawk untuk perintah berikut ini.

```
$ echo 'xxAA xxBxx C' |
> gawk -F '(^x+)|( +)' '{ for(i=1; i<=NF; i++) printf
  "-->%s<--\n", $i}'
```

Argumen -F adalah untuk mengubah field separator. Pada program di atas FS diberikan pada argumen berikutnya berupa regex "(^x+)|(+)" yang akan cocok dengan setiap yang diawali dengan karakter x, atau cocok dengan satu atau lebih karakter spasi. Operator '^' pada regex tersebut berlaku untuk setiap awal field, berbeda dengan gawk versi 3.1.6 yang memberlakukan operator tersebut hanya pada awal record.

c. Menggunakan setiap karakter sebagai FS

Ada kalanya anda ingin memeriksa masing-masing karakter dari sebuah record secara terpisah, hal ini dapat dilakukan pada awk dengan menggunakan *null string* "" (string kosong) sebagai FS seperti pada program berikut ini:

```
$ echo a b | gawk 'BEGIN { FS="" }
>     {
>           for ( i=1; i<=NF; i=i+1 )
>           print "Field", i, "is", $i
>     }'
```

Pada program di atas nilai FS diberikan nilai berupa string kosong "". Dengan demikian gawk akan memisahkan setiap karakter menjadi sebuah field. Input program di atas terdiri dari karakter 'a', karakter spasi, dan karakter 'b'. Masing-masing karakter tersebut merupakan field terpisah.

Pada dasarnya, arti dari FS adalah sama dengan spasi " " adalah tidak didefinisikan sebelumnya. Pada kasus ini, banyak versi dari Unix awk secara sederhana memperlakukan semua record hanya satu field. Beberapa versi yang kompatibel mendefiniskan FS adalah null string, termasuk gawk.

d. Setting FS dari command line

FS dapat disetting dari command line menggunakan parameter –F, dengan perintah sebagai berikut:

```
awk -F, 'program' input-files
```

Contoh di atas memberi nilai FS dengan karakter ','. Sebagai catatan bahwa untuk opsi –F (huruf besar) berbeda dengan –f (huruf kecil). Untuk –f menunjukkan file yang didalamnya terdapat program awk. Sedangkan untuk –F menunjukkan untuk pemberian nilai FS dengan karakter tertentu. Anda dapat

menggunakan –f dan –F secara bersamaan untuk men-*setting* nilai FS dan mendapatkan program awk dari sebuah file.

Karakter spesial yang lainnya pada FS harus dilakukan proses *escape* (seperti pada bab terdahulu tentang *escape sequence*). Contohnya penggunaan karakter back-slash '\' sebagai FS maka anda harus mengetikan

```
awk –F\\\\ '…' files …
# same as FS = "\\"
```

Karena karakter '\' digunakan untuk quoting pada shell-programming. Program awk melihat '-F\\' lalu memproses '\\' sebagai karakter escape dan pada akhirnya karakter '\' di akhir digunakan untuk FS.

Contoh lainnya pada mode kompatibel, jika argumen –F adalah karakter 't', maka FS akan diberikan nilai karakter TAB. Jika Anda memasukkan '-F\t' pada shell tanpa ada quote, maka karakter '\' akan dihapus. Jadi awk mengetahui bahwa Anda ingin memisahkan field dengan karakter TAB bukan karakter 't'. Gunakanlah ' -v FS="t" ' atau ' -F"t" ' pada command line jika Anda ingin memisahkan field dengan karakter 't'.

Sebagai contoh buatlah sebuah file program awk bernama *"baud.awk"* yang mengandung *pattern* /300/ dan *action* 'print $1':

```
/300/ { print $1 }
```

Kemudian berikan nilai FS dengan karakter '-' kemudian jalankan program pada file BBS-list dengan program berikut ini:

```
$ awk –F- –f baud.awk BBS-list
= aardvark 555
= alpo
= barfly 555
```

Program diatas akan mencetak daftar dari nama bulettin board yang beroperasi dengan baud 300 dan 3 digit pertama dari nomor telepon. Argumen –F pada program di atas digunakan untuk mengganti nilai FS (field separator) melalui command line dengan karakter '-'. Command di atas memakai argumen –f, yang berarti mengambil source program dari file "baud.awk" dengan file input file BBS-list. Coba lihat baris kedua dari hasil output program (alpo), karakter '-' membaca alpo-net, sehingga tidak mencetak 3 baris pertama nomor telepon. Contoh seperti ini adalah pelajaran pentingnya memperhatikan pemilihan field dan record separator (FS dan RS).

