Praktikum 5
Mencetak Output

Tujuan Pembelajaran

Mahasiswa dapat memahami dan menggunakan perintah mencetak output dalam bahasa pemrograman awk.

Dasar Teori

Salah satu intruksi yang paling sering digunakan dalam pemrograman adalah instruksi untuk mencetak/mengeluarkan sebagian maupun seluruh dari hasil input. Pada awk terdapat instruksi *print* untuk mencetak output sederhana dan *printf* untuk mencetak output yang lebih kompleks dengan berbagai format. Instruksi *print* tidak terbatas oleh nilai seperti apa yang hendak dicetak, namun ada dua pengecualian yaitu: tidak dapat menentukan bagaimana mencetaknya, dan bagaimana dapat menentukan formatnya (kolom, notasi eksponensial, notasi matematika, dan sebagainya). Sehingga untuk mencetak dengan spesifikasi tertentu digunakan instruksi printf.

Fungsi *print* digunakan untuk mencetak output sederhana dengan format standar. Fungsi print tidak memerlukan argumen format untuk mencetak, sehingga tidak mendukung pengaturan lebar output cetak maupun notasi. Fungsi print secara otomatis menambahkan karakter newline pada akhir pencetakan. Fungsi *printf* dapat digunakan untuk mencetak output dengan format yang lebih kompleks. Printf mendukung argumen format yang dapat menentukan lebar kolom, notasi penulisan, dan berbagai format lainnya, seperti pada format printf dalam bahasa C. Format printf diatur dengan format-control letters dan modifier. Standard ouput dari fungsi print maupun printf dapat dialihkan dengan mengunakan redirection. Fungsi print dan printf mendukung redirection untuk stadard output, standar error, maupun standard descriptor.

Selain mencetak dasar dan mencetak dengan format tertentu, bab ini juga mencakup pengalihan I/O (input/output) ke file dan pipa, memperkenalkan nama file khusus yang ada pada proses internal gawk, dan membahas built-in function *close()*.

Percobaan 1: Menggunakan perintah *print*

Perintah *print* digunakan untuk menghasilkan output sederhana (format standar). Caranya hanya dengan menentukan string atau angka yang akan dicetak dalam suatu daftar yang dipisahkan dengan karakter koma (‘,’). Output yang dihasilkan dipisahkan dengan sebuah karakter spasi (‘ ‘), kemudian diikuti oleh karakter newline. Perintahnya seperti dibawah ini:

```
print item1, item2, . . .
```

Item untuk mencetak dapat berupa konstanta string atau angka, field atau rekord saat ini (misalnya: $1), variabel, atau ekspresi awk. Nilai numerik dapat dikonversi menjadi string dan kemudian dicetak.

Perintah *print* sederhana tidak memiliki item, itu setara dengan perintah “*print $0*” yang artinya mencetak seluruh rekord saat ini. Untuk mencetak baris kosong, gunakan 'print ""', di mana "" adalah string kosong. Untuk mencetak sepotong tetap teks, menggunakan string konstan, seperti string "jangan panik", sebagai satu item. Jika Anda lupa untuk menggunakan karakter *double-quotes*, teks diambil sebagai ekspresi awk, dan Anda mungkin akan mendapatkan pesan kesalahan. Perlu diingat bahwa spasi dicetak antara dua item.

```
$ awk ‘BEGIN { print "jangan panik" }’
$ awk “BEGIN { print \"jangan panik\" }”
```

Percobaan 2: Contoh pernyataan *print*

Setiap perintah print menghasilkan sedikitnya satu baris output. Namun,perintah ini tidak hanya terbatas satu baris. Jika sebuah nilai item adalah string yang berisi karakter newline, maka karakter newline tersebut juga akan membentuk baris pada output. Sehingga dengan satu perintah print dapat dihasilkan output beberapa baris.

```
awk ‘BEGIN { print "line one\nline two\nline three" }’
```

Contoh berikut ini mengambil dari file *inventory-shipped*:

```
awk ’{ print $1, $2 }’ inventory-shipped
```

Pada percobaan diatas print untuk mencetak field-1 dan field-2 pada file inventory-shipped. Perintah print tersebut memisahkan setiap item yang akan dicetak dengan koma, sehingga hasilnya output setiap field secara otomatis dipisahkan dengan spasi. Contoh berikut menunjukkan print tanpa menggunakan koma:

```
awk '{ print $1 $2 }' inventory-shipped
```

