Praktikum 2
Data File pada "AWK"

Tujuan Pembelajaran

Mahasiswa dapat memahami dan menggunakan data file dalam bahasa pemrograman awk.

Dasar Teori

Kebanyakan program komputer bekerja dengan file. Hal ini karena file membantu dalam menyimpan informasi secara permanen. Compiler membaca file sumber dan menghasilkan file executable. File itu sendiri adalah sekelompok byte yang tersimpan pada beberapa perangkat penyimpanan seperti pita, disk magnetik, disk optik dll file data adalah file yang menyimpan data yang berkaitan dengan aplikasi tertentu, untuk kemudian digunakan.

Data file adalah file pada komputer yang menyimpan data untuk digunakan oleh aplikasi komputer atau sistem. Pada umumnya file ini tidak merujuk ke file yang berisi instruksi atau kode yang akan dieksekusi (biasanya disebut file program), atau file yang mendefinisikan operasi atau struktur dari suatu aplikasi atau sistem (termasuk file konfigurasi, file direktori, dll), tetapi khusus untuk informasi yang digunakan sebagai input, atau ditulis sebagai output oleh beberapa program perangkat lunak lainnya. Hal ini sangat berguna ketika mencari kesalahan program.

Data file terdiri dari file teks (*text-file*) dan file biner (*binary-file*). file teks (juga disebut file ASCII) menyimpan informasi dalam karakter ASCII. File teks berisi karakter yang dapat terlihat. Kita bisa melihat isi dari file pada monitor atau mengeditnya menggunakan salah satu editor teks. Dalam file teks, setiap baris teks dihentikan, (dipisahkan) dengan karakter khusus yang dikenal sebagai EOL (End of Line) karakter. Dalam file teks terjemahan internal yang terjadi ketika karakter ini EOL dibaca atau ditulis. Contoh file teks adalah file yang berisi program C++. Sedangkan file biner adalah file yang berisi informasi yang disimpan dalam format yang sama seperti informasi tersimpan di memori yaitu dalam bentuk biner. Dalam file biner, tidak ada pemisah untuk baris. Juga tidak ada terjemahan terjadi pada file biner. Akibatnya, file biner lebih cepat dan lebih mudah dibaca dan ditulis oleh program daripada file teks. Selama file tidak perlu membaca atau porting ke berbagai jenis sistem, file biner adalah cara terbaik untuk menyimpan informasi program. Contohnya *executable file* dan *object file*.

Pada bahasa C++, file ada pada tingkat terendah, diintrepretasikan hanya sebagai urutan/*stream* (aliran) byte. Dengan kata lain, file pada tingkat pengguna bahasa C++ terdiri dari urutan kemungkinan, bercampur tipe data dan karakter, nilai aritmatika, objek kelas, dan lainnya. Berbeda dengan bahasa pemrograman awk yang berfungsi sebagai "*text-manipulation"* atau istilah yang lain menyebutnya sebagai "*word processor*", dengan demikian menjadikan operasi file lebih mudah.

Percobaan 1: Contoh stuktur data file

Berikut ini disediakan dua buah contoh data file. Data file pertama bernama "BBS-list" merepresentasikan sebuah daftar sistem "computer bulletin board" beserta informasi mengenai sistem tersebut.

```
aardvark    555-5553    1200/300          B
alpo-net    555-3412    2400/1200/300     A
barfly      555-7685    1200/300          A
bites       555-1675    2400/1200/300     A
camelot     555-0542    300               C
core        555-2912    1200/300          C
fooey       555-1234    2400/1200/300     B
foot        555-6699    1200/300          B
macfoo      555-6480    1200/300          A
sdace       555-3430    2400/1200/300     A
sabafoo     555-2127    1200/300          C
```