Pada Unix password system file menggunakan single character sebagai field separatornya, field pertama adalah usernamenya dan field kedua adalah passwordnya (biasanya sudah dienkripsi/shadow). Contohnya seperti beriktu ini:

```
anom:xyzzy:2076:10:AnomBesari:/home/arnold:/bin/bash
```

Program berikut ini mencari password system file dan mencetak user yang tidak memiliki password.

```
$ awk -F: '$2 == ""' /etc/passwd
```

e. Rangkuman dari field separator (FS)

Tabel berikut ini merangkum bagaimana cara memisahkan field berdasarkan pada nilai FS (tanda '==' artinya adalah "sama dengan"):

Nilai FS	Penjelasan
`FS == " "`	Field dipisahkan berdasarkan karakter whitespace, posisi whitespace berada di depan dan jumlahnya yang berlipat (*leading and trailing whitespace*) secara default diabaikan.
`FS == single character`	Field dipisahkan oleh setiap kemunculan karakter. Beberapa karakter yang muncul berturut-turut tetap dipakai untuk membatasi field kosong, seperti pada saat *leading and trailing*. Karakter tersebut bahkan bisa menjadi regex metakarakter dan tidak perlu dilakukan escape.
`FS == regex`	Field dipisahkan oleh kemunculan karakter yang cocok dengan regex. *Leading and trailing* tetap dicocokkan dengan regex untuk membatasi field yang kosong.
`FS == ""`	Setiap masing-masing karakter dalam record menjadi field yang terpisah. (Ini ada di awk, bukan standar POSIX).

Percobaan 6: Membaca data yang *fixed-width*

Gawk menyediakan fasilitas untuk memanipulasi fixed-width field tanpa field separator yang jelas. Sebagai contoh, sebuah tabel kadang kadang field kosongnya dibiarkan dengan terisi karakter kosong (whitespace). Dan karenanya ketika digunakan FS sebagai pemisah field tidak dapat bekerja seperti yang diinginkan.

Pemisahan input record menjadi fixed-width field ditentukan dengan cara memberikan nilai string yang berisi whitespace menjadi built-in-variable FIELDWIDTHS. Setiap angka menentukan lebar field, termasuk kolom antar field. Jika Anda ingin mengabaikan kolom antar field, Anda dapat menentukan lebar sebagai field terpisah yang kemudian diabaikan. Ini adalah sebuah kesalahan fatal dengan memberikan nilai lebar field dengan bukan bilangan positif. Berikut ini adalah contoh sebuah output dari utilitas unix yang dipakai untuk menggambarkan penggunaan FIELDWIDTHS:

```
10:06pm up 21 days, 14:04, 23 users
User        tty   login       idle        JCPU  PCPU  what
hzuo        ttyV0 8:58pm                  9     5     vi p24.tex
hzang       ttyV3 6:37pm      50                      -csh
eklye       ttyV5 9:53pm                  7     1     em thes.tex
dportein    ttyV6 8:17pm      1:47                    -csh
gierd       ttyD3 10:00pm     1                       elm
dave        ttyD4 9:47pm                  4     4     w
brent       ttyp0 26Jun91     4:46        26:46 4:41  bash
dave        ttyq4 26Jun91     15days      46    46    wnewmail
```

Kita akan membuat sebuah program yang mengambil data di atas sebagai input, kemudian mengubah waktu idle untuk ke dalam satuan detik, dan mencetak kolom pertama, kedua dan waktu idle dalam satuan detik. Simpan data diatas dalam sebuah file misalnya namanya “data.txt”. Kemudian buatlah sebuah program file dengan nama “baud.awk”. Source code program sebagai berikut:

```
BEGIN { FIELDWIDTHS = "9 6 10 6 7 7 35" }
NR > 2 {
      idle = $4
      sub(/^ */, "", idle) # strip leading spaces
      if (idle == "")
            idle = 0
      if (idle ~ /:/) {
            split(idle, t, ":")
            idle = t[1] * 60 + t[2]
      }
      if (idle ~ /days/)
            idle *= 24 * 60 * 60

      print $1, $2, idle
}
```

Kemudian jalankan dengan perintah seperti dibawah ini:

```
$ awk -f baud.awk data.txt
```

Jika benar, program diatas akan menghasilkan tampilan berikut ini:

```
hzuo        ttyV0       0
hzang       ttyV3       50
eklye       ttyV5       0
dportein    ttyV6       107
gierd       ttyD3       1
dave        ttyD4       0
brent       ttyp0       286
dave        ttyq4       1296000
```