Pada perintah print di atas setiap item-nya tidak dipisah dengan koma, hal ini akan menghasilkan output nilai field yang tidak dipisahkan dengan spasi. Karena item yang tidak dipisahkan dengan tanda koma akan dikerjakan (dicetak) secara bersamaan dan hasilnya akan digabung. Pencetakan item dengan cara seperti ini dapat mengakibatkan kerancuan informasi yang dihasilkan. Terutama apabila yang dicetak adalah data tabel yang terdiri dari beberapa field.

Pemberian header sebelum mencetak field data dapat memperjelas hasil output. Pencetakan header tersebut dapat dilakukan pada blok BEGIN dari program awk. Seperti pada contoh berikut ini:

```
$ awk 'BEGIN { print "Month Crates"
               print "----- ------" }
             { print $1, $2 }' inventory-shipped
```

Program diatas memberikan heading month ($1) dan crates ($2) dengan menggunakan perintah BEGIN yang dieksekusi pertama kali oleh awk. Tetapi hasil dari program di atas tidak lurus dan kurang rapi, sehingga tetap sulit untuk dibaca. Caranya mengatasi hal ini adalah menambahkan spasi diantara 2 field, seperti pada contoh berikut ini:

```
$ awk 'BEGIN { print "Month Crates"
               print "----- ------" }
             { print $1, " ", $2 }' inventory-shipped
```

Jika menggunakan perintah print maka untuk mengatur lebar kolom/field agar sesuai dengan jarak header dilakukan dengan mencetak beberapa spasi untuk menyesuaikan lebar kolom, karena perintah print tidak mendukung format khusus untuk lebar kolom setiap item.

Percobaan 3: *Output Separator*

Perintah print memproses input yang berupa daftar item yang dipisahkan dengan karakter koma dan menghasilkan output yang setiap itemnya dipisahkan dengan karakter spasi (secara default). Pemisah output tersebut tersimpan pada built-in variable OFS (*output field separator*). Nilai awal (default) dari variable tersebut adalah sebuah karakter spasi (" ").

Disisi lain, perintah print akan menghasilkan satu output yang keseluruhannya adalah satu record. Pemisah default dari record adalah karakter *newline* ("\n"). Namun pada hasil outputnya mungkin saja satu output tersebut berupa beberapa baris bila dalam intem inputnya mengandung karakter *newline*.

Normalnya pemisah record keluaran adalah *newline* yang disimpan pada built-in variable ORS (*output record separator*).

Jika kita ingin merubah bagaimana output field dan record dipisahkan, berikan nilai yang baru pada variabel OFS dan ORS pada saat BEGIN. Hal ini akan dilakukan sebelum semua input diproses. Seperti pada contoh berikut ini:

```
$ awk 'BEGIN { OFS = ";" ; ORS="\n\n" }
>            { print $1, $2}' BBS-list
```

Pada program di atas, OFS diberi nilai string ";" sedangkan ORS dengan string "\n\n". Tampak setiap field pada hasil output dipisahkan dengan karakter ";", dan record dipisahkan dengan dua newline, sehingga tampak terdapat satu baris kosong diantara masing-masing record. Jika nilai dari ORS tidak mengandung *newline,* maka output dari program akan ditampilkan dalam satu baris. Seperti pada contoh berikut ini:

```
$ awk 'BEGIN { OFS = ";" ; ORS="#" }
>            { print $1, $2}' inventory-shipped
```

Percobaan 4: Mengontrol output numerik dengan perintah *print*

Perintah print sendiri sebenarnya tidak mendukung format khusus, karena print hanya mencetak secara sederhana dengan format yang standar. Ketika mencetak nilai numerik dengan perintah print, awk secara langsung akan mengkonversi nilai numerik menjadi kumpulan string dari karakter. Tapi awk, secara internal, menggunakan fungsi *sprintf()* untuk mengkonversi format. Fungsi *sprintf()* mengukung format untuk string maupun angka sehingga sebuah nilai dapat direpresentasikan dalam beberapa format output.

