Buku Panduan

Pemrograman CLIPPER

Dwi Sakethi

Email: dwijim@maiser.unila.ac.id
URL: http://dwijim.tux.nu

Copyright © 2003 IlmuKomputer.Com

IlmuKomputer.Com

Bagian I

Pengantar

Bab 1

Sambutan

1.1 Kata Pengantar

Tidak ada pujian yang pantas kecuali hanya untuk Allah swt. sebagai Sang Penguasa dan Pencipta alam semesta. Sholawat dan salam semoga senantiasa dicurahkan kepada pembawa cahaya penerang, pemberi tauladan, pembawa kabar gembira, Rasulullah saw. beserta keluarganya, sahabatnya, pengikutnya, para *salafush sholeh*, dan kepada ummatnya hingga akhir zaman.

Buku ini merupakan buku pertama yang penulis susun dengan menggunakan perangkat lunak LaTeX. Oleh karenanya, dengan berbagai kendala, maka buku ini masih tampil apa adanya. Sesuai batas pengetahuan tentang LaTeXitu sendiri. Penggunaan perangkat LaTeXini didasari semangat penggunaan perangkat lunak yang legal disamping rasa penasaran seperti apa LaTeXitu. Juga karena kebosanan dengan perangkat lunak yang ada.

Materi buku ini disusun belum menggunakan teknik-teknik yang mempercepat eksekusi program. Barangkali dalam buku lanjutan, akan dibahas teknik-teknik untuk efisiensi waktu eksekusi program. Mengapa ? Karena buku ini diarahkan untuk mereka yang baru belajar pemrogram Clipper. Suatu bahasa pemrograman yang pada saat ini (tahun 2002), bisa dibilang sudah usang dan tradisional. Namun pada kenyataan, penulis masih menggunakannya untuk pengolahan data. Dan mungkin juga masih digunakan di tempat-tempat lain. Dalam kalimat 'pembelaan', penulis mengungkapkan bahwa untuk membunuh seekor nyamuk, tidak perlu menggunakan senjata AK-47. Cukup dengan sapu lidi. Sapu lidi itulah yang dimaksud dengan Clipper.

Penulis mengucapkan terima kasih yang kepada Bapak Warsono, Ph.D, yang telah memberikan 'atmosfir' yang kondusif untuk penulisan buku ini. Juga kepada reken-rekan di ADPC Puskom Unila beserta asisten (Didik, Jazuli, Yudi, Subian) yang untuk beberapa menjadi lahan percobaan buku ini. Tidak lupa juga kepada *Mbah Linux* Made Wiryana, karena dari tulisan LaTeX-nya penulis mengenal LaTeX. Begitu juga kepada komunitas Linux.

Satu hal lan yang cukup menarik dari buku ini (kalau penulis boleh ber-*gr*) adalah adalah indeks. Hal ini menarik, karena pada kebanyakan buku berbahasa Indonesia, Anda tidak menjumpai indeks (daftar kata-kata).

Dalam buku ini tentu saja banyak terdapat kekurangan maupun kesalahan. Selain karena sedikitnya pengetahuan yang ada pada penulis, juga karena buku

ini belum pernah mendapat kritik, saran dari siapa pun. Untuk itulah, sangat terbuka peluang untuk mengkritisi buku ini, dari sudut manapun juga.

Buku ini *Insyaa Allah* tersedia di http://dwijim.tux.nu. Siapa saja boleh men-*download*, memodifikasi, memperbanyak sebagian atau seluruh isi buku ini. Mudah-mudahan memberi manfaat seluas-luasnya kepada siapa saja yang memerlukannya. Hanya kepada Allah-lah kami berharap balasan dan ampunan-Nya.