Program diatas mendifinisikan FILEWIDTHS dengan “9 6 10 6 7 7 35”. Kemudian jika NR (number of record) lebih dari 2, isi field ke-4 disimpan pada variabel “idle”. Kemungkinan adanya spasi pada nilai variable tersebut dihilangkan dengan fungsi sub, memanfaatkan regex ‘/^ */’ yang cocok dengan setiap spasi di awal string, diganti dengan string kososng “”. Kemudian memindai idle yang

berisi karakter ‘:’. Pemindaian tersebut mencari field yang berisi informasi waktu (‘:’ adalah separator waktu). Jika diperoleh string yang berisi informasi waktu, kemudian dipecah dengan fungsi split agar diperoleh jam, menit dan detik. Jika idle mengandung ‘days’ berarti informasi tersebut adalah waktu dalam hari, sehingga dikalikan dengan 24*60*60 agar diperoleh waktu dalam detik. Perintah print $1, $2, idle mencetak field-1 (user name), field-2 (TTY), dan nilai variable idle (menyimpan data waktu dalam detik).

Percobaan 7: Mendefinisikan field dengan content (isi)

Biasanya ketika menggunakan FS, gawk mendefinisikan field sebagai bagian dari record yang terjadi di antara setiap field separator. Dengan kata lain, FS mendefinisikan apakah ada sebuah field atau tidak. Namun, ada saat dimana Anda ingin mendefinisikan field sesuai dengan isi field itu sendiri bukan berdasarkan FS nya.

Kasus tersebut yang paling terkenal adalah seperti apa yang disebut dengan *comma separated value* (CSV). Banyak program *spreadsheet*, misalnya, dapat mengekspor data ke file teks, di mana setiap record diakhiri dengan *newline*, dan field yang dipisahkan oleh koma. Jika data hanya dipisahkan dengan koma, hal itu tidak akan menjadi masalah. Masalah akan muncul ketika salah satu field yang berisi *embedded comma*. Meskipun tidak ada standar formal untuk data CSV, dalam kasus ini, sebagian besar program *embed* field dalam tanda kutip ganda. Contohnya data file “addresses.csv” seperti ini:

```
Robbins,Arnold, "1234 A Pretty Street,NE", MyTown, MyState, 12345-
6789, USA
```

Variabel FPAT berikut akan menawarkan solusi untuk kasus seperti ini. Nilai dari FPAT adalah sebuah string yang berisi regex yang menggambarkan konten atau isi dari masing-masing field. Pada kasus data CSV seperti diatas definisi field adalah sebagai berikut:

```
“anything that is not a comma,” or “a double quote, anything that
is not a double quote, and a closing double quote.”
```

Jika ditulis dengan dengan regex dalam file “simple-csv.awk” adalah sebagai berikut:

```
BEGIN {
      FPAT = "([^,]+)|(\"[^\"]+\")"
}

{
      print "NF = ", NF
      for (i = 1; i <= NF; i++) {
            printf("$%d = <%s>\n", i, $i)
```

```
        }
}
```

Kemudian jalankan dengan perintah di bawah ini:

```
$ gawk -f simple-csv.awk addresses.csv
```

Akan ditampilkan hasil sebagai berikut:

```
NF = 7
$1 = <Robbins>
$2 = <Arnold>
$3 = <"1234 A Pretty Street, NE">
$4 = <MyTown>
$5 = <MyState>
$6 = <12345-6789>
$7 = <USA>
```

Catatan: bahwa *embedded comma* (koma yang berada dalam tanda petik) ada pada $3. Jika ingin menambah program diatas agar dapat menghilangkan *quotes* atau tanda petik (”) coba tambahkan baris program berikut ini:

```
if (substr($i, 1, 1) == "\"") {
      len = length($i)
      $i = substr($i, 2, len - 2)
      # Get text within the two quotes
}
```

Seperti dituliskan sebelumnya, regex untuk FPAT membutuhkan setiap field harus memiliki minimal satu karakter. Modifikasi regex diatas dengan mengganti ‘+’ pertama dengan ‘*’ akan mengijinkan field boleh berisi kosong. Cobalah ganti regex diatas dengan:

```
FPAT = "([^,]*)|(\"[^\"]+\")"
```

Dibab selanjutnya akan dibahas fungsi *patsplit()* yang digunakan untuk memisahkan regular string.