Built-in variabel yang bernama OFMT mengandung spesifikasi format default yang digunakan untuk menampilkan output dengan perintah *sprintf()* ketika akan dilakukan konversi dari numerik menjadi string untuk keperluan tampilan. Nilai default dari OFMT adalah "%.6g". Cara perintah *print* mencetak numerik dapat diubah dengan cara mengganti nilai dari variabel OFMT seperti pada contoh berikut:

```
$ awk 'BEGIN {
> print 17.23, 17.54
> OFMT = "%.0f" #print numbers as integers (rounds)
> print 17.23, 17.54 }'
```

Pada program di atas variable OFMT (output format) menyimpan format yang akan digunakan untuk fungsi sprintf dalam perintah print. Format dalam variable teresebut digunakan untuk mengkonversi number (angka) ke dalam string.

Dengan mengubah format angka menjadi string, maka untuk dimungkinkan untuk mencetaknya dengan perintah print. Format "%.0f" yang diberikan dalam variable OFMT, adalah format untuk mencetak bilangan float tanpa angka dibelakang koma, sehingga hasilnya adalah menjadi nilai bulat (integer). Namun, OMFT memiliki keterbatasan hanya untuk konversi format bilangan float.

Percobaan 5: Menggunakan perintah *printf* untuk hasil yang lebih baik

Perintah *printf* mendukung pengaturan fotmat output yang lebih baik daripada printah *print*. *Printf* dapat digunakan untuk menentukan lebar masing-masing item, dan berbagai variasi pilihan format untuk angka/bilangan. Format output pada perintah printf diberikan pada argument berupa string yang berisi format, yang disebut format string yang dapat mengatur bagaimana dan dimana mencetak argumen-argumennya.

a. Pengenalan perintah *printf*

Format perintah *printf* secara sederhana seperti berikut.

```
printf format, item1, item2, …
```

Seluruh daftar argumen dapat dipilih di dalam tanda kurung. Tanda kurung diperlukan jika salah satu item ekspresi menggunakan operator relasional '>', jika tidak, dapat membingungkan dengan penggunaan "*output redirection*".

Perbedaan antara *printf* dan *print* adalah pada format argumen. Ini adalah sebuah ekspresi yang nilainya diambil dari sebuah string. Hal ini menunjukkan bagaimana cara mengeluarkan setiap dari argumen lainnya, inilah yang disebut dengan format string.

Format string tersebut hampir sama dengan fungsi *printf()* pada *ISO C library*,

Fungsi *printf* tidak secara automatis menambahkan sebuah *newline* pada outputnya. Fungsi tersebut hanya memberikan keluaran yang sesuai dengan format string yang ditulis. Jadi jika sebuah newline diperlukan, Anda harus memasukkanya pada format string. Variabel output separator OFS dan ORS tidak akan berpengaruh pada pernyataan *printf*, seperti pada contoh berikut ini:

```
$ awk 'BEGIN { ORS = "\nOUCH!\n" ; OFS="+"
               msg = "DontPanic!"
               printf "%s\n",
               msg
             }'
```

Pada program di atas, format yang diberikan adalah "%s\n". Format pada *printf* sangat mirip dengan format yang digunakan dalam bahasa pemrograman C. Format "%s" adalah control format untuk string. "\n" ditambahkan pada akhir dari argument format jika menghendaki output diakhiri nengan *newline*, karena *printf* tidak secara otomatis menambahkan *newline* pada akhir output. Tampak pada program nilai ORS diberikan "\nOUCH!" dan "+" untuk OFS, namun pada output kedua separator tersebut tidak muncul. Karena *printf* menggunakan format tersendiri yaitu dengan format argumen, maka variable ORS dan OFS tidak berpengaruh terhadap output dari printf.

b. Format-contol letters

Format pada printf ditentukan dengan "*format-control letter"*, yaitu huruf khusus yang diawali dengan karakter '%'. Format-control letter menentukan bagaimana bentuk dari suatu nilai yang hendak dicetak. Keseluruhan format terdiri dari pilihan *modifiers* yang mengatur bagaimana mencetak sebuah nilai, seperti lebar data (*field width*). Tabel berikut ini menunjukkan *format-control letter:*