Data file kedua bernama "inventory-shipped" merepresentasikan sebuah informasi daftar pelayaran. Masing-masing *record* terdiri dari: bulan, jumlah peti, jumlah kotak, jumlah tas, dan jumlah paket. Ada 16 *entries* mencakup data 12 bulan dan 4 bulan pertama.

```
Jan   13    25    15    115
Feb   15    32    24    226
Mar   15    24    34    228
Apr   31    52    63    420
May   16    34    29    208
Jun   31    42    75    492
Jul   24    34    67    436
Aug   15    34    47    316
Sep   13    55    37    277
Oct   29    54    68    525
Nov   20    87    82    577
Dec   17    35    61    401

Jan   21    36    64    620
Feb   26    58    80    652
Mar   24    75    70    495
Apr   21    70    74    514
```

Perintah berikut ini menjalankan program awk sederhana yang mencari karakter string “foo” dalam input file “BBS-list”.

```
$ awk '/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
```

Dengan perintah di atas, maka saat ditemukan baris yang mengandung ‘foo’ akan dicetak karena “`print $0`” berarti mencetak baris tersebut.

Karakter slash (‘`/`’) mengapit karakter ‘foo’ adalah pola yang dicari. Tipe pola tersebut merupakan *regular expression*. Pola yang digunakan ini membolehkan kesesuaian untuk sebagian kata. Terdapat single quote (tanda petik) yang melingkupi program awk sehingga shell tidak akan mengenalinya sebagai karakter shell khusus.

Secara praktis banyak program awk yang terdiri dari hanya satu atau dua baris. Berikut ini adalah kumpulan dari program yang sederhana yang bisa langsung Anda coba. Beberapa program mengandung *syntax* (konstruksi) yang belum dibahas dalam bab ini dan akan dibahas dalam bab yang akan datang. Tetapi jangan kuatir, dicoba saja karena akan memberikan Anda pemahaman tentang bagaimana awk bekerja.

Percobaan 2: Contoh program sederhana

Sebagian besar contoh program awk menggunakan file data bernama “data”. Ini hanyalah contoh saja. Tidak mengapa jika Anda menggunakan program ini dengan mengganti nama file Anda sendiri untuk data. Contoh di bawah ini adalah file “data”.

```
This is first line
This is second line
This is third line
```

Untuk referensi di masa mendatang, perhatikan bahwa sering ada lebih dari satu cara untuk melakukan hal yang sama di awk. Suatu ketika, mungkin Anda ingin melihat kembali contoh ini dan melihat apakah Anda dapat melakukan dengan cara yang berbeda untuk melakukan hal yang sama. Berikut ini beberapa contoh program sederhana awk:

a) Berikut ini program untuk awk yang menggunakan operator logika untuk mencari baris terpanjang. Panjang baris tersebut dapat diketahui dari *built-in variable length*.

```
$ awk '{ if (length($0) > max) max = length($0) }
     END { print max }' data
```

b) Berikut ini program untuk mencetak setiap baris yang lebih dari 80 karakter. Untuk satu syarat kondisi tertentu, tidak diperlukan logika if, cukup dengan memakai operator logika dalam program. Secara default, tanpa memberikan perintah print, maka program akan mencetak satu baris secara keseluruhan (bukan field tertentu).

```
$ awk 'length($0) > 80' data
```

c) Berikut ini program untuk mencetak panjang dari baris yang paling panjang. Input file diproses melakukan perubahan terhadap karakter "TAB" menjadi "space" jadi lebarnya baris dapat dibandingkan dengan kondisi yang sebenarnya. Perintah pada blok END akan dilakukan pada akhir proses program. Bagian ini bermanfaat untuk memperoleh informasi hasil akhir dari proses/kalkulasi yang telah dilakukan dalam bagian program sebelumnya.

```
$ expand data | awk '{ if (x<length()) x=length()
}
     END { print "maximum line length is " x }'
```

d) Berikut ini program untuk mencetak baris yang memiliki minimal satu field. Ini adalah cara yang mudah untuk menghapus baris kosong (*blank lines*) dari sebuah file atau untuk membuat file baru mirip dengan file lama dimana baris kosongnya telah dihapus. NF adalah *built-in variable* yang menyimpan jumlah field. Secara default, jumlah field juga merepresentasikan jumlah kolom, karena satu field adalah satu kolom.

```
$ awk 'NF > 0' data
```