Percobaan 8: Multiple line records

Pada beberapa database, sebuah *single line* (satu baris) tidak dapat dengan mudah menampung semua informasi dalam satu masukan. Dalam kasus tersebut, Anda dapat menggunakan *multi-line record*. Langkah pertama adalah memilih format data Anda. Salah satu teknik adalah menggunakan karakter atau string untuk memisahkan records. Contohnya penggunaan “\f” pada awk atau pemrograman C sebagai karakter *formfeed* untuk memisahkan tiap record pada suatu file. Untuk melakukan ini tinggal kita set variabel RS dengan “\f”.

Cara yang lain dengan menggunakan *blank lines* (baris kosong) untuk memisahkan record. Nilai dari RS didefinisikan sebagai *"empty string"* yang mengindikasikan bahwa record dipisahkan dengan satu atau lebih *blank lines*. Ketika RS diisi dengan empty string, setiap record selalu berakhir pada baris kosong yang pertama kali dijumpai. Record berikutnya tidak akan dimulai sampai baris pertama diikuti oleh baris yang *"nonblank"* (tidak kosong). Tidak peduli berapa banyak baris kosong muncul berturut-turut, mereka semua bertindak sebagai satu record separator (RS). Baris kosong harus benar-benar kosong, baris yang hanya berisi spasi tidak dihitung.

Kita ketahui bahwa input dipisahkan menjadi record, langkah kedua adalah untuk memisahkan record menjadi field. Salah satu cara untuk melakukan ini adalah dengan membagi setiap baris record menjadi field dengan cara yang biasa. Hal ini terjadi secara default sebagai hasil dari fitur khusus. Ketika RS diinisialisasi sebagai *empty string*, dan FS diinisialisasi sebagai satu karakter, karakter *new line* baris baru selalu bertindak sebagai field separator (FS).

Adanya pengecualian khusus yaitu FS sama dengan karakter *empty string* (""). Fitur ini bisa menjadi masalah jika Anda benar-benar tidak ingin karakter *new line* digunakan untuk memisahkan field, karena tidak ada cara untuk mencegahnya. Namun, Anda bisa menyiasatinya dengan menggunakan fungsi *"split()"* untuk memecah rekord secara manual (akan dijelaskan di bab selanjutnya). Jika Anda memiliki karakter yang digunakan sebagai field separator, Anda dapat bekerja di sekitar fitur khusus dengan cara yang berbeda. Caranya adalah dengan membuat FS menjadi regex karakter tunggal. Sebagai contoh, jika field separator adalah karakter persen (%), maka penulisannya bukan FS = "%", tetapi gunakan FS = "[%]".

Cara yang lain untuk memisahkan field adalah dengan meletakkan setiap field pada baris yang terpisah. Untuk melakukannya cukup dengan memberikan nilai pada variabel FS dengan "\n" yang mengindikasikan karakter *newline* (baris baru). Contoh berikut ini menggambarkan data file yang terorganisir pada sebuah mailing list dimana setiap masukan dipisahkan dengan *blank lines*.

Source file "data.txt"

```
Jane Doe
123 Main Street
Anywhere, SE 12345-6789

John Smith
456 Tree-lined Avenue
Smallville, MW 98765-4321
```

Source program "addrs.awk"

```
# addrs.awk --- simple mailing list program
```

```
# Records are separated by blank lines.
# Each line is one field.

BEGIN{RS="";FS="\n"}
{
      Print "Name is:",$1
      Print "Address is:",$2
      Print "City and Stateare:",$3
      Print ""
}
```

Cobalah jalankan perintah dibawah ini:

```
$ awk –f addrs.awk data.txt
```

Dengan mengganti nilai RS (record separator) dengan “”, maka record akan dipisahkan dengan blank line (baris kosong). Dengan mengganti FS (field separator) dengan “\n”, maka field akan dipisahkan dengan karanter new line.

Tabel berikut ini menunjukkan bagaimana record dipisahkan berdasarkan nilai RS.

Nilai RS	Penjelasan
`RS == "\n"`	Record dipisahkan dengan karakter newline (‘\n’). Setiap baris pada data file akan dipisahkan, termasuk baris kosong. Default-nya seperti ini.
`RS == any single character`	Record dipisahkan dengan kemunculan karakter tertentu. Kemunculan karakter bersama-sama akan membuat record kosong.
`RS == ""`	Record dipisahkan dengan adanya baris kosong. Jika FS adalah sebuah *single character,* lalu karakter *newline* akan menjadi field separator, dengan kata lain nilai FS bisa bernilai apapun. *Leading* dan *trailing* untuk newline dalam sebuah file diabaikan.
`RS == regexp`	Record dipisahkan dengan kemunculan karakter yang cocok dengan regex. *Leading* dan *trailing* yang cocok dengan regex akan membatasi record kosong. (Khusus gawk, bukan standard POSIX)

Pada banyak kasus gawk memberikan nilai RT sebagai input teks yang dicocokkan dengan nilai RS yang spesifik. Tetapi jika input file berakhir tanpa teks yang cocok dengan RS, maka gawk memberikan nilai RT menjadi *null string*.