Tabel: *format-control letter*

format-control letter	Penjelasan
%c	Karakter control %c mencetak bilangan sebagai sebuah karakter ASCII. Maka, 'printf "%c", 65' akan menghasilkan output huruf 'A' (65 adalah kode ASCII untuk huruf 'A'). Karena %c mencetak huruf/karakter dari kode ASCII, maka nilai yang dapat diproses dengan %c adalah antara 0 sampai 255.
%d, %i	Karakter control %d maupun %i menghasilkan output yang sama. Kedua karakter control tersebut sama, yaitu untuk mencetak bilangan bulat (integer).
%e, %E	Karakter control %e dan %E menghasilkan output yang sama, yaitu berupa format bilangan scientific (eksponensial). Format "%4.3e" maupun "%4.3E" menghasilkan format bilangan eksponensial dengan empat angka penting termasuk tiga angka dibelakang koma diikuti notasi eksponensial (10 pangkat), hanya perbedaannya adalah simbol eksponensial-nya, jika %e maka symbol eksponensialnya adalah 'e', sedangkan %E maka simbol eksponensial-nya adalah 'E'.
%f	Karakter control "%f" mencetak bilangan dalam notasi floating-point. Perintah 'printf "%4.3f", 1950' menghasilkan output 1950.000, oleh karena format "%4.3f" adalah untuk representasi bilangan floating-point yang terdiri dari empat angka penting dengan tiga angka dibelakang koma.

%F	Karakter control %F mirip dengan %f tapi, untuk nilai tak terhingga dan bukan bilangan, direpresentasikan dengan kata yang dicetak dalam huruf capital. Tidak semua sistem mendukung format "%F", sehingga gawk menggunakan format "%f".
%g, %G	Karakter control "%g" maupun "%G" menghasilkan representasi dalam notasi scientific atau floating-point.
%o	Karakter control "%o" mencetak bilangan bulat oktal tidak bertanda (unsigned octal integer).
%s	Karakter control "%s" mencetak string.
%u	Karakter control "%u" mencetak bilangan bulat pecahan tidak bertanda. Format ini jarang digunakan, karena pada dasarnya semua bilangan dalam awk adalah floating-point. Format ini hanya untuk mendukung kompatibilitas dengan C.
%x, %X	Karakter control "%x" maupun "%X" menghasilkan format bilangan bulat heksadesimal, hanya perbedaannya adalah jika %x maka menggunakan 'a'sampai 'f', sedangkan %X menggunakan 'A' sampai 'F'.
%%	Karakter control '%' atau '%%' bukan merupakan control format, malainkan akan menghasilkan/mencetak karakter '%'.

Contoh dari karakter kontrol diatas dapat dilihat pada program berikut:

Karakter kontrol c, d dan i

```
$ awk 'BEGIN { printf "%c\n", 65}'
$ awk 'BEGIN { printf "%d\n", 65}'
$ awk 'BEGIN { printf "%i\n", 65}'
```

Karakter kontrol e dan E

```
$ awk 'BEGIN { printf "%4.3e\n", 1950}'
$ awk 'BEGIN { printf "%4.3E\n", 1950}'
$ awk 'BEGIN { printf "%4.3e\n", (sqrt(100))}'
```

Karakter kontrol f dan F

```
$ awk 'BEGIN { printf "%4.3f\n", 1950}'
$ awk 'BEGIN { printf "%4.3F\n", 1950}'
```

Karakter kontrol g dan G

```
$ awk 'BEGIN { printf "%g\n", 1950}'
$ awk 'BEGIN { printf "%G\n", 1950}'
```

Karakter kontrol o, s dan u

```
$ awk 'BEGIN { printf "%o\n", 1950}'
$ awk 'BEGIN { printf "%s\n", "1950 tahun"}'
$ awk 'BEGIN { printf "%u\n", "1950"}'
$ awk 'BEGIN { printf "%u\n", 1950.34}'
```

Karakter kontrol x dan X

```
$ awk 'BEGIN { printf "%x\n", 7331}'
$ awk 'BEGIN { printf "%X\n", 7331}'
```

Karakter kontrol % dan %%

```
$ awk 'BEGIN { printf "%\n" }'
$ awk 'BEGIN { printf "%%\n" }'
```

c. *Modifier* untuk format *printf*

Pemberian format juga dapat memasukkan modifier yang menentukan jumlah nilai item yang akan dicetak, hal ini juga dapat menentukan lebar spasi yang diperlukan. Modifier diberikan di antara '%' dan format-control letter. Tabel berikut ini menunjukkan modifier yang dapat digunakan untuk format *printf*.