dwi sakethi
dwijim@maiser.unila.ac.id
http://dwijim.tux.nu

Daftar Isi

Daftar Tabel

Daftar Gambar

Bagian II

Pengantar Clipper

Bab 2

Basis Data dengan Clipper

2.1 Pendahuluan

Clipper adalah bahasa pemrograman yang lebih ditujukan untuk pengelolaan data. Clipper terkenal pada dekade tahun 80-an sampai pertengahan 90-an. Clipper dimasukkan ke dalam kelompok keluarga Xbase, yaitu dBase, Clipper, Foxbase, FoxPro, dan lain-lain. Beberapa keuntungan Clipper di antaranya : tidak memerlukan perangkat keras yang tinggi (PC AT 286 pun bisa digunakan, dan ingat di pelosok mungkin masih ada komputer PC AT 386), berbasis teks (tidak grafis) sehingga cepat dalam pencetakan ke *printer* dan mudah dalam pemasukan data, kecepatan pemasukan data (tentu disenangi oleh operator pemasukan data). Selain itu juga ada kelemahan- kelemahannya, seperti : Clipper tidak menerapkan konsep basis data secara penuh (relasi antar file/entiti tidak didukung oleh manajemen Clipper, tapi harus dilakukan oleh *programmer*, misal tentang *referrential integrity*), tidak mendukung aplikasi internet (kalau aplikasi jaringan, bisa memakai Novell Netware), segala sesuatu dikerjakan oleh *programmer* (misal : tidak ada *generate code* secara otomatis oleh sistem).

Kemudian, dalam pemrogramannya bisa dikatakan struktur program dalam Clipper tidak beraturan, tidak seperti Pascal. Ini dapat dipandang sebagai kekurangan tapi dapat juga dimanfaatkan oleh pemrogram sehingga menjadi kelebihan ... :)

Bagi para pemakai Linux, sudah ada proyek untuk melakukan *porting* Clipper ke Linux. Di mana tempatnya ? Anda diyakini dapat mencari sendiri.

Pemilihan perangkat lunak tentu ditentukan oleh kebutuhan pemakai. Bagaimana kita memahami masalah kualitas ini ? Perangkat lunak yang berkualitas bukanlah perangkat lunak yang canggih dengan versi terbaru. Perangkat lunak yang berkualitas adalah perangkat lunak yang memenuhi dan sesuai kebutuhan pemakai. Oleh karenanya, untuk membunuh seekor nyamuk, tidak perlu memakai senjata AK-47, tapi cukup dengan sapu lidi ... :) Dan dalam satu contoh kasus, isi data base dalam perangkat lunak Oracle, justru kemudian diproses dengan Clipper.

Sebagai bahan studi kasus, pada materi Clipper ini, akan digunakan masalah PKAB. Namun dalam dokumen ini, tidak akan dijelaskan analisa dan perancangan program PKAB dimaksud. Jika Anda ingin mengetahui dokumentasinya,

Anda dapat membaca dokumentasi program PKAB.

2.2 Perangkat-Perangkat yang Dipakai

Clipper bukan merupakan perangkat lunak yang berdiri sendiri. Ada beberapa perangkat bantu (*tools*) yang dapat digunakan, yaitu :

- Sistem Manajemen Basis Data (*Data Base Management System*)

  Sistem Manajemen Basis Data (*Data Base Management System*) di sini, bukan dalam pengertian yang dipahami dalam konteks basis data. Sistem Manajemen Basis Data dalam Clipper menjalankan sebagian fungsi Sistem Manajemen Basis Data yang sebenarnya. Fungsi-fungsi yang dapat dijalankan, yaitu : pembuatan file data (dalam Clipper disebut data base file, yaitu file berekstensi .dbf, dan ingat data base di sini bukanlah pengertian data base dalam perangkat lunak Oracle, mySQL, PostGre SQL, dan lainnya, karena dalam Oracle misalnya .dbf ini disebut dengan tabel). Beberapa program yang dapat dipakai, misalnya : data base utility dari Clipper (dbu.exe), dBase (dbase.exe), Foxplus (mfoxplus.exe). Dengan beberapa pertimbangan, akan digunakan mfoxplus.exe.