Percobaan 9: Explicit input menggunakan getline

Sejauh ini Anda sudah mencoba mendapatkan input data dari *awk main input stream* yang merupakan standar input (biasanya menggunakan terminal, kadang-

kadang outputnya dari program lainnya) atau dari file yang ditentukan pada command line. Bahasa awk memiliki *built-in command* khusus disebut dengan "getline" digunakan untuk membaca input yang dapat Anda kontrol secara eksplisit.

Perintah "getline" digunakan dalam beberapa cara yang berbeda dan bukan untuk pemula. Contoh-contoh perintah "getline" di bawah ini ada beberapa yang belum diajarkan, oleh karena itu Anda direkomendasikan untuk kembali belajar perintah "getline" setelah Anda membaca buku ini dan memiliki pengetahuan yang baik tentang bagaimana awk bekerja.

Perintah "getline" mengembalikan nilai "1" jika menemukan sebuah rekord dan menggembalikan nilai "0" jika menemukan akhir file atau *end of file* (EOF). Jika ada beberapa *error* / kesalahan dalam mendapatkan rekord (seperti file yang tidak dapat dibuka) maka perintah getline akan mengembalikan nilai "-1". Pada kasus ini, gawk menetapkan variabel ERRNO sebagai sebuah string yang menjelaskan kesalahan yang terjadi. Pada contoh berikut, yang dimaksud *command* / perintah adalah nilai string yang mewakili perintah pada shell.

a. Menggunakan getline tanpa arguments

Untuk membaca input file, perintah getline dapat digunakan tanpa argumen. Proses diselesaikan dengan membaca input record berikutnya kemudian memecahnya menjadi field sehingga menjadi lebih efektif. Hal ini berguna jika Anda sudah selesai memproses record saat ini, tetapi Anda ingin melakukan proses yang lainnya pada record setelahnya saat itu juga. Contohnya pada program berikut ini:

Source program getline_noArgs.awk:

```
{
      if ((t = index($0, "/*")) != 0) {

            # value of 'tmp' will be "" if t is 1
            tmp = substr($0, 1, t - 1)
            u = index(substr($0, t + 2), "*/")
            offset = t + 2

            while (u == 0) {

                  if (getline <= 0) {
                        m = "unexpected EOF or error"
                        m = (m ": " ERRNO)
                        print m > "/dev/stderr"
                        exit
                  }
                  u = index($0, "*/")
                  offset = 0
            }
            # substr() expression will be "" if */
```

```
            # occurred at end of line
            $0 = tmp substr($0, offset + u + 2)
      }
      print $0
}
```

Cobalah jalankan perintah dibawah ini dan masukkan beberapa string berikut:

```
$ awk –f getline_noArgs.awk
> komentar ini /*akan*/ dihapus
> komentar ini /*akan*/ dihapus /*klo yang ini bagaimana?*/
```

Program awk di atas menghapus comment seperti pada bahasa pemrograman C, yaitu string input yang diapit oleh oleh '/* … */'. Kemudian digantikan dengan 'print $0' atau baris perintah lainnya. Anda dapat melakukan proses yang lebih kompleks pada input yang berupa komentar, seperti mencari yang cocok dengan regex. Program ini memiliki masalah yaitu tidak bekerja jika satu komentar berakhir dan yang lain dimulai pada baris yang sama. Bentuk ini memberikan nilai pada NF, NR, FNR, dan nilai dari $0.

b. Menggunakan getline ke dalam variabel

Anda bisa menggunakan perintah "getline *var*" untuk membaca record selanjutnya dari input awk dan disimpan di variabel *var* dan tidak ada proses lainnya. Contohnya, misalkan baris selanjutnya adalah *comment* atau *special string*, dan Anda ingin membacanya tanpa ada aturan-aturan yang menjadi *trigger* (pemicu). Bentuk getline yang seperti ini mengijinkan Anda untuk membaca baris tersebut dan menyimpannya pada sebuah variabel. Sehingga looping dari "*read-a-line-and-check-each-rule"* yang utama dari awk tidak pernah melihatnya. Contoh berikut ini menukar setiap dua baris input.