Tabel: Modifier untuk format *printf*

Modifier	Penjelasan
N$	Modifier karakter *N* adalah konstanta bilangan bulat sebelum tanda '$' menyatakan posisi item. Penentuan format ini diterapkan pada argumen bedasarkan urutan yang diberikan dalam format string. Dengan penentuan posisi, maka urutan item yang akan ditampilkan dapat diatur (tidak harus urut).
-	Modifier karakter tanda minus '-' diberikan sebelum width modifier, digunakan untuk menjadikan rata kiri. Normalnya (tanpa tanda minus), untuk bilangan akan dijadikan rata kanan dalam space yang ditempati, namun dengan tanda tersebut bilangan dijadikan rata kiri, sehingga space yang lebih berada di kanan.
spasi	Modifier karakter spasi digunakan untuk konversi numeric, memberikan awalan berupa spasi untuk nilai positif dan awalan tanda minus untuk nilai negatif.
+	Modifier karakter plus '+', diberikan sebelum width modifier, digunakan agar selalu memberikan tanda (positif/negatif) untuk bilangan walaupun bilangan

	tersebut positif.
#	Modifier karakter pagar '#' menggunakan bentuk alternatif untuk beberapa control letter. Untuk "%o", akan memberikan awalan nol (0). Untuk "%x" dan "%X" memberikan awalan "0x" atau "0X" untuk hasil (bilangan) bukan nol. Untuk "%e", "%E", dan "%f" hasilnya akan selalu mengandung desimal.
0	Modifier karakter awalan '0' (nol) berfungsi sebagai penanda yang menandakan bahwa output akan ditambah dengan 0 (nol) daripada spasi. Modifier ini hanya berfungsi pada format output bilangan, dan ketika ukuran (width) yang ditentukan lebih lebar dari output nilai yang dicetak.
'	Modifier karakter petik tunggal (') atau apostrophe digunakan untuk memberikan notasi pemisah ribuan (seperti dalam penulisan mata uang).
width	Modifier "width" adalah bilangan yang diberikan untuk menyatakan lebar minimal sebuah field. Sehingga, lebar pencetakan akan sebesar jumlah karakter ehingga, lebar pencetakan akan sebesar karakter yang jumlahnya ditentukan dengan modifier ini. Format "%4s" akan menjadikan output yang kurang dari 4 karakter akan diperlebar menjadi 4 karakter dengan cara menambahkan spasi. Jika outputnyaj lebih dari 4 karakter, maka pencetakannya akan mengikuti jumlah karakter output.
. prec	Modifier karakter ". prec" Menentukan presisi bilangan yang digunakan dalam pencetakan ouput.

Contoh dari modifier diatas dapat dilihat pada program berikut:

Modifier karakter indeks (*N*$)

```
$ awk 'BEGIN { printf "%s %s\n", "satu", "dua"}'
$ awk 'BEGIN { printf "%2$s %1$s\n", "satu", "dua"}'
```

Modifier karakter minus (-)

```
$ awk 'BEGIN { printf "%-4s\n", "foo"}'
$ awk 'BEGIN { printf "%-4sbar\n", "foo"}'
```

Modifier karakter spasi, - dan +

```
$ awk 'BEGIN { printf "%d \t %d\n", -1, 1}'
$ awk 'BEGIN { printf "% d \t % d\n", -1, 1}'
```

```
$ awk 'BEGIN { printf "%-d \t %-d\n", -1, 1}'
$ awk 'BEGIN { printf "%+d \t %+d\n", -1, 1}'
```

Modifier karakter pagar (#)

```
$ awk 'BEGIN { printf "%#o\n", 19}'
$ awk 'BEGIN { printf "%#x\n", 19}'
$ awk 'BEGIN { printf "%#e\n", 199}'
$ awk 'BEGIN { printf "%#E\n", 199}'
$ awk 'BEGIN { printf "%#f\n", 199}'
$ awk 'BEGIN { printf "%#f\n", 199.99}'
$ awk 'BEGIN { printf "%#G\n", 199}'
$ awk 'BEGIN { printf "%#g\n", 199}'
```

