KATA PENGANTAR

Puji Syukur saya panjatkan kehadirat Allah SWT yang telah memberikan rahmat dan hidayah-Nya kepada saya, sehingga saya dapat menyelesaikan laporan proyek akhir Shell ini dengan lancar.

Saya menyadari bahwa laporan akhir ini masih jauh dari sempurna, oleh karena itu kritik dan saran dari semua pihak yang bersifat membangun selalu saya harapkan demi kesempurnaan program ini.

Saya juga menyampaikan terima kasih kepada semua pihak yang telah berperan serta dalam pembuatan program dan penyusunan laporan akhir ini dari awal sampai akhir. Semoga Allah SWT senantiasa meridhai segala usaha kita. Akhirnya penyusun mengharapkan semoga dari laporan yang saya buat dapat diambil hikmah dan manfaatnya sehingga dapat memberikan inpirasi terhadap pembaca. Aamiin.

Surabaya, 25 Desember 2016

Penyusun

DAFTAR ISI

BAB 1

PENDAHULUAN

1.1 Tujuan

1. Memenuhi persyaratan dalam mata kuliah shell programming.
2. Dapat menggunakan pemrograman shell untuk membuat game urut angka
3. Menggunakan aplikasi database yang membantu simulasi program tersebut.

BAB 2

DASAR TEORI

2.1 Perintah Dasar Shell Linux

a. Shell

Shell adalah program (penterjemah perintah) yang menjembatani user dengan sistem operasi dalam hal ini kernel (inti sistem operasi), umumnya shell menyediakan prompt sebagai user interface, tempat dimana user mengetikkan perintah-perintah yang diinginkan baik berupa perintah internal shell (*internal command*), ataupun perintah eksekusi suatu file progam (*eksternal command*), selain itu shell memungkinkan user menyusun sekumpulan perintah pada sebuah atau beberapa file untuk dieksekusi sebagai program. Perintah-peritah yang diketikkan oleh user dari input standart yaitu keyboard akan ditafsirkan oleh shell, jika yang diketikkan oleh user merupakan program yang dapat di eksekusi maka shell akan menjalankan program tersebut.

Contoh :

$ ls

test

$ hello

bash: hello: command not found

$

Berdasarkan contoh diatas dapat dilihat bahwa ketika user mengetikan ls maka program ls dijalankan, sedangkan apabila user mengetikkan hello dan program hello memang tidak ada maka shell tidak dapat mengeksekusinya.

b. Pengeditan Perintah

Dengan bash shell, user dapat mengedit suatu baris perintah cukup dengan tombol panah kiri dan panah kanan pada keyboard. Setelah selesai mengedit, hanya dengan menekan tombol [ENTER] untuk menjalankan perintah.

c. Pipeline

Pipeline dengan tanda vertical bar (|) adalah fasilitas di shell UNIX yang berfungsi untuk memberikan input dari suatu proses yang berasal dari output proses yang lain. Misalkan sebagai contoh :

- Sebelum kita gunakan pipeline

*$ find **

dead.letter

mbox

test

- Setelah kita gunakan pipeline

*$ find * | grep test*

test

$

Pada contoh diatas output perintah *find* menjadi input dari perintah *grep* yang kemudian hanya mengambil kata “*test*” dari output *find*.

d. Regular Expression

Regular Expression adalah cara untuk menentukan sebuah pola karakter untuk pencarian dan pemfilteran. Pola karakter yang eksak atau karakter-karakter khusus yang memiliki arti tersendiri. Penggunaan karakter khusus dalam pola regular exspression dapat dilihat pada tabel 1.

Tabel 1. Karakter khusus dalam pola regular exspression

Karakter	Arti
.	Cocok dgn sembarang satu karakter
*	Cocok dgn sembarang lebih dari satu karakter
^	Cocok dengan awal baris
$	Cocok dengan akhir baris
\<	Cocok dengan awal kata
\>	Cocok dengan akhir kata
[ ]	Cocok dgn salah satu karakter yg terdapat di dalam kurung siku
[^]	Cocok dgn salah satu karakter yg tidak terdapat pada kurung siku
\	Karakter selanjutnya dianggap literal

e. Redirection

Pada UNIX terdapat istilah standard input, standard output, dan standard error. Standart input adalah masukkan atau input standard dari suatu perintah atau program. Input standard ini adalah keyboard. Standard output adalah keluaran atau output standard dari suatu perintah atau program. Output standard ini adalah monitor atau terminal. Standard error adalah keluaran atau output standard jika pada program atau perintah terjadi error. Keluaran ini berupa pesan-pesan kesalahan yang berguna bagi pembuat program atau orang lain yang membutuhkan. Standard error biasanya adalah layar console.

Proses pembelokkan ini disebut redirection, menggunakan symbol > (membelokkan standard output ke file), symbol < (membelokkan standard input dari file).

Dengan fasilitas redirection memungkinkan user untuk dapat menyimpan output dari sebuah proses untuk disimpan ke file lain (Ouput Redirection) atau sebaliknya menggunakan isi dari file sebagai input dalam suatu proses (Input Redirection). Komponen-kmponen dari redirection adalah symbol-simbol berikut: <, >, <<. >>.

Standard input, output dan error, yaitu untuk mengalihkan file descriptor dari 0, 1 dan 2. Linux berkomunikasi dengan file melalui file descriptor yang direpresentasikan melalui angka yang dimulai dari 0, 1, 2 dan seterusnya.

Tiga buah file descriptor standar yang lalu diciptakan oleh proses adalah :

0 = keyboard (standar input)

1 = layar (standar output)

2 = layar (standar error)

Simbol untuk pembelokan adalah :

0<	atau	< pengganti standard input
1>	atau	> pengganti standard output
2>	atau	> pengganti standard error

2.2 Expresi Dan Test

Ekspresi pada bahasa-bahasa tingkat tinggi terbagi atas dua macam yaitu ekspresi kondisi dan ekspresi aritmatika. Kedua macam ekspresi ini menggunakan operator.

Operator Kondisi

||	atau (OR)
&&	dan (AND)
!	tidak(NOT)

Konstanta kondisi ada dua: 0 untuk kondisi salah (FALSE) dan 1 untuk kondisi benar (TRUE). Sedangkan untuk operator relasi yang menghasilkan TRUE atau FALSE sebagai berikut:

Operator Relasi

==	sama dengan
!=	tidak sama dengan
=~	sama dengan untuk string
!~	tidak sama dengan untuk string
<=	lebih kecil atau sama dengan
>=	lebih besar atau sama dengan
>	lebih besar
<	lebih kecil

Operator yang berhubungan dengan aritmatika adalah:

Perkalian	*
Pembagian	/
Penjumlahan	+
Pengurangan	-
Modulo (sisa hasil bagi)	%

Contoh:

($#argv > 2 || $#argv == 0) jumlah parameter sama dengan 0 atau lebih dari dua

($1 =~ -)* parameter pertama dimulai dengan – (strip)

*($nama !~ *.c)*nama tidak diakhiri dengan .c

Untuk operasi matematika ada 3 cara yang dapat digunakan, dengan statement builtin *let* atau *expr* atau *perintah subtitusi* seperti contoh berikut:

```
#memakai let
let jumlah=$a+$b
#memakai expr
bagi=`expr $a / $b`
```

#memakai perintah subtitusi $((ekspresi))

modul =$(($a%$b))

fungsi expr begitu berdaya guna baik untuk operasi matematika ataupun string contohnya:

$ mystr="linux"

$ expr length $mystr

5

Selain itu Bash pada system linux juga menyediakan statement *declare* dengan opsi *-i* hanya untuk data integer (bilangan bulat). Contohnya:

declare -i angka

angka=100;

Apabila variabel yang dideklarasikan menggunakan declare -i ternyata nilainya string (karakter), maka Bash akan mengubahnya ke nilai 0, tetapi jika tidak menggunakannya maka dianggap sebagai string.

Test

Test adalah utility sh shell yang berguna untuk memeriksa informasi tentang suatu file dan berguna untuk melakukan perbandingan suatu nilai baik string ataupun numerik

syntaxnya: ***test ekspresi*** atau ***[expression]***

Proses kerja test yaitu dengan mengembalikan sebuah informasi status yang dapat bernilai 0 (benar) atau 1 (salah) dimana nilai status ini dapat dibaca pada variabel spesial **$?**.

$ test 5 -gt 3

$ echo $?

0

pernyataan 5 -gt (lebih besar dari) 3 yang dievaluasi test menghasilkan 0 pada variabel status $? itu artinya pernyataan tersebut adalah benar, selanjutnya hasil dari evaluasi dengan expresi berikut

$ test 3 -lt 1

$ echo $?

1

status bernilai 1, berarti pernyataan salah.