Source program getline_var.awk:

```
{
      if ((getline tmp) > 0) {
            print tmp
            print $0
      } else
            print $0
}
```

Cobalah jalankan perintah dibawah ini dan masukkan beberapa string berikut:

```
$ awk –f getline_var.awk
> Satu
> Dua
> Tiga
> Empat
```

Pada program di atas, getline menyimpan record yang dibaca ke dalam variabel tmp. Karena getline membaca record berikutnya dari record input, maka getline

memberikan nilai return tidak sama dengan '0' jika setidaknya ada dua input. Input yang dibaca oleh getline disimpan pada variabel tmp sedangkan nilai dari record sebelumnya tetap. Perintah getline yang digunakan dalam format ini hanya berpengaruh pada nilai variabel NR dan FNR.

c. Menggunakan getline dari file

Perintah "getline *< file*" dapat digunakan untuk membaca record selanjutnya dari sebuah file. Istilah "file" dalam perintah di atas adalah sebuah string yang menunjukkan nama dari sebuah file. Ekspresi '*< file*' disebut dengan proses "redirection" karena itu mengarahkan input berasal dari tempat yang berbeda. Contohnya program berikut ini membaca input record dari file "*secondary.input"* ketika bertemu dengan field pertama dengan nilai sama dengan 10 dalam file input saat ini:

Source program getline_file.awk:

```
{
      if ($1 == 10) {
            getline < "secondary.input"
            print
      } else
            print
}
```

Cobalah jalankan perintah dibawah ini dan masukkan beberapa string berikut:

```
$ awk –f getline_file.awk
> 15 fakir miskin
> 12 anak yatim
> 10 janda tua
```

Program diatas menggunakan perintah getline untuk membaca setiap record yang bersumber dari file "secondary.input". Jika field pertama dari record yang dibaca bernilai sama dengan 10, maka record tersebtu akan dicetak kembali. Format "getline < file" dapat digunakan untuk membaca record berikutnya dari suatu file input. "< file" merupakan redirection karena mengalihkan input agar berpindah ke tempat lain. Karena input stream utama tidak digunakan, nilai variabel NR dan FNR tidak berubah. Namun, karena record yang dibaca dipecah menjadi field seperti biasanya, maka nilai $0 dan field lainnya berubah saat nilai NF berubah.

d. Menggunakan getline ke dalam variabel dari file

Perintah "*getline var < file*" digunakan untuk membaca input dari *file,* dan meletakkannya pada variabel "*var"*. Istilah "file" dalam perintah di atas adalah sebuah string yang menunjukkan nama dari sebuah file yang akan dibaca. Pada getline jenis ini, tidak ada built-in-variable yang berubah dan record tidak dipisah

menjadi fields. Satu-satunya variabel yang berubah adalah "*var*". Contohnya, program berikut ini akan menyalin semua dari file input ke file output, kecuali untuk record yang berisi *"@include filename"*. Seperti sebuah record diganti dengan isi dari file yang bernama "*filename*":

Source program getline_varfile.awk:

```
{
      if (NF == 2 && $1 == "@include") {
            while ((getline line < $2) > 0)
                  print line
            close($2)
      } else
            print
}
```

Cobalah jalankan perintah dibawah ini dan masukkan beberapa string berikut:

```
$ awk –f getline_varfile.awk
> @include no file
> @include record.input
```

Program diatas membaca input, jika inputnya terdiri dari dua field (NF==2) dan field pertama bernilai "@include" ($1=="@include"), maka akan melanjutkan membaca input field yang nama file nya diperoleh dari field ke-dua dari input. Pada masukan pertama, "@include no file", tidak dilanjutkan membaca isi file input karena jumlah field masukan tersebut lebih dari 2, sehingga input tersebut dicetak kembali dengan fungsi "*print*". Kemudian pada masukan ke-dua, *"@include record.input*", dilanjutkan membaca file input, yaitu "*records.input*".

Selama record dalam file records.input masih ada, maka akan dilakukan pembacaan. Proses tersebut mengkombinasikan format "*getline var*" dan *"< file*". Format "*getline var*" digunakan untuk membaca record berikutnya dari suatu input program awk kemudian menyimpannya ke dalam variable var. Format "*getline < file*" digunakan untuk membaca record berikutnya dari suatu file input. Bentuk *"< file*" merupakan redirection karena mengalihkan input agar berpindah ke tempat lain. Maka dengan "*getline line < $2*" getline akan membaca record dari file record.input kemudian menyimpan nilai dari field ke-dua ke dalam variabel line. Fungsi print digunakan untuk mencetak hasilnya.