- Penulisan Program (*Editor*)

  Untuk penulisan program bisa digunakan perangkat lunak apapun, yang penting dapat menulis dalam format ASCII.Jadi dapat dipakai perangkat lunak Editor Microsoft (edit.com), Microsoft Word, Notepad, Word Star (anak-anak muda mungkin tidak mengenal ini), Word Perfect. Namun disarankan untuk memakai perangkat QEdit (q.exe). Adapun alasannya adalah : QEdit kecil sehingga mudah dibawa-bawa, ada fasilitas blok kolom (copy, find & replace dalam kolom), bisa membuka banyak file, pindah ke baris tertentu dengan cepat, membuat logo dengan mudah, dapat digunakan untuk mencetak program dengan pemberian nomor baris. Orang tua yang terbiasa dengan Word Star, akan tidak asing dengan perintah-perintah dalam QEdit. Tulisan ini pun ditulis dengan perangkat QEdit.

- Kompilator (*Compiler*)

  Untuk mencek tata aturan penulisan, suatu program harus dikompilasi. Proses kompilasi dilakukan oleh kompilator (*compiler*), yang dalam Clipper diwakili oleh program clipper.exe. Jika terdapat kesalahan penulisan, Clipper akan memberi tahu nomor baris dan jenis kesalahannya (di sinilah kelebihan QEdit terpakai, yaitu pindah ke suatu nomor baris dengan cepat). Ada pun kesalahan logika tidak akan terdeteksi. Hasil dari proses kompilasi akan didapat file obyek, yaitu file dengan ekstensi .obj. Untuk melakukan proses kompilasi, terkadang perlu file- file lain, ini tergantung programnya. File-file lain yang diperlukan, biasanya tersimpan di *folder* include.

- Linker

  File obyek yang didapat dari proses kompilasi, untuk menjadi file yang bisa dijalankan (dieksekusi), harus di-link. Proses linking memakai program

Blinker (blinker.exe) atau program bawaan Clipper sendiri (rtlink.exe). Dalam proses linking, perlu file-file library (.lib) dan mungkin file obyek tambahan lain (.obj). File hasil linking berekstensi .exe yang berarti file tersebut dapat dijalankan. File .exe ini belum tentu bebas error.

- Pencari tulisan

  Untuk program-program yang besar apalagi jika sudah lama tidak, ada kemungkinan pemrogram atau pengelola sistem lupa letak suatu modul. Nah ... dengan adanya program pencari tulisan, kita dapat mencari letak suatu modul. Hasil pencarian, kita akan mendapatkan file dan nomor baris ditemukannya modul yang kita cari. Di sinilah kolaborasi antara QEdit dengan program pencari ini. Program pencari yang akan digunakan adalah ts.exe. Program ts.exe fungsinya hampir mirip dengan Find-File dalam Microsoft Office. Kelebihan ts.exe adalah adanya nomor baris yang akan memudahkan perbaikan program.

2.3 Manajemen Basis Data

Sebagaimana diungkapkan sebelumnya, manajemen basis data di sini dilakukan dengan mamakai perangkat lunak mfoxplus.exe. Dan Anda juga tentu sudah maklum bahwa manajemen basis data di sini tidak mencakup makna yang sesungguhnya. Proses yang dilakukan hanya sekedar membuat file basis data (.dbf) dengan *field-field* pelengkapnya (dalam basis data yang *beneran* disebut dengan tabel dan atribut).

2.3.1 Menjalankan Foxbase

Program Foxbase tersedia dalam versi banyak pemakai *multi user* dan satu pemakai *single user*. Untuk menjalankan program Foxbase ini, dari Dos Prompt (Start-Program-MS Dos Prompt), ketikkan mfoxplus[enter]. Namun sebaiknya Anda pindah dulu ke direktori tempat file data Anda, meskipun hal ini dapat dilakukan belakangan. Jika Anda berhasil, maka Anda akan mendapatkan tampilan seperti ini :

Untuk membuat file data (tabel), gunakan perintah *create nama-file*. Contoh *create ipa2002*. Untuk merekam, dari posisi *create* tadi, tekan tombol Ctrl-W. Dalam pengisian nama atribut (*field*), cukup ketikkan huruf pertama dari namanya.