Modifier karakter nol (0)

```
$ awk 'BEGIN { printf "%01d\n", 999}'
$ awk 'BEGIN { printf "%02d\n", 999}'
$ awk 'BEGIN { printf "%03d\n", 999}'
$ awk 'BEGIN { printf "%04d\n", 999}'
$ awk 'BEGIN { printf "%05d\n", 999}'
```

Modifier karakter single quote (‘)

```
$ cat thousands.awk
  BEGIN { printf "%'d\n", 1234567 }
$ LC_ALL=C gawk -f thousands.awk
  1234567
$ LC_ALL=en_US.UTF-8 gawk -f thousands.awk
  1,234,567
```

Modifier karakter width ()

```
$ awk 'BEGIN {printf "5s\n", "foo" }'
$ awk 'BEGIN {printf "5s\n", "foobar" }'
```

Modifier karakter titik (.)

```
$ awk 'BEGIN {printf "%.2f\n", 123.456 }'
$ awk 'BEGIN {printf "%.2e\n", 123.456 }'
$ awk 'BEGIN {printf "%.2E\n", 123.456 }'
$ awk 'BEGIN {printf "%.2g\n", 123.456 }'
$ awk 'BEGIN {printf "%.2G\n", 123.456 }'
$ awk 'BEGIN {printf "%.2d\n", 123.456 }'
$ awk 'BEGIN {printf "%.2i\n", 123.456 }'
$ awk 'BEGIN {printf "%.2o\n", 123.456 }'
$ awk 'BEGIN {printf "%.2u\n", 123.456 }'
```

```
$ awk 'BEGIN {printf "%.2x\n", 123.456 }'
$ awk 'BEGIN {printf "%.2s\n", 123.456 }'
```

Modifier karakter titik presisi (. prec)

```
$ awk 'BEGIN { x=5 y=3 s="abcdefg"
               printf "%*.*s\n", x,y,s }'
$ awk 'BEGIN { x=5 y=3 s="abcdefg"
               printf "%5.3s\n", s }'
$ awk 'BEGIN { x=5 y=3 s="abcdefg"
               printf "%" x "." y "s\n", s }'
```

d. Contoh penggunaan *printf*

Program berikut ini adalah contoh penggunaan *printf* untuk membuat tabel dengan perataan teks.

```
$ awk '{ printf "%-10s %s\n", $1, $2 }' BBS-list
```

Program tersebut mencetak nama bulletin board ($1) dalam file "BBS-list" sebagai string yang terdiri dari 10 karakter yang rata kiri. Program tersebut juga mencetak nomer telepon ($2) disebelahnya. Nomor telepon dicetak dengan format string "%s" karena dalam nomor telepon tersebut mengandung karakter dash ('-'), jika menggunakan format number, maka angka setelah dash tidak akan dicetak. Pada format nomor telepon juga tidak perlu ditentukan lebarnya, karena semua item memiliki ukuran (jumlah karakter) yang sama dan berada pada kolom terakhir. Hasilnya adalah tabel dengan dua kolom yang rata teks yang memuat data nama dan nomor telepon. Output Field nama lurus rata kiri dalam kolom yang lebarnya sepuluh karakter. Agar tabel tersebut terlihat lebih baik perlu ditambahkan *header* di atas kolom dengan menggunakan perintah BEGIN.

```
awk 'BEGIN { print "Name       Number"
             print "----       ------" }
     { printf "%-10s %s\n", $1, $2 }' BBS-list
```

Untuk memberikan header dilakukan dengan mencetaknya dalam blok BEGIN, karena *header* dicetak hanya pada awal. Jika menggunakan perintah print, maka pengaturan posisi header/lebar kolom dilakukan secara manual, yaitu dengan menambahkan beberapa spasi sehingga jaraknya bersesuaian. Dengan pengaturan yang sesuai, baik print dapat menghasilkan output yang sama dengan printf dalam pencetakan format sederhana.

Berikut ini penggunaan digunakan perintah *printf* untuk menghasilkan seperti pada program di atas:

```
awk 'BEGIN { printf "%-10s%s\n", "Name", "Number"
             printf "%-10s%s\n", "----", "------" }
     { printf "%-10s%s\n", $1, $2 }' BBS-list
```

JIka menggunakan perintah *printf*, maka pengaturan posisi header/lebar kolom dilakukan dengan memberikan *width modifier* pada *format argument*. Dengan *printf*, pencetakan masing-masing kolom menggunakan format yang sama sehingga baik header sama seperti kolom dibawahnya.