Peggunaan getline dalam format ini tidak mempengaruhi nilai dari built-in variable, yang berubah hanya nilai dari variabel "*var*" atau dalam program ini variabel line. Nilai variabel line akan berubah sesuai dengan nilai record yang dibaca.

e. Menggunakan getline dari pipe

Output dari sebuah perintah dapat dilakukan proses *pipe* ke perintah getline menggunakan "*command | getline*". Pada kasus ini, string *command* dijalankan sebagai perintah shell dan outputnya dapat dilakukan proses *pipe* ke awk untuk digunakan sebagai input. Bentuk getline seperti ini membaca satu record pada satu waktu dari proses *pipe.* Contoh program berikut ini menyalin input ke output, kecuali untuk baris yang dimulai dengan "*@execute*", dimana akan digantikan dengan output dari baris setelahnya sebagai perintah shell.

Source program getline_pipe.awk:

```
{
      if ($1 == "@execute") {
            tmp = substr($0, 10) # Remove "@execute"
            while ((tmp | getline) > 0)
            print
            close(tmp)
      } else
      print
}
```

Cobalah jalankan perintah dibawah ini dan masukkan beberapa string berikut:

```
$ awk –f getline_pipe.awk
@execute halo
@execute echo ini hasil tampilan perintah echo
@execute awk 'BEGIN { print "ini tampilan print dari awk"}'
```

Program di atas membaca input, jika field pertama bernilai sama dengan *"@execute*", maka akan menyimpan input sebagari string dari indeks ke-10 dengan menggunakan fungsi substr ke dalam variabel tmp. getline dimanfaat sebagai filter (*pipe*) untuk melakukan cek, apakah variabel tmp kosong atau tidak, kemudian nilai dari variabel tmp akan dicetak ke commandline untuk dieksekusi. Dalam konteks ini, string input tersebut akan dijalankan sebagai shell command. Fungsi *close()* digunakan untuk memastikan jika ada 2 "*@execute*" yang muncul pada satu baris input, maka menjalankan masing-masing perintahnya.

f. Menggunakan getline pada variabel dari pipe

Ketika Anda menggunakan "*command | getline var*", output dari perintah "command" tersebut dikirimkan lewat sebuah *pipe* ke getline dan ditujukan ke varibel *var*. Sebagai contoh, program berikut ini membaca tanggal dan waktu kemudian disimpan ke variabel *current_time*, menggunakan utilitas *date*, lalu mencetaknya:

Source program getline_varpipe.awk

```
BEGIN {
      "date" | getline current_time
```

```
        close("date")
        print "Report printed on " current_time
}
```

Cobalah jalankan perintah dibawah ini dan masukkan beberapa string berikut:

```
$ awk –f getline_varpipe.awk
```

Program di atas menfilter (pipe) output dari program "*date*", dengan membaca record dari output tersebut kemudian menyimpannya ke dalam variabel current_time. Input dari suatu pipe ke dalam getline merupakan operasi satu arah. Perintah yang diawali dengan "*command | getline*" hanya mengirim data ke dalam program awk. Tidak ada built-in-variable yang diubah dan record-nya tidak dipisahkan menjadi field. Jika diperlukan pengiriman data ke program lain dengan tujuan memproses dan membaca ulang, maka dapat dijalankan dengan membuat "*coprocess"*. *Coprocess* tersebut dapat memungkinkan program gawk untuk melakukan komunikasi dua arah.

g. Menggunakan getline dari coprocess

Masukan ke dalam getline dari sebuah *pipe* adalah operasi satu arah. Perintah yang dimulai dengan "*command | getline*" hanya mengirimkan data program awk Anda. Pada kesempatan lain, Anda mungkin ingin mengirim data ke program lain untuk diproses dan kemudian membaca kembali hasilnya. Pada gawk dimungkinkan agar Anda dapat memulai "*coprocess"* yang memungkinkan komunikasi dua arah. Hal ini dilakukan dengan operator "|&". Biasanya, Anda pertama kali menuliskan data ke *coprocess* dan kemudian membaca kembali hasilnya, seperti yang ditunjukkan pada program berikut yang akan mengirimkan query ke db-server dan kemudian membaca hasilnya:

```
print "some query" |& "db_server"
"db_server" |& getline
```

Nilai-nilai NR dan FNR tidak berubah, karena aliran masukan utama (*main input stream*) tidak digunakan. Namun, record akan dibagi menjadi field dengan cara yang biasa, sehingga mengubah nilai $0, dari field lainnya, dan NF. Coprocesses adalah fitur canggih, nanti akan dibahas dengan detail di bab selanjutnya.