Simbol -gt dan –lt pada contoh diatas disebut sebagai operator, secara sederhana operator adalah karakter khusus (spesial) yang melakukan operasi terhadap sejumlah operand, misalkan 2+3, "+" adalah operator sedangkan 2 dan 3 adalah operandnya, pada contoh test diatas yang bertindak sebagai oparatornya adalah -lt dan -gt, sedangkan bilangan disebelah kiri dan kanannya adalah operand. cukup banyak operator yang disediakan bash antara lain (dapat dilihat dengan *man bash* atau *info bash* di prompt shell):

1. **Operator untuk integer**

Operator	Keterangan
bil1 *-eq* bil2	Mengembalikan Benar jika bil1 *sama dengan* bil2
bil1 *-ne* bil2	Benar jika bil1 *tidak sama dengan* bil2
bil1 *-lt* bil2	Benar jika bil1 *lebih kecil dari* bil2
bil1 *-le* bil2	Benar jika *bil1 lebih kecil atau sama dengan* bil2
bil1 *-gt* bil2	Benar jika bil1 *lebih besar dari* bil2
bil1 *-ge* bil2	Benar jika bil1 *lebih besar atau sama dengan* bil2

2. **Operasi string**

Operator		Keterangan
-z STRING		Mengembalikan Benar jika panjang STRING adalah zero
-n STRING		Mengembalikan Benar jika panjang STRING adalah non zero
STRING1	==	Benar jika STRING1 sama dengan STRING2
STRING2		

3. **Operator file**

Operator	Keterangan
-f FILE	Mengembalikan Benar jika FILE ada dan merupakan *file biasa*
-d FILE	Benar jika FILE ada dan merupakan *direktory*

4. **Operator logika**

ekspr1 -o ekspr2	**Benar jika jika salah satu ekspresi benar *(or,||)***
ekspr1 -a ekspr2	Benar jika ekspresi1 dan ekspresi2 benar *(and,&&)*
! ekspresi	Mengembalikan Benar jika ekspresi tidak benar *(not!)*

2.3 Pengkondisian (Perintah If Dan Case)

KONSTRUKSI if

Statement builtin ***if*** berfungsi untuk melakukan seleksi berdasarkan suatu kondisi tertentu. Secara umum ada dua bentuk umum sintak perintah *if* , seperti ditunjukkan dibawah ini

Sintak :

1.
```
if  [ kondisi ]
  then
        statements
fi
```

2.
```
if  [ kondisi ]
  then
        statements
  else
        statements
fi
```

Perbedaan antara kedua bentuk adalah bentuk pertama mempunyai perintah tunggal jika ekspresi/kondisi yang diuji benar, sedangkan bentuk kedua mempunyai banyak perintah yang akan dijalankan jika ekspresi yang diuji benar.

Contoh bentuk pertama:

```
let hasil = "$b * $c"
if  [ "$hasil" = 10 ]
        then
echo "Hasil perkalian kedua bilangan = $hasil"
fi
```

Contoh bentuk kedua:

```
let hasil = "$b * $c"
if  [ "$hasil" = 10 ]
 then
        echo "Hasil perkalian kedua bilangan = $hasil"
 else
        echo "selesai"
fi
```

Kalau diperhatikan, perintah *if* ini hampir sama dengan perintah *if* pada bahasa-bahasa tingkat tinggi, seperti Pascal, C, dan juga hampir sama dengan perintah *if* pada batch file-nya DOS. Pada bentuk pertama maupun bentuk kedua dari sintak diatas adalah statement dalam blok *if...fi* akan dieksekusi apabila kondisi *if* terpenuhi. Dari kedua bentuk diatas dapat pula ditambahkan perintah untuk pengecekan kondisi dengan ***elif*** (else if), contoh sintaknya adalah sebagai berikut:

3.

```
if  [ kondisi ];
  then
        perintah1;
elif [ kondisi2 ];
  then
        perintah2;
else
        alternatif_perintah;
fi
```

klausa ***else*** akan dieksekusi jika ***if*** tidak terpenuhi, sebaliknya jika ***if*** terpenuhi maka ***else*** tidak akan dieksekusi.

Contoh bentuk ketiga:

```
if winter
then
        snowremoval
        weatherstrip
```

```
elif spring
 then
        startgarden
        mowlawn
else
        echo So        mething is wrong
fi
```

KONSTRUKSI CASE

Case digunakan untuk menyederhanakan pemakaian *if* yang berantai, sehingga dengan case, kondisi dapat dikelompokkan secara logis dengan lebih jelas dan mudah untuk ditulis. Statement ***case*** juga digunakan untuk menyeleksi kondisi majemuk, dibanding ***if***, pemakaian case lebih efisien.

Sintak :

```
case   string   in
        pilihan)
          commands
          ;;
        pilihan)
          commands
          ;;
        *)
          default commands
          ;;
esac
```

Case diakhiri dengan esac dan pada setiap kelompok instruksi diakhiri dengan ;;. Pada akhir pilihan yaitu *) yang berarti adalah "default", bila kondisi tidak memenuhi pola sebelumnya. Contoh:

```
let hasil = "$b * $c"
case  $hasil   in
```

```
        10)
         echo "Hasil perkalian kedua bilangan  = $hasil"
         ;;
        *)
    echo "Selesai"
    esac
```

2.4 Perulangan

KONSTRUKSI FOR

For digunakan untuk pengulangan dengan menggunakan variabel (*name*) yang pada setiap pengulangan akan diganti dengan nilai yang berada pada daftar (list = *word1 word2 ...*). Sintak dari perintah FOR adalah sebagai berikut:

Sintak 1:

```
for name in word1 word2 ...
do
        do-list
done
```

Contoh:

```
#!/bin/bash
for buah in apel jeruk mangga salak
do
        echo $buah adalah buah
done
```

Pada contoh program diatas variabel *$buah* akan diganti dengan data pada list yaitu apel, jeruk, mangga dan salak.

Sintak 2:

```
for name
do
        do-list
done
```

Contoh:

```
#!/bin/bash
```

```
for var
do
 echo $var
done
$./for2 satu 2 tiga
```

Contoh program menggunakan sintak2 variabel *$var* akan diganti dengan data hasil pembacaan argument (satu, 2, tiga) yang disertakan saat script dijalankan.

KONSTRUKSI WHILE

While digunakan untuk pengulangan instruksi, yang umumnya dibatasi dengan suatu kondisi. Selama kondisi tersebut TRUE, maka pengulangan terus dilakukan. Loop akan berhenti, bila kondisi FALSE, atau program keluar dari blok while melalui exit atau break. Sintak dari perintah WHILE adalah sebagai berikut:

Sintak 1. While – end

```
while  ( test_condition )
        commands /kumpulan perintah
end
```

Contoh:

```
set i=$#argv
while  ($i)
        echo –n $argv[$i]
        @i--
end
```

Pada contoh program menggunakan sintak1 akan mencetak parameter yang diterima oleh program, tetapi dalam susunan terbalik karena nilai variabel ***"i"*** dikurangi satu persatu dimulai dari nilai yang tertinggi. Perintah ***echo –n*** digunakan agar setiap kali menampilkan satu parameter, parameter berikutnya tidak akan tercetak pada baris berikutnya.

Sintak 2. While – do

while *[test_condition]*

do

commands

done

Contoh:

```
i=1;
while [ $i -le 10 ];
do
  echo "$i,";
  let i=$i+2;
done
```

Contoh program dengan sintak2 diatas menunjukkan kondisi tidak terpenuhi pada saat nilai i=11 (9+2), sehingga perintah dalam blok while tidak dieksekusi lagi dan nilai i=11 tidak pernah ditampilkan pada layar.

INSTRUKSI DUMMY

Instruksi dummy adalah instruksi yang tidak melakukan apa-apa, namun instruksi ini memberikan status exit 0 (TRUE). Oleh karena itu, instruksi dummy dapat digunakan sebagai kondisi forever pada loop (misalnya ***while***). Simbol instruksi dummy adalah □:

KONSTRUKSI UNTIL

Jika while akan mengulang selama kondisi benar, lain halnya dengan statement until yang akan mengulang selama kondisi salah, berikut contoh script menggunakan ***until***

Sintak: ***until*** *condition*

do

list

done

Contoh:

```
i=1;
until [ $i -gt 10 ];
```

```
do
  echo "$i,";
  let i=$i+1;
done
```

perhatikan kodisi until yang salah [$i -gt 10], dimana nilai awal i=1 dan akan berhenti apabila nilai i = 11 (bernilai benar) 11 -gt 10.

KONSTRUKSI SELECT

Select berguna untuk pembuatan layout berbentuk menu pilihan, sewaktu dijalankan bash akan menampilkan daftar menu yang diambil dari item list.

Sintak: ***select*** *varname* ***in*** *(item list)*

do

commands

done

2.5 Subrutin Atau Fungsi

Subrutin atau Fungsi

Merupakan bagian script atau program yang berisi kumpulan beberapa statement yang melaksanakan tugas tertentu. Dengan subrutin kode script tentunya akan lebih sederhana dan terstruktur, karena sekali fungsi telah dibuat dan berhasil maka akan dapat digunakan kapan saja bila diinginkan. Beberapa hal mengenai fungsi ini adalah:

- Memungkinkan penyusunan kode script ke dalam bentuk modul-modul kecil yang lebih efisien dengan tugasnya masing-masing.
- Mencegah penulisan kode yang berulang - ulang.