Buatlah file data dengan nama dan atribut seperti berikut :

Field	Field Name	Type	Width	Dec
1	KODE	Character	6	
2	KETERANGAN	Character	37	

Tabel 2.1: Atribut Tabel PS.DBF

Field	Field Name	Type	Width	Dec
1	NO_PKAB	Character	5	
2	RESI	Character	1	
3	PILIHAN_1	Character	6	
4	PILIHAN_2	Character	6	
5	NAMA_SISWA	Character	30	
6	NO_INDUK	Character	10	
7	JURUSAN	Character	1	
8	KODE_SMA	Character	8	
9	NAMA_SMA	Character	35	
10	KELAMIN	Character	1	

Tabel 2.2: Atribut Tabel IPA2002.DBF

Field	Field Name	Type	Width	Dec
1	KODE_SMA	Character	8	
2	NEGE_SMA	Character	1	
3	NAMA_SMA	Character	35	
5	JALAN	Character	35	
6	KOTA	Character	25	

Tabel 2.3: Atribut Tabel SLTA.DBF

Untuk keluar dari Foxbase, ketikan *quit*. Program Foxbase tersedia dalam versi banyak pemakai *multi user*. Meskipun tujuan akhir adalah kemampuan membuat program dengan Clipper, namun perintah-perintah dasar pengelolaan basis data (dalam konteks Clipper) tetap diperlukan. Perintah-perintah dasar ini dikenal dengan sebutan itshape dot command, karena diawali oleh tanda '.'.

Perhatikan tanda '.' di atas tulisan Command. Perintah-perintah yang dijalankan di sini, bersifat interaktif. Sistem akan langsung merespon dan menampilkan hasilnya di layar.

2.3.2 Perintah create

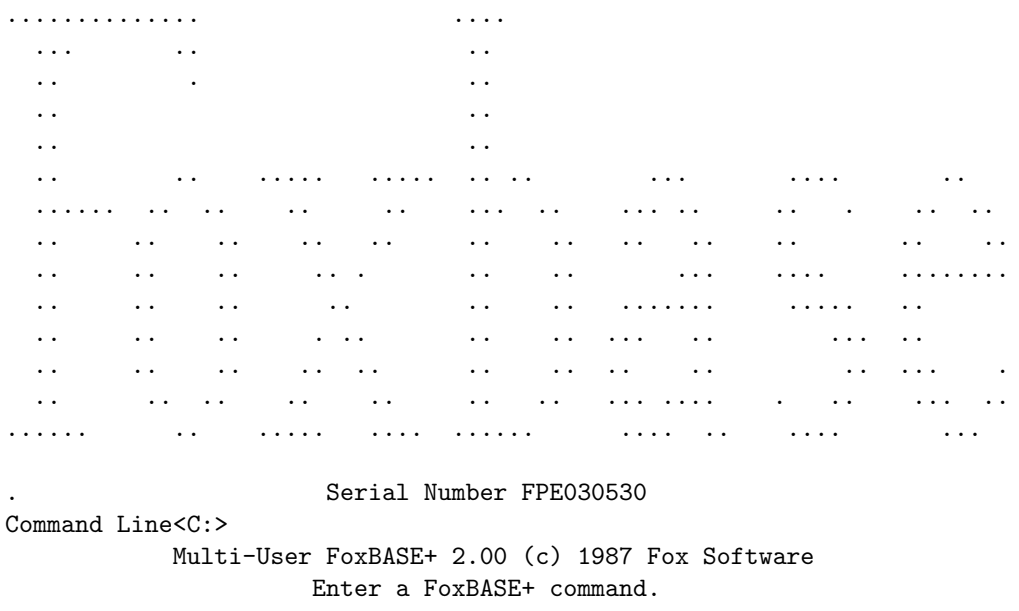
1. Fungsi : perintah **create** digunakan untuk membuat file data (tabel) beserta atribut-atributnya *field*.