h. Menggunakan getline pada variabel dari coprocess

Bila Anda menggunakan "*command |& getline var*", output dari perintah *coprocess* dikirim melalui pipe dengan dua cara, yaitu ke *getline* dan ke variabel *var*. Pada bentuk *getline* yang seperti ini, tidak ada satupun variable built-in yang diubah dan record tidak terpecah menjadi field. Satu-satunya variabel berubah adalah var.

i. Hal-hal yang harus diingat pada getline

Berikut ini adalah hal-hal yang harus diperhatikan pada penggunaan getline:

- Ketika getline mengubah nilai $0 dan NF, awk tidak secara otomatis melompat ke awal program dan mulai menguji rekord baru dengan setiap pola. Kemudian rekord baru tersebut diuji terhadap aturan berikutnya.

- Banyak implementasi awk membatasi jumlah *pipe*, mungkin program awk menyediakan hanya satu. Pada gawk, tidak ada batasan seperti itu. Anda dapat menggunakan banyak *pipe* (dan *coprocesses*) sebagai dasar dari proses pada sistem operasi.

- Efek samping akan terjadi jika Anda menggunakan getline tanpa *redirection* dalam aturan BEGIN. Karena sebuah getline yang *unredirected* akan membaca data file dari *command line*, perintah getline pertama menyebabkan awk untuk memberikan nilai pada FILENAME. Biasanya, FILENAME tidak memiliki nilai pada aturan BEGIN, karena Anda belum mulai memproses data file pada *command line*.

- Menggunakan FILENAME dengan getline (*"getline < FILENAME"*) cenderung membingungkan, dengan awk membuka input stream yang terpisah dari file input. Namun, dengan tidak menggunakan variabel, nilai $0 dan NR akan diperbarui. Jika Anda melakukan ini, mungkin karena kesalahan dan Anda harus mempertimbangkan kembali apa yang Anda lakukan.

- Berikut ini akan disajikan tabel ringkasan variasi getline dan variabel mana yang dapat dipengaruhi. Perlu dicatat bahwa variasi yang tidak menggunakan *redirection* dapat menyebabkan FILENAME akan diperbarui nilainya jika awk mulai membaca sebuah file input yang baru. Tabel berikut ini merangkum dari 8 variasi dari getline, daftar dari built-in-variable ditentukan masing-masing dimana masing-masing variasi tersebut adalah standard gawk.

Variant	Effect	Standard/Extension
`getline`	Sets $0, NF, FNR, and NR	Standard
`getline var`	Sets var, FNR, and NR	Standard
`getline < file`	Sets $0 and NF	Standard
`getline var < file`	Sets var	Standard
`command | getline`	Sets $0 and NF	Standard
`command | getline var`	Sets var	Standard
`command |& getline`	Sets $0 and NF	Extension
`command |& getline var`	Sets var	Extension

- Jika variabel yang diberi nilai adalah ekspresi yang berefek samping, versi yang lain dari awk akan berbeda hasilnya pada saat menghadapi EOF (end-of-file). Beberapa versi awk tidak mengevaluasi ekspresi, banyak versi (termasuk gawk) melakukan evaluasi. Berikut ini adalah contoh programnya:

Source program getEOF.awk

```
BEGIN {
      system("echo 1 > f")
      while ((getline a[++c] < "f") > 0) { }
      print c
}
```

Cobalah jalankan perintah dibawah ini dan masukkan beberapa string berikut:

```
$ awk –f getEOF.awk
```

Hasil di atas yang menyebabkan efek sampingnya adalah '++c'. Apakah variabel 'c' naik nilainya jika bertemu EOF, sebelum elemennya pada variabel 'a' diberi nilai? Pada gawk, getline diperlakukan seperti pemanggilan fungsi, dan mengevaluasi ekspresi 'a[++c]' sebelum mencoba untuk membaca dari f. Versi lain dari awk hanya mengevaluasi ekspresi setelah mereka tahu bahwa ada nilai string yang akan diisi nilainya.

- Berdasarkan standard POSIX, nama file pada *awk command line* haruslah *text files*. Akan terjadi error jika bukan *text files*. Kebanyakan versi awk, jika mengakses sebuah direktori pada *command line* akan terjadi error. Secara default, gawk memberikan *warning* (peringatan) jika dilakukan pengaksesan direktori pada command line, tetapi kebanyakan versi mengabaikannya. Jika dilakukan pilihan --*posix* atau –*traditional*, gawk akan memberlakukan akses direktori dari command line sebagai error.