Di dalam shell fungsi juga bisa didefinisikan interaktif maupun secara script program, dan meskipun didefinisikan secara interaktif, sebuah fungsi juga bisa dipanggil melalui script yang dibuat dalam sebuah file dengan catatan fungsi tersebut sudah di export. Setelah melalui mekanisme export ini sub-shell juga bisa memanggil fungsi tersebut. Jadi fungsi adalah program yang dapat dipanggil oleh program lainnya dengan menggunakan

notasi NamaFungsi(). Fungsi memberikan exit status ($?) yang dinyatakan dengan *return nr*, atau nilai 0 sebagai default. Untuk membuat subrutin shell telah menyediakan keyword *function* seperti pada bahasa C, akan tetapi ini bersifat optional (artinya boleh digunakan boleh tidak). Bentuk umum dalam mendefinisikan fungsi dalam bash shell adalah sebagai berikut:

Sintak:

Nama_fungsi () { command; command; }

Function nama_fungsi { command; command; }

Function nama_fungsi () { command; command; }

nama_fungsi adalah pengenal (identifier) yang aturan penamaannya sama seperti pemberian nama variabel, setelah fungsi dideklarasikan atau dibuat dapat dipanggil dengan menyebutkan nama fungsinya. lebih jelasnya lihat contoh script berikut:

```
#!/bin/bash
function hai_hello() {
   echo "Hello, apa kbabar"
}

#panggil fungsi
hai_hello;
```

jika keyword *function* disertakan maka boleh tidak menggunakan tanda kurung (), tetapi jika keyword *function* tidak disertakan maka tanda kurung harus digunakan, lihat contoh berikut:

```
function hai_hello{
  echo "Hello,apa khabar"
      }

balas(){
  echo "Baik-baik saja";
  echo "Bagaimana dengan anda ?";
```

Mengirim argumen sebagai parameter ke fungsi

Tentunya suatu fungsi lebih berdaya guna apabila dapat menerima argumen yang dikirim oleh pemanggilnya dan memproses argumen tersebut didalam fungsinya, fungsi yang telah dibuat pada bash shell tentunya harus dapat melakukan hal tersebut, apabila pada pemanggilan fungsi disertakan argumen untuk diproses fungsi tersebut, maka bash akan menyimpan argumen - argumen tersebut pada parameter posisi 1,2,3,dan seterusnya..., dengan memanfaatkan parameter posisi tersebut tentunya dapat diambil nilai yang telah dikirim. lebih jelasnya lihat contoh berikut:

```
#!/bin/bash
function hello{
  if [ -z $1 ]; then
    echo "Hello, apa khabar anda"
  else
    echo "Hello $1, apa khabar";
  fi
}
#masukkan nama anda disini
echo -n "Nama anda :";
read nama

#panggil fungsi dan kirim isi variabel nama ke fungsi untuk dicetak
hello $nama;
```

perhatikan fungsi hello, sebelum mencetak pesan terlebih dahulu melakukan pemeriksaan dengan ***if*** terhadap parameter posisi $1 apabila kosong maka pesan *"Hello, apa khabar anda"* yang akan ditampilkan, tetapi jika ada string yang di inputkan maka string tersebut akan dicetak di dalam blok ***else*** pada fungsi. argumen pertama diteruskan ke variabel 1, argumen kedua pada variabel 2, dan seterusnya ... jika argumen yang dikirim lebih dari satu.

Cakupan Variabel

Secara default variabel - variabel yang digunakan dalam script adalah variabel bersifat global, maksud global adalah bahwa variabel tersebut dikenal dan dapat diakses oleh semua fungsi dalam script, tetapi bash menyediakan keyword *local* yang berfungsi membatasi cakupan (scope) suatu variabel agar dikenal hanya oleh fungsi yang mendeklarasikannya. Jadi variabel dapat didefinisikan dalam fungsi sebagai variable local atau global. Hal yang perlu diperhatikan, nama variable yang digunakan dalam sebuah fungsi, jangan sampai bentrok dengan nama variable yang sama di luar fungsi, sehingga tidak terjadi isi variable berubah. Perhatikan contoh berikut:

```
#!/bin/bash
proses(){
  echo "Isi variabel a=$a";
}
a=2;
proses();
proses $a
```

Jika ditambahkan *local a* pada fungsi proses menjadi

```
proses(){
 local a;
echo -e "a didalam fungsi, a=$a";
}
a=10;
proses()
echo "a diluar fungsi, a=$a"
proses $a
```

Pada contoh diatas jelas perbedaannya jika mendeklarasikan variabel memakai keyword *local* menyebabkan variabel tersebut hanya berlaku pada fungsi yang mendekalarasikannya. Pada contoh dalam fungsi proses variabel *a* dideklarasikan sebagai *variabel local* dan tidak diberi nilai.

2.6 Perintah Sed

Sed adalah editor teks noninteraktif untuk melakukan routine modifikasi file teks. Sed menyediakan kunci modifikasi untuk isi teks dalam sebuah file dan operasi pembacaan file forward. Sebagai contoh, sed dapat mendelete isi semua baris serta memberikan sebagai pattern teks, mengganti satu pattern dengan yang lainnya pada sebuah baris, membaca satu file ke dalam file lainnya dalam tempat yang ditentukan, atau menyebarkan bagian input file ke output file. Sed tidak dapat melakukan tugas menambah sebuah kolom dalam sebuah file, melakukan perubahan file yang rumit, atau menyimpan bagian dari sebuah file yang akan digunakan belakangan. Operasi rumit adalah tugas ideal untuk awk, yang mana sangat diperlukan dalam pemrograman sed. Sintak umum dari perintah sed seperti dibawah ini.

Syntak:

sed {expression} {file}

Sed mirip dengan *ed*, yaitu digunakan sebagai desain control untuk sebuah script. Perbedaanya sed untuk operasi internal, sedangkan ed untuk edit buffer setiap waktu sebuah perintah yang masuk. Sed mencari perintah script setiap saat pada baris baru yang dibaca dari input teks.

Sed merupakan operasi cyclically, yang mana sebuah cycle harus berisi :

1. membaca baris input kedalam pattern space
2. mengeksekusi edit script
3. men-copy pattern space untuk dikeluarkan/outputsed

Seperti halnya ed, sed juga dipergunakan untuk edit buffer dan menahannya sebagai teks edit. Perbedaanya sed edit buffer, memanggil pattern space yang berisi satu baris teks, sedangkan ed edit buffer berisi segala file. Meskipun demikian perintah sed fleksibel untuk baris pertama dalam pattern space. Konsep dari baris pertama yaitu dapat menahan isi dari baris pertama dan menjadikannya baris baru. Disamping pattern space sed juga berisi hold buffer. Beberapa perintah untuk pertukaran teks akhir dan ke empat antara pattern dan hold buffer dan opsi pada perintah sed adalah sebagai berikut:

Tabel 13-1. Opsi pada perintah sed

Option	Meaning
-n	Don't output the pattern space at the end of each cycle. When –n is given output is produced only when of the print commands is encountered
-e	*script* The –e argument specifies that the following argument is an the script
-f	*scriptfile* The –f argument specifies that the following argument is the name of a file that contains an editing script

Sed memberi supply dengan sebuah script, salah satunya adalah pada command line atau didalam sebuah file. Sed juga menggunakan baris alamat dan beberapa operator untuk mencetak, menghapus dan sebagainya. Penggunaan baris alamat pada sed (sed addresses line) dan beberapa operator yang digunakan seperti ditujukkan pada tabel 13-2 hingga 13-4. Perintah sed dapat menerima "nol", "satu", atau "dua" address. Nol address artinya melakukan perintah setiap baris/perbaris.

Tabel 13-2. Penggunaan Address pada sed

Address code	Meaning
n	An absolute line number n. The line counter is not reset each time a new file is processed. First line = 20 lines, line 21is the first line of the second file.
$	The last line of the input
/pat/	A context address matches any line containing the pat regular expression.

Tabel 13-3. Basic sed operators

Operator	Name	Effect
[address-range]/p	print	Print [specified address range]

[address-range]/d	delete	Delete [specified address range]
s/pattern1/pattern2/	substitute	Substitute pattern2 for first instance of pattern1 in a line
[address-range]/s/pattern1/pattern2/	substitute	Substitute pattern2 for first instance of pattern1 in a line, over *address-range*
[address-range]/y/pattern1/pattern2/	transform	replace any character in pattern1 with the corresponding character in pattern2, over *address-range* (equivalent of **tr**)
G	global	Operate on *every* pattern match within each matched line of input

Tabel 13-4. Examples of sed operators

Notation	Effect
8d	Delete 8th line of input.
/^$/d	Delete all blank lines.
1,/^$/d	Delete from beginning of input up to, and including first blank line.
/Jones/p	Print only lines containing "Jones" (with -n option).
s/Windows/Linux/	Substitute "Linux" for first instance of "Windows" found in each input line.
s/BSOD/stability/g	Substitute "stability" for every instance of "BSOD" found in each input line.
s/ *$//	Delete all spaces at the end of every line.
s/00*/0/g	Compress all consecutive sequences of zeroes into a single zero.
/GUI/d	Delete all lines containing "GUI".
s/GUI//g	Delete all instances of "GUI", leaving the remainder of each line intact.

Satu address adalah berarti melakukan perintah untuk semua baris yang sesuai dengan address. Dan dua address berarti melakukan perintah dalam range baris. Jika baris kedua kurang dari baris pertama, maka melakukan perintah hanya untuk baris pertama saja. Untuk mengeksekusi sebuah group dapat menggunakan perintah dengan tanda kurung kurawal *{ }*. Sintak satu atau dua address dinyatakan dengan *{ perintah, perintah pernyataan baris, dan akhir }* untuk memberi batas akhir dari sebuah group. Dibawah ini adalah contoh perintah pembacaan script dari sebuah terminal.