2. Sintaks : **create** [nama-file]

3. Contoh : **create** ipa2002

4. Tampilan :

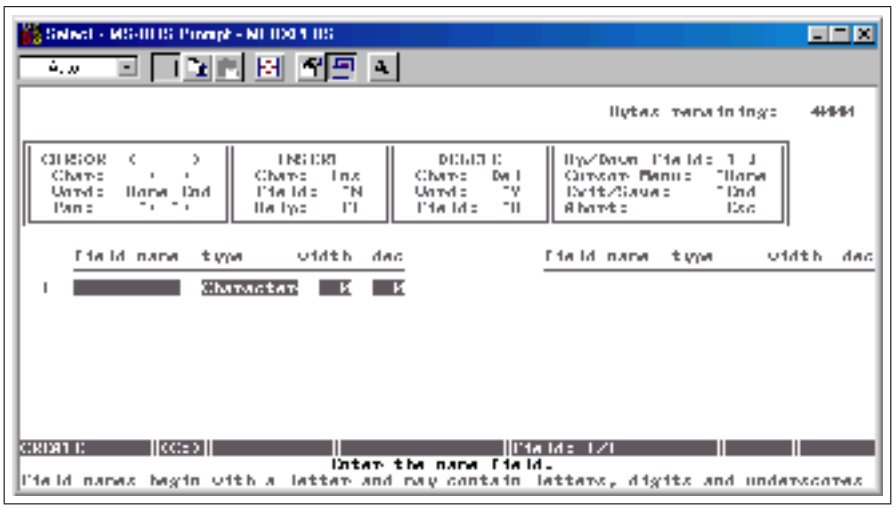
5. Tindak Lanjut : Isi nama field (*field nama*), tipe (*type*), ukuran *width*, dan jumlah angka desimal (*dec*).



Gambar 2.1: Tampilan FoxBase

```
.
Command Line<C:>
            Multi-User FoxBASE+ 2.00 (c) 1987 Fox Software
                      Enter a FoxBASE+ command.
```

Gambar 2.2: Tampilan Posisi *Dot Prompt*



Gambar 2.3: Perintah **create**

2.3.3 Perintah use

1. Fungsi : perintah **use** digunakan untuk membuka file data yang telah dibuat dengan perintah **create**. File data yang telah dibuat, untuk membukanya cukup dengan perintah **use**. Jika digunakan lagi perintah **create**, maka data akan hilang.

2. Sintaks : **use** nama_file_data

3. Contoh : **use** pkab

4. Tampilan :

5. Tindak Lanjut : Tergantung apa yang akan dilakukan, bisa menghapus data, melihat data, menghapus data, merubah atribut, dan sebagainya.

2.3.4 Perintah append

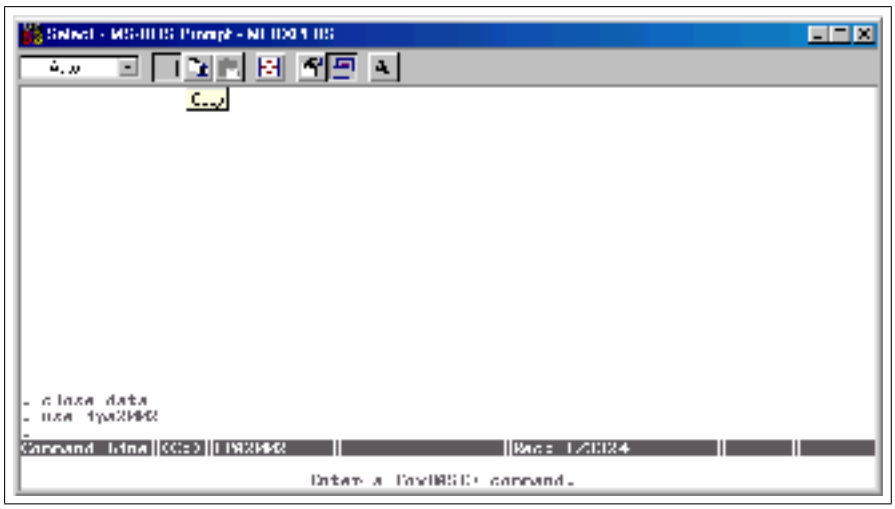
1. Fungsi : perintah **append** digunakan untuk menambah data atau *record* ke dalam file data (tabel).