Jika terdapat beberapa printf yang menggunakan format yang sama, maka dapat disederhanakan dengan menyimpan format tersebut dalam satu variable, yang kemudian variable tersebut dapat digunakan untuk beberapa printf, seperti pada contoh dibawah ini:

```
awk ' BEGIN { format = "%-10s%s\n"
              printf format, "Name", "Number"
              printf format, "----", "------" }
      { printf format, $1, $2 }' BBS-list
```

Selanjutnya cobalah untuk menggunakan pernyataan *printf* untuk menuliskan data pada file inventory-shipped.

Percobaan 6: Redirecting Output dengan print dan printf

Sebetulnya ouput dari *print* maupun *printf* dapat dialihkan ke output lain, selain standard output pada monitor. Pengalihan tersebut menggunakan *redirection*. Redirection diberikan setelah perintah print aau printf. Perintah redirection dapat dilakukan melalui shell command atau dituliskan dalam program awk.Ada empat bentuk dari *output redirection* yang ditunjukkan untuk pernyataan print dan printf:

a. print items > output-file

Bentuk redirection diatas mencetak *items* ke file yang bernama *output-file*. Nama output-file dapat berbeda-beda, dirubah dahulu menjadi string kemudian digunakan sebagai nama file. Ketika digunakan, *output-file* dihapus dahulu sebelum ditulis. Jika ada nama output-file yang sama, tidak akan menghapusnya tetapi hasilnya akan ditambahkan (berbeda dengan konsep redirection pada *shell scripts*). Jika output-file tidak ada, maka akan dibuatkan.

Contoh berikut ini adalah program awk yang dapat menuliskan daftar dari nama BBS pada file yang bernama “name-list” dan daftar nomor telepon ke file lainnya yang bernama “phone-list”.

```
$ awk '{ print $2 > "phone-list"
         print $1 > "name-list" }' BBS-list
$ cat name-list
$ cat phone-list
```

Pada program di atas, output dari 'print $2' dialihkan ke file phone-list, dan output dari 'print $1' dialihkan ke file name-list. Tipe redirection tersebut menggunakan notasi tanda '>', yang berarti bahwa sebelum penulisan pertama ke dalam file tersebut dilakukan, terlebih dulu isi file tersebut dikosongkan (dihapus). Namun, untuk penulisan yang berkelanjutan tidak menghapus hasil penulisan sebelumnya.

Dengan redirection program di atas, maka file "name-list" berisi daftar nama, dan file "phone-list" berisi daftar nomor telepon. Setiap output file berisi satu nama/nomor telepon pada setiap baris, karena menggunakan perintah *print* yang secara otomatis akan menambahkan newline pada akhir output record.

b. print items >> output-file

Redirection pada program diatas menggunakan perintah *print*. Tipe redirection tersebut mencetak item kedalam output file yang telah ada sebelumnya (atau jika belum ada, maka file akan dibuat terlebih dulu).

```
$ awk '{ print $2 >> "phone-list"
         print $1 >> "name-list" }' BBS-list
$ cat name-list
$ cat phone-list
```

Perbedaanya dengan notasi '>' tunggal adalah, tipe redirection ini ('>>') tidak menghapus isi awal dari file output, melainkan menambahkan pada akhir file.

c. print items | command

Selain redirection ke dalam file, output print juga dapat dilakukan dengan menggunakan pipe. Tipe redirection tersebut melakukan pipe (lanjutan perintah) ke command, dan menulis nilai dari item melalui pipe tersebut dengan perintah yang diberikan dalam command. Command adalah argument yang berisi ekspresi awk. Nilai dari command dikonversi ke string yang isinya menjadi shell command yang akan dijalankan seperti pada contoh berikut ini.

```
$ awk '{ print $1 > "name.unsorted"
         command = "sort -r > name.sorted"
         print $1 | command }' BBS-list
```

Program di atas menghasilkan dua buah file output. File “name.unsorted” berisi daftar tidak urut dari nama dalam BBS-list, sedangkan file “name.sorted” berisi daftar yang terurut secara kebalikan urutan alphabetis. Daftar yang tidak terurut ditulis dengan redirection biasa, sedangkan daftar yang terurut ditulis dengan melakukan pipe melalui utilitas sort. Command “*sort –r”* pada program tersebut mengurutkan daftar secara kebalikan urutan apfabetis.