$ sed –n –f /dev/tty/ usr/dict/words

Modifikasi Teks

Perintah sed modifikasi teks dapat ditambahkan insert, atau delete line sebagai berikut:

- mengganti satu group line dengan lainnya
- mensubtitusi satu pattern teks dengan lainnya
- mentranslate satu group karakter kedalam group lainnya

contoh:

addr a\

text\

text Perintah penambahan tempat *text* pada output, sebelum baris input berikutnya dibaca.

addr1,addr2 c\

text\

text\ Perintah pengganti delete setiap address pattern space, output text, dan memulai cycle baru.

Addr1, addr2 d

Perintah delete untuk menghapus pattern space dan memulai

cycle baru.

Addr1, addr2 D

Perintah delete variant untuk menghapus inisial segmen pada pattern pace dam memulai cycle baru. D adalah ekuivalen dari d

jika pattern space berisi hanya satu baris.

Addr i\

text\

text\ Perintah insert untuk segera menempatkan text pada output.

Tetapi semua baris akhir menggunakan \ dan menuju baris baru.

addr1,addr2 s/expr/repl/f

Perintah subtitusi repl dengan expr pada semua baris address

addr1,addr2 y/string1/string2/

Perintah translate karakter pada string1 dan string2

Perintah penambahan dan insert hanya untuk satu address, sedangkan delete, mennganti baris, subtitusi pattern teks, dan translate untuk satu atau dua address. Contoh perintah sed script editing:

$ sed –f script remind

dimana : script adalah nama file script/program

remind adalah nama file yang berisi data yang akan diakses dengan perintah sed dengan opsi –f atau –e.

Control Flow

Control flow pada sed ada dua macam yaitu:

1. ***branch***

mengganti point untuk eksekusi didalam perintah script. Pernyatan branh dengan sebuah label atau untuk akhir dari script. Sed mempunyai unconditional branch yang merupakan/selalu branch dan conditional branch, dimana branch jika berupa bentuk subtitusi.

2. ***Control flow*** yang biasa digunakan untuk mengganti operasi cycle.

Contoh control flow untuk operasi cycle:

:label Membuat label dengan nama symbol untuk lokasi dalam script

addr1,addr2 b label

Branch pada label atau akhir dari script jika label tidak ada

addr1,addr2 n

Menulis pattern space pada output dan membaca baris input berikutnya.

addr1,addr2 N

Penambahan baris berikutnya pada input untuk pattern space.

Addr q

Keluar dari proses penulisan pattern space ke output, dan proses dihentikan.

addr1,addr2 t label

Branch pada label jika terdapat subtitusi yang dibuat dari akhir t atau sampai akhir dari baris input.

2.7 Graphical User Interface (GUI)

Zenity

Zenity mempunyai peran sebagai pembuat GUI pada shell programing atau yang disebut *Grafik User Interface* dimana mempunyai fungsi mempermudah user untuk mengoperasikan programnya melalui grafis secara interaktif. Zenity mempunyai banyak opsi seperti:

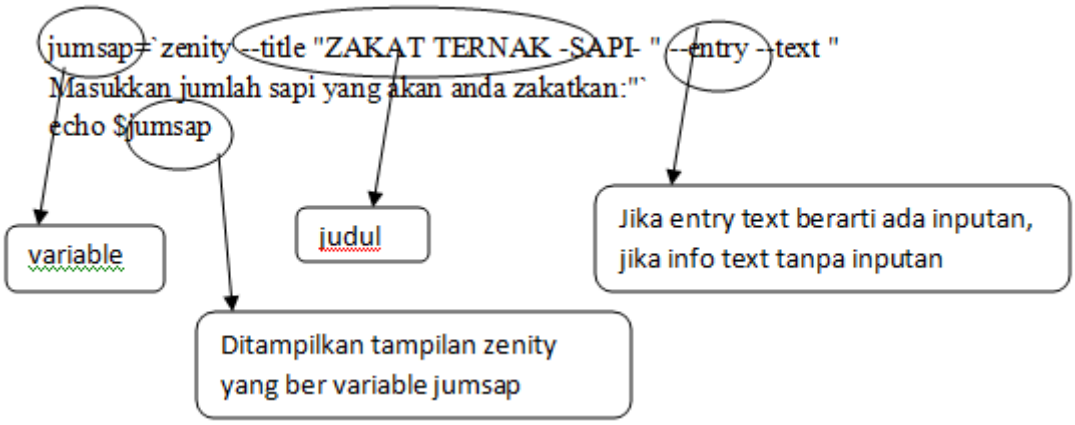
--entry : meminta input dari keyboard

--info : menampilkan text yang dimana berfungsi sebagai sebuah info

--list : membuat list berdasarkan column dan row secara tertabel dan lain-lain.

Dengan opsi seperti diatas zenity dapat digunakan untuk membuat sebuah *question dialog box*. Disamping itu zenity juga dapat digunakan untuk aplikasi yang lain seperti *calendar, entry, error, info, file selection, list, notification, progress, warning, scale* dan *text info*. Pada bab ini akan di ilustrasikan bagaimana membuat aplikasi zenity dialog.

Contoh sintak zenity :



Kalender Dialog

Dengan menggunakan option –-calendar dapat dibuat sebuah kalender dialog. Pada kalender dialog ini user diijinkan untuk memilih inisial yang khusus pada perintah date yaitu dengan menggunakan option :

--text=text
Spesikasi "text" digunakan untuk menampilkan teks pada kalender dialog.

--day=day

Spesifikasi "day" digunakan untuk memilih tanggal di dalam kalender dialog. Day harus diberi nilai angka untuk tanggal dari 1 sampai 31.

--month=month
Spesifikasi "month" digunakan untuk memilih bulan di dalam kalender dialog. Month harus diberi nilai untuk bulan dari 1 sampai 12.

--year=year
Spesifikasi "year" digunakan untuk memilih tahun di dalam kalender dialog.

--date-format=format
Menentukan kembali format dari dialog kalender setelah seleksi tanggal (date). Format standar tergantung pada system lokal dan format yang dapat diterima oleh fungsi ***strftime*** misalnya %A %d/%m/%y.

File Selection Dialog

Untuk membuat *file selection dialog* digunakan opsi *--file-selection*. *Zenity* akan melakukan seleksi file atau direktori ke output standard. Mode default *file selection dialog* adalah buka file. *File selection dialog* memiliki beberapa opsi:

--*filename*=filename

Menentukan file atau direktori yang dipilih pada dialog pemilihan file ketika dialog yang pertama ditampilkan.

--multiple

Menentukan pemilihan beberapa nama file dalam dialog pemilihan file.

--directory

Menentukan pilihan direktori pada dialog pemilihan file.

--save

Set dialog pemilihan file ke mode save.

--separator=*separator*

Menentukan string yang digunakan untuk membagi kembali daftar nama file.

List Dialog

Menggunakan opsi --*list* Untuk membuat *list dialog.* Zenity akan mengembalikan entry dalam kolom pertama baris teks yang dipilih ke output standar. Data untuk dialog kolom harus ditentukan menurut kolom, baris demi baris. Data dapat disediakan untuk dialog melalui input standar. Setiap entri harus dipisahkan oleh karakter baris baru. Jika menggunakan opsi --checklist atau --radiolist, setiap baris harus dimulai dengan ’TRUE’ atau ’FALSE’. List dialog memiliki opsi berikut:

--*column*=column

Menentukan header kolom yang ditampilkan dalam list dialog. Harus ditentukan opsi – colomn untuk setiap kolom yang ingin ditampilkan dalam dialog.

--*checklist*

Menentukan bahwa kolom pertama pada list dialog berisi kotak cek.

--*radiolist*

Menentukan bahwa kolom pertama pada list dialog berisi kotak radio.

--*editable*

Memungkinkan ditampilkan item yang akan diedit.

--*separator*=separator

Menentukan string apa yang digunakan ketika dialog mengembalikan daftar entry yang dipilih.

--print-column=*column*

Menentukan apakah kolom harus dicetak pada seleksi. Default kolom adalah '1 '. ‘ALL’ yang dapat digunakan untuk encetak semua kolom dalam list.

Untuk melihat penjelasan lebih detail tentang penggunaan zenity, dapat dilihat di terminal dengan mengetik ***man zenity,*** maka akan muncul petunjuk penggunaan zenity.