2. Sintaks : **append**

3. Contoh : **append**

4. Tampilan :

5. Tindak Lanjut : Masukkan data-data sesuai dengan data-data yang ada.



Gambar 2.4: Perintah **use**

2.3.5 Perintah append blank

1. Fungsi : Perintah **append blank** digunakan untuk menambahkan data atau *record* kosong ke dalam file data (tabel).

2. Sintaks : **append blank**

3. Contoh : **append blank**

4. Tampilan :

5. Tindak Lanjut : -

2.3.6 Perintah edit

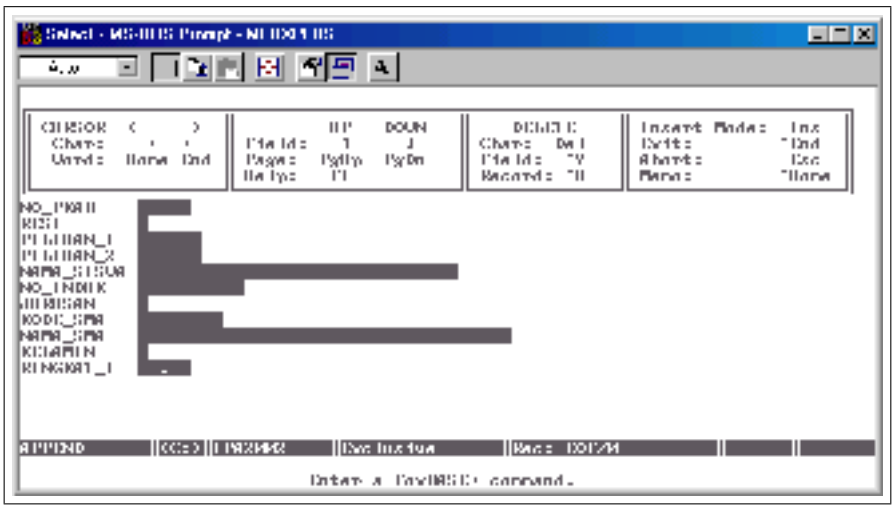
1. Fungsi : perintah **edit** digunakan untuk memperbaiki isi data atau *record* di dalam file data (tabel). Data yang diperbaiki sesuai dengan posisi *record* sekarang.