Berikut ini adalah potongan program yang menggunakan redirection untuk mengirim pesan ke “mail bug-system”. Hal ini bisa berguna ketika terjadi masalah pada sistem awk yang berjalan secara periodik untuk keperluan maintenance sistem. Cobalah untuk mengimplementasikan program dibawah ini:

```
report = "mail bug-system"
print "Awk script failed:", $0 | report
m = ("at record number " FNR " of " FILENAME)
print m | report
close(report)
```

d. print items |& command

Tipe redirection ini digunakan untuk mencetak item ke dalam input dari command. Perbedaan jenis redirection ini dengan single-‘|’ adalah output dari command dapat dibaca dengan getline. Dengan demikian, command akan menjadi coprocess (proses kedua) yang bekerja bersamaan dengan program awk, atau dapat juga sebagai perintah tambahan.

```
$ awk '{ print $1 > "name.unsorted"
         command = "sort -r > name.sorted"
         print $1 |& command }' BBS-list
```

Kesimpulannya, penggunaan redirection menggunakan ‘>’, ‘>>’, ‘|’, atau ‘|&’ adalah meminta sistem untuk membuka sebuah file, pipe, atau coprocess hanya jika file atau command yang dimaksudkan belum ditulis oleh program Anda atau jika sudah tertutup sejak yang terakhir ditulis. Jika nama file yang sama atau shell command yang sama digunakan oleh getline lebih dari satu kali selama eksekusi program awk, maka file tersebut hanya dibuka saat awal kali pertama program dijalankan. Saat itu, record pertama dari input dibaca dari file atau command tersebut. Selanjutnya, file atau command yang sama digunakan dengan getline, di mana record lain juga dibaca dari ini, demikian seterusnya. Sama hal nya, ketika sebuah file atau pipe dibuka untuk output, maka nama file atau command (perintah) yang berkaitan dengannya akan dicatat oleh awk, dan subsequent melakukan penulisan pada file tersebut atau command yang telah diberikan sebelumnya. Baik file maupun pipe tersebut akan tetap terbuka sampai awk keluar. Sehingga untuk dapat membaca lagi dari awal pada file yang sama atau mengembalikan sebuah shell command, harus dilakukan `close(filename)` untuk menutup file atau `close(command)` untuk menutup command.

Tugas

1. Dari contoh baris program “prima.c” di bawah ini:

```
#include <stdio.h>

void main()
{
      int bilangan, j, cek;
      printf(“Masukkan bilangan maksimum :”);
      scanf(“%d”, &bilangan);

      // validasi bilangan prima

      while( bilangan >= 2 ) {
            cek = 0;
            j = 2;

            while( j < bilangan ){

                  if( bilangan%j == 0 ){
                        cek = 1;
                  }
                  j++;
            }

            if ( cek == 0 ){
                  printf(“%d \n”, &bilangan");
            }
            bilangan--;
      }
}
```

Buatlah program awk yang dapat melakukan tugas-tugas dari analisis leksikal seperti pada compiler bahasa C,

- Mengecek kebenaran/validasi program
  - Jika benar akan tampil error 0
  - Jika salah akan tampil jumlah error <sekian>
    & tampil letak error pada baris <sekian> <sekian>
- Membuang komentar
- Membuang white space (spasi, tab, newline)
- Menyeragamkan menjadi huruf kecil semua atau sebaliknya
- Menyimpan hasil kompilasi pada sebuah file “target.txt”

2. Sintaks pemrograman awk mendukung beberapa file khusus yang memiliki arti tersendiri secara internal, yaitu mendukung akses ke standard file descriptor, process-related information, dan TCP/IP networking. Tugas Anda adalah membuat program awk dapat memberikan output secara serial kepada device luar (misalnya menggunakan linux tty).

Contoh:
Buatlah program di awk untuk mengkompilasi soal fisika dengan output variabel dan nilai besarannya, kemudian dikirimkan output tersebut secara serial ke arduino untuk kemudian ditampilkan ke LCD.