e) Berikut ini program untuk mencetak bilangan secara acak (*random*) dari 0 sampai 100. Fungsi *rand()* membangkitkan bilangan acak. Proses looping dilakukan dengan *for* dan memakai variable *I* sebagai kontrolnya.

```
$ awk 'BEGIN { for (i = 1; i <= 7; i++)
     print int(101 * rand()) }'
```

f) Berikut ini program untuk mencetak jumlah bytes yang digunakan dalam sebuah file. Informasi jumlah byte dapat diperoleh dari utilitas *ls* dengan option *-l*. Dengan itu maka informasi ukuran byte suatu file berada pada kolom ke-5. Tanda '$'menunjukkan kolom yang diambil. Sehingga dengan $5 dapat diperoleh nilai byte suatu file dari utilitas *ls -l.*

```
$ ls -l files | awk '{ x += $5 }
     END { print "total bytes: " x }'
```

g) Berikut ini program untuk mencetak jumlah kilo-bytes yang digunakan dalam sebuah file.

```
$ ls -l files | awk '{ x += $5 }
     END { print "total K-bytes:", x / 1024 }'
```

h) Berikut ini program untuk mencetak daftar semua user yang sudah diurutkan. Data user pada output berada pada kolom ke-1, sehingga dapat ditampilkan dengan perintah print $1.

```
$ awk -F: '{ print $1 }' /etc/passwd | sort
```

i) Berikut ini program untuk mencetak jumlah baris dalam sebuah file. *Built-in variable NR* menyimpan jumlah record yang telah dibaca. Secara default, jumlah record juga merepresentasikan jumlah baris (*row*), karena satu record adalah satu baris dengan RS (*record separator*) nya adalah *new line character*.

```
$ awk 'END { print NR }' data
```

j) Berikut ini program untuk mencetak baris genap dalam sebuah file. Ketika membaca suatu baris dengan hitungan yang ada pada variable NR, jika sisa hasil pembagiannya (modulus) dengan dua adalah nol maka baris yang sedang dibaca tersebut adalah baris genap, dan jika modulusnya tidak nol (satu) maka menunjukkan baristersebut baris bernomor ganjil. Jika digunakan karakter 'NR % 2 == 1' program akan mencetak baris ganjil.

```
$ awk 'NR % 2 == 0' data
```

<u>Percobaan 3: Contoh program dengan aturan ganda (*two rules*)</u>

Utilitas awk membaca file input per satu baris dalam satu waktu. Untuk masing-masing baris, awk mencoba pola tiap aturan. Jika beberapa pola sesuai, maka beberapa aksi dilakukan dengan urutan dimana pola tersebut muncul di program awk. Jika tidak ada pola yang sesuai, maka tidak ada aksi yang dilakukan.

```
$ awk '/12/ { print $0 }
       /21/ { print $0 }' BBS-list inventory-shipped
```

Lihat hasilnya, akan tampak bahwa baris "sabafoo" pada BBS-list tercetak dua kali, karena memenuhi kedua aturan.

Percobaan 4: Contoh program yang lebih kompleks

Berikut ini menunjukkan bagaimana awk dapat digunakan untuk menyimpulkan, memilih, dan menata ulang output dari utilitas lain. Misalnya dari list semua file yang kita punya sebagai berikut:

```
-rw-r--r-- 1 arnold user 1933    Nov   7     13:05 Makefile
-rw-r--r-- 1 arnold user 10809   Nov   7     13:03 awk.h
-rw-r--r-- 1 arnold user 983     Apr   13    12:14 awk.tab.h
-rw-r--r-- 1 arnold user 31869   Jun   15    12:20 awkgram.y
-rw-r--r-- 1 arnold user 22414   Nov   7     13:03 awk1.c
-rw-r--r-- 1 arnold user 37455   Nov   7     13:03 awk2.c
-rw-r--r-- 1 arnold user 27511   Dec   9     13:07 awk3.c
-rw-r--r-- 1 arnold user 7989    Nov   7     13:03 awk4.c
```