2. Sintaks : **edit**

3. Contoh : **edit**

4. Tampilan :

5. Tindak Lanjut : Gantilah isi data-data sesuai dengan data-data yang ada.



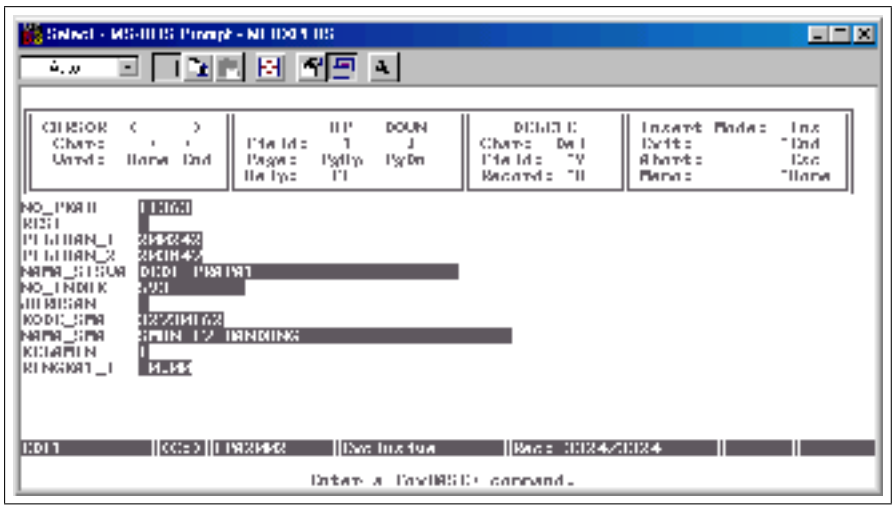
Gambar 2.5: Perintah **append**

2.3.7 Perintah browse

1. Fungsi : perintah **browse** merupakan perintah yang memiliki beberapa fungsi, yaitu : menampilkan data, merubah data, menghapus data.
2. Sintaks : **browse** [field, nama-nama *field*
3. Contoh : **browse field** NO_PKAB,NAMA_SISWA
   browse
4. Tampilan :
5. Tindak Lanjut :
   (a) Untuk menggantilah isi data-data, maka pada posisi kursor, data bisa langsung diganti.
   (b) Untuk menghapus data, pada posisi kursor, tekan tombol **del**, maka data akan dihapus.
   (c) Untuk menampilkan data, otomatis data sudah ditampilkan. Sedangkan untuk berpindah-pindah *record*, bisa menggunakan tombol panah, pg-up, pg-dn. Untuk pindah ke samping, jika *field* terlalu banyak, gunakan tombol Ctrl-panah.

2.3.8 Perintah go

1. Fungsi : Fungsi **go** digunakan untuk mengarahkan petunjuk atau *pointer* ke *record* tertentu di dalam file data (tabel).
2. Sintaks : **go** [nomor *record*]
3. Contoh : **go** 11



Gambar 2.6: Perintah **edit**

4. Tampilan : Perhatikan tulisan Rec:11/3324, yang artinya sekarang berada di *record* ke-11 dari 3324 *record*.

5. Tindak Lanjut : Sesuai keperluan, mau memperbaiki data, menambah data, dan sebagainya.

2.3.9 Perintah list

1. Fungsi : Fungsi **list** digunakan untuk menampilkan data.

2. Sintaks : **list** [nama **field**] [for kondisi]

3. Contoh : Perintah list ini sangat bervariasi, tergantung keperluan.

   (a) use ipa2002

   (b) list

   (c) list NO_PKAB, NAMA_SISWA, PILIHAN_1

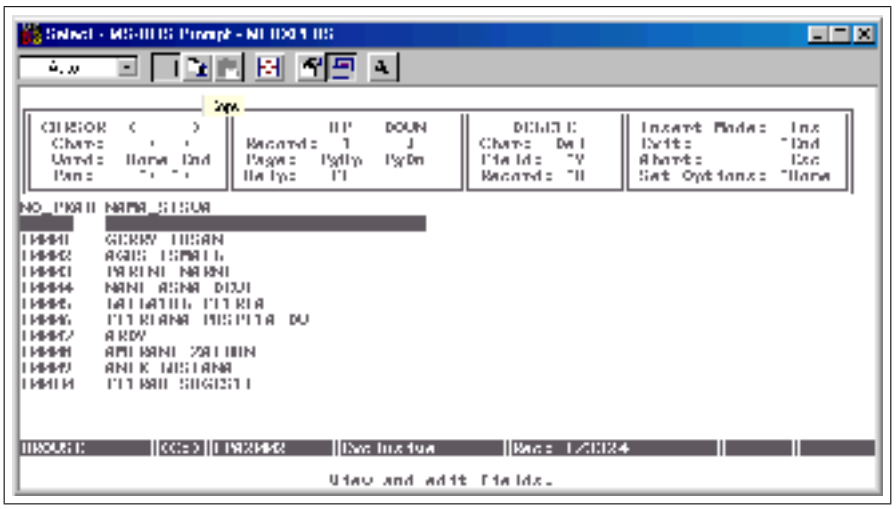
   (d) list NO_PKAB, NAMA_SISWA, PILIHAN_1 for PILIHAN_1='201647'

4. Tampilan : Perintah list yang terakhir artinya tampilkan Nomor PKAB, Nama Siswa, Pilihan 1 untuk peserta yang memilih program studi Matematika.

5. Tindak Lanjut : Sesuai keperluan, mau memperbaiki data, menambah data, dan sebagainya.

2.3.10 Perintah zap

1. Fungsi : Fungsi **zap** digunakan untuk menghapus seluruh data. Data akan terhapus tapi struktur data masih ada.



Gambar 2.7: Perintah **browse**

```
. go 11
.
Command Line<C:>IPA2002                    Rec: 11/3324

                              Enter a FoxBASE+ command.
```