BAB 3

ISI

3.1 Listing Program

```
1. #!/bin/bash
2. ##########################################################################
3. #              PEMBUKAAN
4. ##########################################################################
5. declare -A field
6. tanggal=`date +%d/%m/20%y`
7. waktu=`date +%H:%M:%S`
8. clear
9.
10.   pembukaan() {
11.   zenity  --info --width="525" --height="14" \
12.   --title "Puzzle Number For You (3x3)" \
13.   --ok-label="Lanjut =>" \
14.   --text \
15.   "<span size=\"xx-large\">Informasi Permainan :</span>
16.   <span size=\"large\"> 1. Terdapat 9 buah angka yang acak
17.         2. Urutkan 9 angka tersebut dengan secepat-cepatnya
18.         3. Jika angka sudah urut, anda menang
19.         4. Dapatkan waktu tercepat anda</span>"
20.   if [ $? == 1 ]; then
21.         clear
22.         exit;
23.   else
24.         intro
25.   fi
26.   }
27.
28.
29.   intro() {
30.   while [[ (-z $name1) ]]; do
31.         pemain=$(zenity --entry --title="Kenalan Dong!!" \
32.               --text="Nama Kamu: " );
33.         maju=$?
34.   case $maju in
35.   1)
36.   exit 0;;
37.   0)
38.   pilih_level;;
39.   esac
40.   done
41.   }
```

```
42.  ##################################################################
43.  #           PROSES UTAMA
44.  ##################################################################
45.
46.  total_skor(){
47.  fmt -1 dataScore.txt | awk -F '\\s{3,} | , ' 'NR
     {for(i=1;i<=1;i++) {print $i}}' | zenity --list --width="560" --
     height="351" --column="Nama" --column="Perolehan waktu(detik)" --
     column="level" --column="Tanggal" --column="Jam"
48.  }
49.
50.
51.  main_lagi(){
52.  jwb=`zenity --entry --title "Puzzle Number For You (3x3)"  --
     width="300" --height="10"  --text  "Anda mau main lagi?? (y/t)"`
53.  case $jwb in
54.  y|Y)
55.  pilih_level
56.  ;;
57.  t|T)
58.  reset
59.  echo -e "  \033[33;45m--> Terimakasih telah mencoba permainan ini
     <--\033[0m"
60.  echo -e "           \033[43;31m--> #s0mprett0_tenan <--\033[0m"
61.  echo ""
62.  exit 0
63.  ;;
64.  0)
65.  reset
66.  esac
67.  }
68.
69.
70.  cekUrut(){
71.  if [[ ${field[1,1]} == 1 && ${field[1,2]} == 2 && ${field[1,3]}
     == 3 && ${field[2,1]} == 4 && ${field[2,2]} == 5 && ${field[2,3]} ==
     6 && ${field[3,1]} == 7 && ${field[3,2]} == 8 && ${field[3,3]} == 9
     ]];then
72.  end_time=$(date +%s)
73.  let total_time=end_time-start_time
74.  echo
75.  echo -e " \033[33;1;46mLooh kok sudah urut\033[0m (^_^)"
76.  echo -e " \033[36;1;44mSelamat Gan!!!!!!\033[0m"
77.  echo -e -n " \033[31;1;47m$pemain Menyelesaikan Permainan dalam
     waktu\033[0m "
78.      if [[ "$?" -eq 0 ]]; then
79.          date -u -d @${total_time} +%T
80.      else
81.          date -u -r ${total_time} +%T
82.      fi
83.
```

```
84.   echo "$pemain    $total_time      $lvl  $tanggal   $waktu" >>
   dataScore.txt
85.   echo -e " \033[36;1;44mLihat keseluruhan score?? (y/t)\033[0m"
86.   read skor
87.   case $skor in
88.      y|Y)
89.         total_skor
90.         main_lagi;;
91.
92.      t|T) main_lagi;;
93.   esac
94.   fi
95.   }
96.
97.
98.   move(){
99.   while true
100.  do
101.  read -d'' -s -n1 input  # read input
102.
103.  case $input in
104.     e) let var=field[1,3]
105.        let field[1,3]=field[2,3]
106.        let field[2,3]=var;;
107.
108.     w) let var=field[1,2]
109.        let field[1,2]=field[1,3]
110.        let field[1,3]=var;;
111.
112.     q) let var=field[1,1]
113.        let field[1,1]=field[1,2]
114.        let field[1,2]=var;;
115.
116.    a) let var=field[2,1]
117.        let field[2,1]=field[1,1]
118.        let field[1,1]=var;;
119.
120.    z) let var=field[3,1]
121.        let field[3,1]=field[2,1]
122.        let field[2,1]=var;;
123.
124.    x) let var=field[3,2]
125.        let field[3,2]=field[3,1]
126.        let field[3,1]=var;;
127.
128.    c) let var=field[3,3]
129.        let field[3,3]=field[3,2]
130.        let field[3,2]=var;;
131.
132.    d) let var=field[2,3]
133.        let field[2,3]=field[3,3]
```

```
134.       let field[3,3]=var;;
135.
136.    s) let var=field[2,2]
137.       let field[2,2]=field[2,3]
138.       let field[2,3]=var;;
139.  esac
140.
141.  clear
142.  echo -e "\033[33;1;46m                Informasi !!
      \033[0m"
143.  echo -e "\033[31;1;47mPerintah untuk memindahkan angka(sesuai
   posisi indeks) :\033[0m"
144.  echo -e "\n\033[33;41m Q W E \033[0m  \e[1mQ->\033[33;1;46m(tukar
   Q dan W)\033[0m  \e[1;31m|\033[0m  \e[1mW->\033[33;1;46m(tukar W dan
   E)\033[0m  \e[1;31m|\033[0m  \e[1mE->\033[33;1;46m(tukar E dan
   D)\033[0m\n\033[33;41m A S D \033[0m  \e[1mA->\033[33;1;46m(tukar A
   dan Q)\033[0m  \e[1;31m|\033[0m  \e[1mS->\033[33;1;46m(tukar S dan
   D)\033[0m  \e[1;31m|\033[0m  \e[1mD->\033[33;1;46m(tukar D dan
   C)\033[0m\n\033[33;41m Z X C \033[0m  \e[1mZ->\033[33;1;46m(tukar Z
   dan A)\033[0m  \e[1;31m|\033[0m  \e[1mX->\033[33;1;46m(tukar X dan
   Z)\033[0m  \e[1;31m|\033[0m  \e[1mC->\033[33;1;46m(tukar C dan
   X)\033[0m\n\n\n"
145.  printf '\n'
146.  for l in $(seq 3); do
147.  printf '\e[1;31m+------'
148.  done
149.  printf '\e[1;31m+\n'
150.  for b_new in $(seq 3); do
151.        printf '\e[1;31m|'
152.             for k_new in $(seq 3); do
153.             printf '\e[1;34m %4d \e[0m' ${field[$b_new,$k_new]}
154.             printf '\e[1;31m|'
155.                  done
156.        let b_new==4 || {
157.             printf '\n'
158.             for l in $(seq 3); do
159.          printf '\e[1;31m+------'
160.             done
161.             printf '\e[1;31m+\n'
162.        }
163.  done
164.  cekUrut
165.  done
166.  }
167.
168.  ##################################################################
      #                       CETAK AREA
169.  ##################################################################
170.  print_field(){
171.  clear
172.  declare -i start_time=$(date +%s)
```

```
173.  echo -e "\033[33;1;46m                Informasi !!
      \033[0m"; sleep 0.5
174.  echo -e "\033[31;1;47mPerintah untuk memindahkan angka(sesuai
   posisi indeks) :\033[0m"; sleep 0.5
175.  echo -e "\n\033[33;41m Q W E \033[0m  \e[1mQ->\033[33;1;46m(tukar
   Q dan W)\033[0m  \e[1;31m|\033[0m  \e[1mW->\033[33;1;46m(tukar W dan
   E)\033[0m  \e[1;31m|\033[0m  \e[1mE->\033[33;1;46m(tukar E dan
   D)\033[0m\n\033[33;41m A S D \033[0m  \e[1mA->\033[33;1;46m(tukar A
   dan Q)\033[0m  \e[1;31m|\033[0m  \e[1mS->\033[33;1;46m(tukar S dan
   D)\033[0m  \e[1;31m|\033[0m  \e[1mD->\033[33;1;46m(tukar D dan
   C)\033[0m\n\033[33;41m Z X C \033[0m  \e[1mZ->\033[33;1;46m(tukar Z
   dan A)\033[0m  \e[1;31m|\033[0m  \e[1mX->\033[33;1;46m(tukar X dan
   Z)\033[0m  \e[1;31m|\033[0m  \e[1mC->\033[33;1;46m(tukar C dan
   X)\033[0m\n\n\n"; sleep 0.5
176.  printf '\n'
177.  for l in $(seq 3); do
178.  printf '\e[1;31m+------'
179.  done
180.  printf '\e[1;31m+\n'
181.  for b in $(seq 3); do
182.        printf '\e[1;31m|'
183.              for k in $(seq 3); do
184.              printf '\e[1;34m %4d \e[0m' ${field[$b,$k]}
185.              printf '\e[1;31m|'
186.                    done
187.        let b==4 || {
188.              printf '\n'
189.              for l in $(seq 3); do
190.           printf '\e[1;31m+------'
191.              done
192.              printf '\e[1;31m+\n'
193.        }
194.  done
195.  move
196.  }
197.
198.  ##################################################################
199.  #           LEVEL PERMAINAN
200.  ##################################################################
201.  easy(){
202.  field[1,1]=2
203.  field[1,2]=3
204.  field[1,3]=6
205.  field[2,1]=1
206.  field[2,2]=5
207.  field[2,3]=9
208.  field[3,1]=4
209.  field[3,2]=7
210.  field[3,3]=8
211.  lvl="Easy"
212.  print_field
```

```
213.  }
214.
215.  medium(){
216.  field[1,1]=2
217.  field[1,2]=3
218.  field[1,3]=6
219.  field[2,1]=9
220.  field[2,2]=4
221.  field[2,3]=1
222.  field[3,1]=7
223.  field[3,2]=5
224.  field[3,3]=8
225.  lvl="Meduim"
226.  print_field
227.  }
228.
229.  hard(){
230.  field[1,1]=5
231.  field[1,2]=3
232.  field[1,3]=8
233.  field[2,1]=1
234.  field[2,2]=6
235.  field[2,3]=9
236.  field[3,1]=4
237.  field[3,2]=7
238.  field[3,3]=2
239.  lvl="Hard"
240.  print_field
241.  }
242.
243.  pilih_level() {
244.  level=$(zenity  --title "Sliding Number For You" --list  --list -
   -text "Pilih Level Permainan Gan" --radiolist  --column "Pick" --
   column "Level Anda" TRUE Easy FALSE Medium FALSE Hard); echo $level
245.  case $maju in
246.        0)
247.        (echo "50" ; echo "# Mulai Main!!!") |
248.        zenity --progress --no-cancel --pulsate --title "Puzzle
   Number For You" \ --text "Loading...!!" --percentage=0
249.        ;;
250.        1)
251.        echo "Terjadi Error!" ;;
252.        esac
253.  case $level in
254.  Easy)
255.  easy;;
256.  Medium)
257.  medium;;
258.  Hard)
259.  hard;;
260.  esac
```

```
261.  }
262.  ##################################################################
263.
264.  Pembukaan #(awal program dijalankan)
```