Kemudian Anda ingin menampilkan jumlah file yang tercatat dibuat pada bulan November. Perintahnya adalah sebagai berikut:

```
$ LC_ALL = C ls -l | awk '$6 == "Nov" { sum += $5 }
     END { print sum }'
```

Percobaan 5: Pernyataan awk dengan perubahan baris

Sering kali, setiap baris dalam program awk merupakan statement terpisah atau aturan terpisah, seperti ini:

```
$ awk `/12/ { print $0 }
       /21/ { print $0 }' BBS-list inventory-shipped
```

Bagaimanapun, awk dapat mengabaikan baris baru setelah salah satu dari simbol dan keyword berikut ini:

```
     ,     {     ?     :     ||    &&    do    else
```

Selain itu baris baru diangap akhir dari statement. Cobalah masing-masing simbol-simbol tersebut untuk memisahkan program diatas.

Jika Anda ingin membagi sebuah pernyataan tunggal menjadi dua baris pada suatu titik di mana baris baru akan menghentikannya, Anda bisa melanjutkannya dengan mengakhiri baris pertama dengan karakter *backslash* ('\'). Backslash harus menjadi karakter terakhir pada baris agar dapat diakui sebagai karakter berkelanjutan (*continuation*). "*Backslash continuation"* diperbolehkan di mana saja di pernyataan, bahkan di tengah *string* atau ekspresi reguler (*regular expression*). Lebih jelasnya dapat dilihat pada perintah awk berikut ini:

```
$ awk '/This regular expression is too long, so continue it\
on the next line/ { print $1 }'
```

Tetapi jangan terlalu sering menggunakan *backslash continuation*, karena dapat menyulitkan pembacaan program. Gunakanlah sewajarnya sehingga program tetap dapat terbaca dengan baik. Dengan demikian, untuk portabilitas program awk, yang terbaik adalah tidak membagi baris Anda di tengah-tengah *regular expression* atau *string*.

```
$ awk 'BEGIN {
> print \
> "hello, world"
> }'
= hello, world
```

Awk adalah bahasa yang "*line-oriented*". Setiap tindakan aturan ini harus dimulai pada baris yang sama sebagai pola. Untuk memiliki pola dan tindakan pada baris yang berbeda, Anda harus menggunakan *backslash continuation*, tidak ada pilihan lain. Lebih lanjut backslash paling berguna ketika program awk berada dalam file sumber (*source file*) yang terpisah, dan tidak ada dalam dari baris perintah. Bagaimana penerapannya?

Satu hal yang perlu diingat adalah bahwa *backslash continuation* dan komentar tidak bercampur. Begitu awk melihat '#' yang dipakai untuk memulai komentar, dan akan mengabaikan segala sesuatu pada baris setelahnya. Sebagai contoh:

```
$ awk 'BEGIN { print "dont panic" # a friendly \
> BEGIN rule
> }'
error awk: cmd. line:2: BEGIN rule
error awk: cmd. line:2: ^ parse error
```

Ketika program awk dibuat dalam satu aturan yang singkat, Anda mungkin ingin menempatkan lebih dari satu aturan pada garis. Hal ini dilakukan dengan memisahkan pernyataan dengan tanda titik koma (';'). Hal ini juga berlaku untuk aturan sendiri. Dengan demikian, program bisa ditulis dengan cara sebagai berikut:

```
$ awk '/12/ { print $0 } ; /21/ { print $0 }' BBS-list
inventory-shipped
```

Persyaratan yang menyatakan bahwa aturan-aturan pada baris yang sama harus dipisahkan dengan titik koma bukan merupakan bahasa asli awk, melainkan telah ditambahkan untuk konsistensi pernyataan yang dibuat dengan tindakan yang akan dijalankan.