3.2 Analisa Program

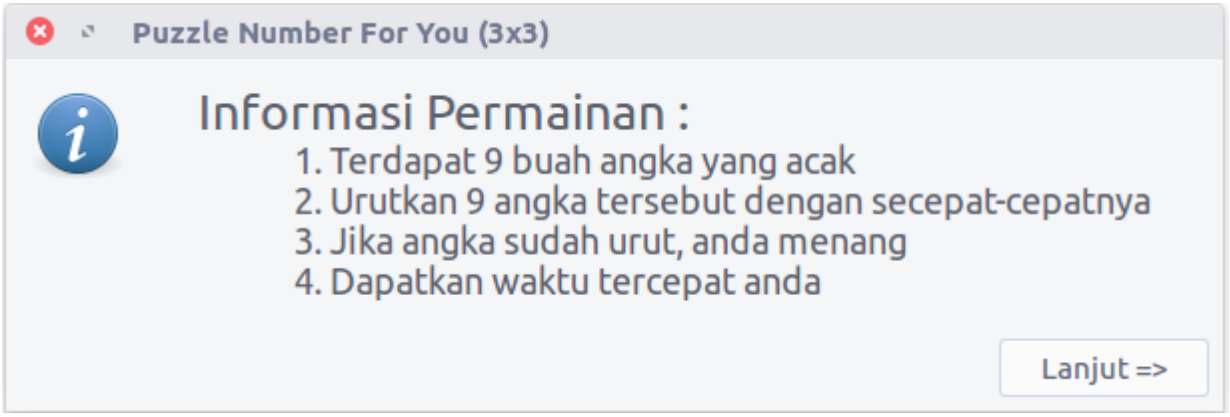
Pada program “Game urut Angka” terdapat beberapa source code yang digunakan dalam mewujudkan serta menciptakan program yang menarik dan interaktif. Beberapa yang harus dibutuhkan untuk menjalankan program ini dengan baik adalah membuat area permainan lewat kombinasi simbol yang ditampilkan diterminal serta menggunakan perintah yang ada di dalam GUI.

Program ini dimulai dari penggunaan interface berupa information dialog sebagai awal pembukanya. Zenity sendiri mempunyai peran sebagai pembuat GUI pada shell programing atau yang disebut Graphical User Interface dimana mempunyai fungsi untuk mempermudah user mengoperasikan programnya melalui grafis secara interaktif, komunikatif dan langsung bertatap muka dengan si pengguna. Zenity mempunyai banyak opsi seperti:

--entry : meminta input dari keyboard

--info : menampilkan text yang dimana berfungsi sebagai sebuah info

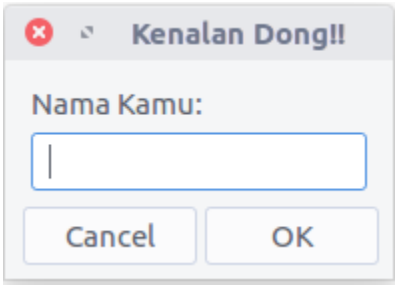
--text : membuat list berdasarkan column dan row secara tertabel dan lain-lain

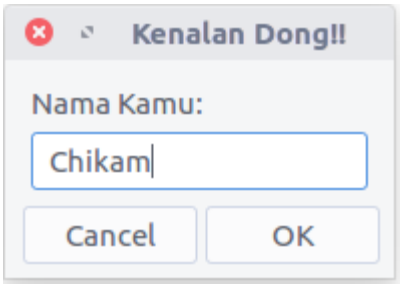


Pada cuplikan tampilan diatas merupakan tampilan awal dari program. Terdapat informasi tentang permainan yang disajikan. Berikut source code nya :

```
#!/bin/bash
#########################################################################
#                          PEMBUKAAN
#########################################################################
declare -A field
tanggal=`date +%d/%m/20%y`
waktu=`date +%H:%M:%S`
clear

pembukaan() {
zenity  --info --width="525" --height="14" \
--title "Puzzle Number For You (3x3)" \
--ok-label="Lanjut =>" \
--text \
"<span size=\"xx-large\">Informasi Permainan :</span>
<span size=\"large\">   1. Terdapat 9 buah angka yang acak
        2. Urutkan 9 angka tersebut dengan secepat-cepatnya
        3. Jika angka sudah urut, anda menang
        4. Dapatkan waktu tercepat anda</span>"
if [ $? == 1 ]; then
        clear
        exit;
else
        intro
fi
}
```

Dari source code diatas, ditampilkan sebuah zenity info dengan menampilkan bebrapa keterangan dari aturan game acak angka. Lalu ada pilihan “lanjut” yang mengarahkan kita pada tampilan berikut:

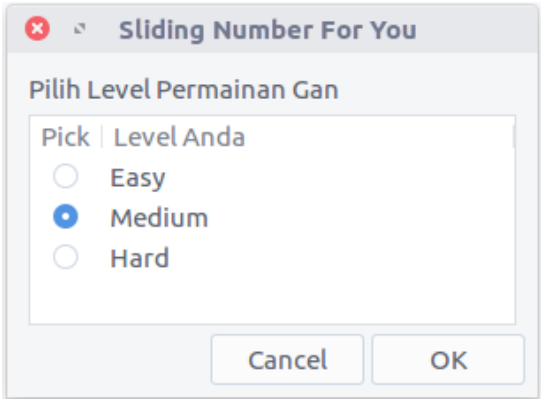


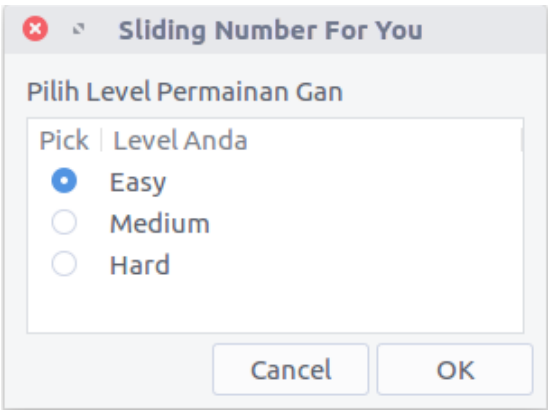


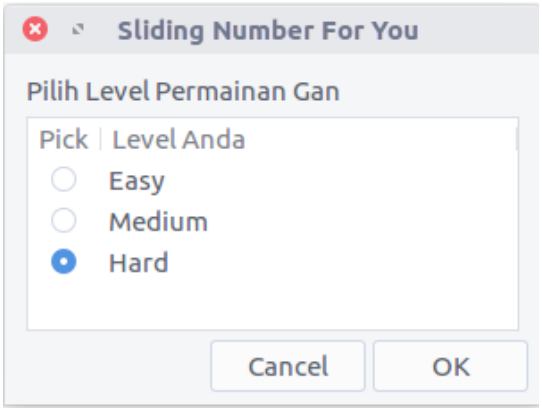
Berikut ini source code nya:

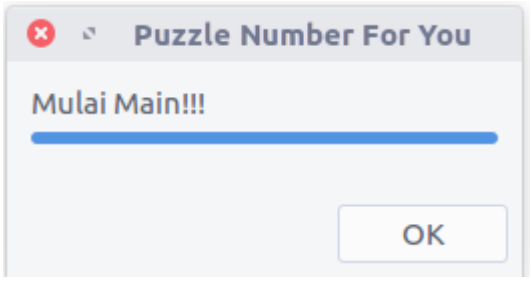
```
intro() {
while [[ (-z $name1) ]]; do
        pemain=$(zenity --entry --title="Kenalan Dong!!" \
                --text="Nama Kamu: " );
        maju=$?
case $maju in
1)
exit 0;;

0)
pilih_level;;
esac
done
}
```

Kita diberikan kesempatan untuk mengisi identitas berupa nama, jika telah kita inputkan nama selnjutnya kita akan masuk pada menu pilihan level permainan. Berikut ini tampilannya:









Dan source code nya adalah:

```
#########################################################################
#               LEVEL PERMAINAN
#########################################################################
easy(){
field[1,1]=2
field[1,2]=3
field[1,3]=6
field[2,1]=1
field[2,2]=5
field[2,3]=9
field[3,1]=4
field[3,2]=7
field[3,3]=8
lvl="Easy"
print_field
}

medium(){
field[1,1]=2
field[1,2]=3
field[1,3]=6
field[2,1]=9
field[2,2]=4
field[2,3]=1
field[3,1]=7
field[3,2]=5
field[3,3]=8
lvl="Meduim"
print_field
}
```

```
hard(){
field[1,1]=5
field[1,2]=3
field[1,3]=8
field[2,1]=1
field[2,2]=6
field[2,3]=9
field[3,1]=4
field[3,2]=7
field[3,3]=2
lvl="Hard"
print_field
}

pilih_level() {
level=$(zenity  --title "Sliding Number For You" --list  --list --
text "Pilih Level Permainan Gan" --radiolist  --column "Pick" --
column "Level Anda" TRUE Easy FALSE Medium FALSE Hard); echo $level
case $maju in
        0)
        (echo "50" ; echo "# Mulai Main!!!") |
        zenity --progress --no-cancel --pulsate --title "Puzzle
Number For You" \ --text "Loading...!!" --percentage=0
        ;;
        1)
        echo "Terjadi Error!" ;;
        esac
case $level in
Easy)
easy;;
Medium)
medium;;
Hard)
hard;;
esac
}
```

```
#########################################################################
#               CETAK AREA
#########################################################################
print_field(){
clear
declare -i start_time=$(date +%s)
echo -e "\033[33;1;46m
Informasi !!                        \033[0m"; sleep 0.5
echo -e "\033[31;1;47mPerintah untuk memindahkan angka(sesuai
posisi indeks) :\033[0m"; sleep 0.5
echo -e "\n\033[33;41m Q W E \033[0m  \e[1mQ->\033[33;1;46m(tukar Q
dan W)\033[0m  \e[1;31m|\033[0m  \e[1mW->\033[33;1;46m(tukar W dan
E)\033[0m  \e[1;31m|\033[0m  \e[1mE->\033[33;1;46m(tukar E dan D)
\033[0m\n\033[33;41m A S D \033[0m  \e[1mA->\033[33;1;46m(tukar A
dan Q)\033[0m  \e[1;31m|\033[0m  \e[1mS->\033[33;1;46m(tukar S dan
D)\033[0m  \e[1;31m|\033[0m  \e[1mD->\033[33;1;46m(tukar D dan C)
\033[0m\n\033[33;41m Z X C \033[0m  \e[1mZ->\033[33;1;46m(tukar Z
dan A)\033[0m  \e[1;31m|\033[0m  \e[1mX->\033[33;1;46m(tukar X dan
Z)\033[0m  \e[1;31m|\033[0m  \e[1mC->\033[33;1;46m(tukar C dan X)
\033[0m\n\n\n"; sleep 0.5
printf '\n'
for l in $(seq 3); do
printf '\e[1;31m+------'
done
printf '\e[1;31m+\n'
for b in $(seq 3); do
        printf '\e[1;31m|'
                for k in $(seq 3); do
                printf '\e[1;34m %4d \e[0m' ${field[$b,$k]}
                printf '\e[1;31m|'
                done
        let b==4 || {
        printf '\n'
        for l in $(seq 3); do
        printf '\e[1;31m+------'
        done
```

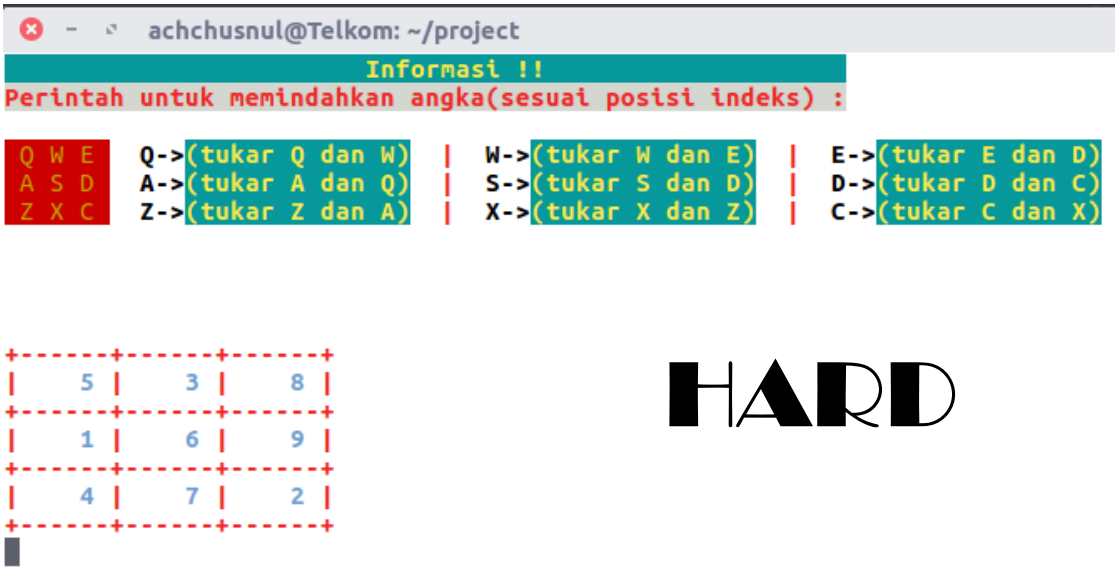
Terdapat tiga pilihan level permainan yaitu easy, medium, dan hard. Ketiganya mempunyai tingkat kesulitan yang berbeda beda karena pola tempat acak nya angka nya pun juga berbeda. Disaat yang bersamaan juga terdapat terdapat perhitungan waktu yang nantinya dibuat untuk perolehan waktu dalam pengurutan angka yang dideklarasikan dengan “ declare –i start_time=$(date +%s)”. Berikut ini pengacakan sesuai level permaianan:

```
                    Informasi !!
Perintah untuk memindahkan angka(sesuai posisi indeks) :

 Q W E   Q->(tukar Q dan W)  |  W->(tukar W dan E)  |  E->(tukar E dan D)
 A S D   A->(tukar A dan Q)  |  S->(tukar S dan D)  |  D->(tukar D dan C)
 Z X C   Z->(tukar Z dan A)  |  X->(tukar X dan Z)  |  C->(tukar C dan X)


+------+------+------+
|    2 |    3 |    6 |
+------+------+------+
|    1 |    5 |    9 |
+------+------+------+
|    4 |    7 |    8 |
+------+------+------+
```

EASY

```
achchusnul@Telkom: ~/project
                    Informasi !!
Perintah untuk memindahkan angka(sesuai posisi indeks) :

 Q W E   Q->(tukar Q dan W)  |  W->(tukar W dan E)  |  E->(tukar E dan D)
 A S D   A->(tukar A dan Q)  |  S->(tukar S dan D)  |  D->(tukar D dan C)
 Z X C   Z->(tukar Z dan A)  |  X->(tukar X dan Z)  |  C->(tukar C dan X)


+------+------+------+
|    1 |    6 |    2 |
+------+------+------+
|    9 |    4 |    8 |
+------+------+------+
|    7 |    5 |    3 |
+------+------+------+
```

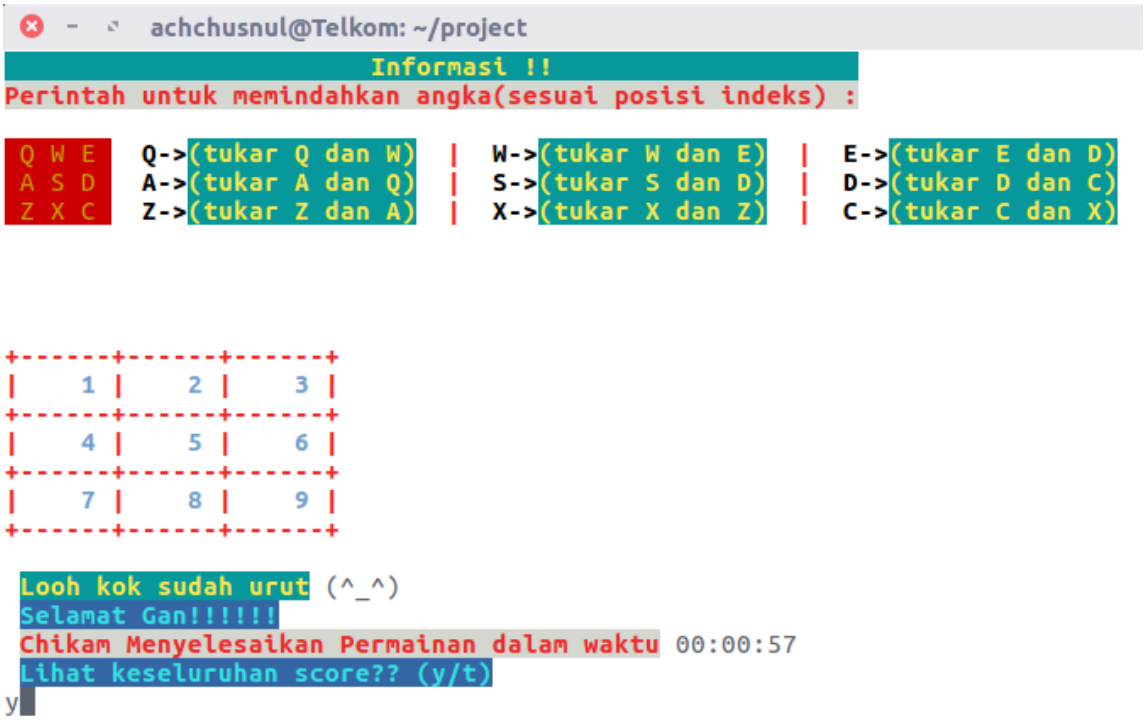
MEDIUM



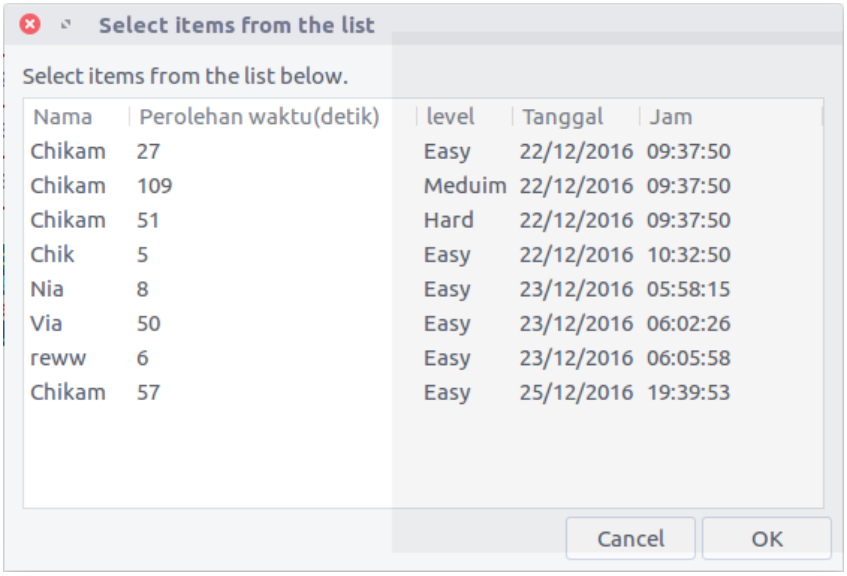
Didalam sebuah terminal tersebut terdapat petunjuk berupa tombol (Q W E A S D Z X C), yang mempunyai perintah menukarkan nilai sesuai indeks yang telah ditentukan. Disini kita harus memikirkan bagaimana caranya agar semua angka menjadi urut. Diproses ini , program yang berjalan adalah seperti berikut:

```
cekUrut(){
if [[ ${field[1,1]} == 1 && ${field[1,2]} == 2 && ${field[1,3]} ==
3 && ${field[2,1]} == 4 && ${field[2,2]} == 5 && ${field[2,3]} == 6
&& ${field[3,1]} == 7 && ${field[3,2]} == 8 && ${field[3,3]} == 9
]];then
end_time=$(date +%s)
let total_time=end_time-start_time
echo
echo -e " \033[33;1;46mLooh kok sudah urut\033[0m (^_^)"
echo -e " \033[36;1;44mSelamat Gan!!!!!!\033[0m"
echo -e -n " \033[31;1;47m$pemain Menyelesaikan Permainan dalam
waktu\033[0m "
  if [[ "$?" -eq 0 ]]; then
      date -u -d @${total_time} +%T
  else
      date -u -r ${total_time} +%T
  fi

echo "$pemain   $total_time     $lvl    $tanggal        $waktu" >>
dataScore.txt
echo -e " \033[36;1;44mLihat keseluruhan score?? (y/t)\033[0m"
read skor
case $skor in
   y|Y)
        total_skor
        main_lagi;;

   t|T) main_lagi;;
esac
fi
}
```

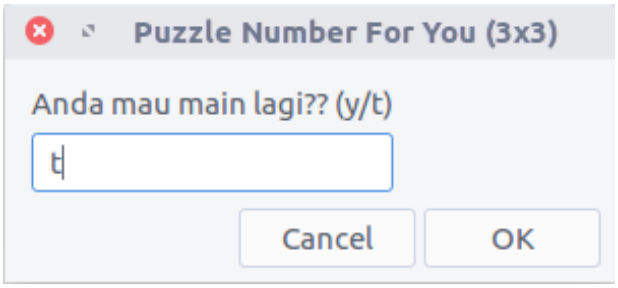
Saat kita menukarkan nilai, akan dicek posisi angka nya apakah sudah urut atau belum. Ini merupakan bagian dari fungsi cekUrut. Pengecekan dilakukan berulang –ulang hingga didapatkan angka yang urut dan mendapat hasil seperti:

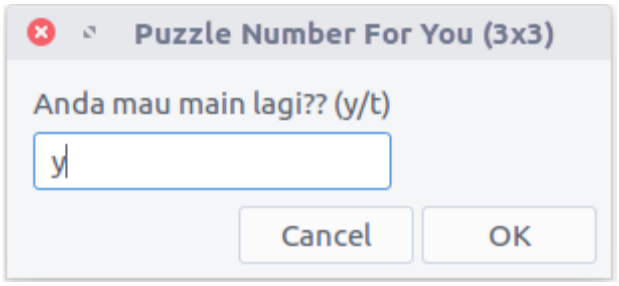


Disana ada pilihan untuk ditampilkan keseluruhan score, jika kita jawab "y" akan muncul database score yang seperti ini:

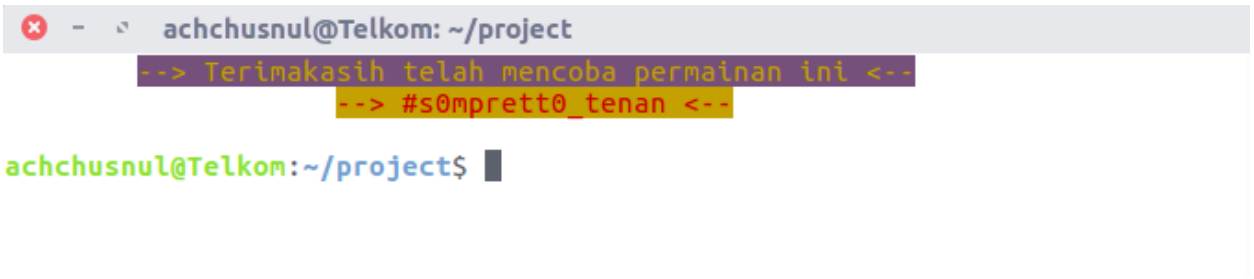


Jika kita menjawab "t" score tidak ditampilkan. Untuk proses selanjutnya akan ada pilihan untuk main lagi atau tidak seperti berikut:





Jika dijawab "y" akan disuguhkan pilihan level permainan lagi, dan jika pilihannya "t" maka program akan dihentikan.



BAB 4

PENUTUP

4.1 Kesimpulan

Pada program “Game Urut Angka” dapat disimpulkan bahwa dalam sebuah permainan terdapat aturan-aturan dan petunjuk yang harus dipahami agar kita dapat mencapai tujuan. Dalam hal ini adalah kemenangan yang ditandai dengan urutnya nomer yang telah diacak sebelumnya. Semakin cepat kita bisa mengurutkan nomer semakin bagus juga perolehan waktu kita. Kita harus mempunyai strategi dalam permainan ini karena jika ada salah pergerakan akan mengganggu kecepatan waktu yang didapat.

Program ini merupakan suatu implementasi dari logika berpikir kita yang kita tuangkan dalam bentuk algoritma dan selanjutnya dieksekusi dengan bahasa pemrograman dalam hal ini shell programming. Kombinasi antara terminal dan GUI membuat program ini lebih terlihat interaktif.

Program ini dirancang dengan menggunakan OS Linux bahasa pemrograman tingkat tinggi yang dikembangkan dengan sintak yang ada didalamnya. Tampilan dari program ini kami gunakan GUI, serta kombinasi simbol diterminal. Untuk keseluruhan program yang mengatur alurnya, kita gunakan perintah subrutin atau fungsi, supaya programnya lebih ringkas dan teratur.

4.2 Penutup

Pada program “Game Urut Angka” menampilkan kombinasi simbol dan perintah dasar diterminal sebagai papan angka yang dicetak dan visual dari Graphical User Interface (GUI) yaitu zenity yang memiliki berbagai model. Program ini dibuat untuk mengaplikasikan pengetahuan penulis tentang Shell programming. Ucapan terimakasih kepada dosen pengampu mata kuliah shell programming karena telah membimbing kami mengerjakan project akhir ini dan teman-teman saya yang telah membantu menyelesaikan program ini.

Penulis menyadari adanya kekurangan dalam pembuatan dan penulisan laporan project akhir shell programming. Penulis ingin menerima saran dan masukkan guna membangun kualitas pembuatan projek dan penulisan laporan berikutnya agar lebih baik lagi.

DAFTAR PUSTAKA

Modul teori dan praktikum *shell programming*