Gambar 2.8: Perintah **go**

2. Sintaks : **zap**
3. Contoh : zap
4. Tampilan : -

2.3.11 Perintah append from

1. Fungsi : Fungsi **append from** digunakan untuk memasukkan isi satu file tabel ke tabel aktif (yang sedang di-**use**). File tabel yang akan dimasukkan diasumsikan sudah ada. Data yang dimasukkan sesuai nama atribut *field* masing-masing.
2. Sintaks : **append from** [file data]
3. Contoh :
   (a) use ipa2002
   (b) append from ipa2001
4. Tampilan : -

```
1541  11547   TANTRI RAHMAWATI              201647
1591  11599   HENI MARLENA                  201647
1641  11650   NADRA                         201647
1642  11651   MELATI CENDRA KASIH S         201647
1643  11652   ISRUNA HADIANI SIREGAR        201647
1656  11665   ALEX SOELISTYO                201647
1701  11710   SITA YULIANTARI               201647
1729  11738   MERRY ANGGRAENI               201647
1762  11771   DEWI AMARTHANI                201647
1764  11773   ROHANNA DUMA SARI S.          201647
1774  11783   INDAH WAHYUNI                 201647
1802  11811   LINDRA GULTOM                 201647
1813  11822   NURHASANAH PITRI HRP          201647
1825  11835   RAHMA NOVIYANI                201647
1830  11840   WINDA DERITA SINURAT          201647
1836  11846   OLIVIA SRI FEBRIANTY          201647
1905  11915   IKLILLAH HAYATI               201647
2088  11541   TUTI SURYATUL AZMI            201647
3293  10321   DWINA OKTAVIANI               201647
3305  11099   APRI YULIANI                  201647
```

Gambar 2.9: Tampilan Perintah **list**

2.3.12 Perintah index

1. Fungsi : Fungsi **index** digunakan untuk mengurutkan isi satu file tabel sesuai kebutuhan. Dengan adanya index, maka manajemen data menjadi lebih mudah. Index ini urutannya selalu dari kecil ke besar *ascending*. Oleh karena itu perlu menggunakan trik khusus, misalnya memberi tanda '-' (minus) pada kunci index yang bertipe numerik.
2. Sintaks : **index on** [atribut] to [nama file]
3. Contoh :
   (a) use ipa2002
   (b) list nama_siswa,skor
   (c) index on -SKOR to hasil
   (d) list nama_siswa,skor
4. Tampilan : sebelum dan setelah di-index

2.3.13 Perintah delete record

1. Fungsi : Fungsi **delete record** digunakan untuk menghapus record tertentu. Data akan terhapus tapi data masih ada selama belum ada perintah **pack**.

```
3305  APRI YULIANI                  813.370
3306  DEVITA ACHDALIA               734.460
3307  WIANA PUSPITASARI             767.220
3308  RESY APRILYA                  803.070
3309  DEWI SETIATI                  843.480
3310  NURMA YULITA                  793.460
3311  NOFRINACHAN FACHRI            795.570
3312  FAFIAN                          0.000
3313  DENI AWAL SETIAWAN            706.490
3314  YUNITA BUDIARTI               781.500
3315  NANDYA KARTIKA PUTRI          787.660
3316  YULIENI                       666.070
3317  EKO HERWINANDA                777.300
3318  MUSPIKA WATNA                 795.070
3319  SANDRA RUSDIANA               747.310
3320  ROY KARDO RAYMUN SITUMORANG   754.910
3321  ROZI DEFRIO                   786.000
3322  IDA RETNO NINGSIH             781.890
3323  NURLAILA                      852.870
3324  DEDI PRAPAT                   778.090

3139  HENDI EKA SETIAWAN            896.210
   9  AMIRANI ZAIBUN                897.060
1875  MUHAMMAD DAVID                897.660
3304  IIN NOVIANI                   899.010
3180  WAHYUNI RAJA GUK-GUK          899.110
 728  PANCA RAHAYU PRASETYANINGSIH  899.550
 143  LISA YUNITA                   900.190
 571  YENI ANGGRIANA                901.140
1943  VERDINA DEKAWATI              902.590
1921  LISA KARTIKA DEWI             904.630
 181  RIKI AFRIANDA                 905.000
1904  MAYATIKA                      906.800
 182  DINA SANTI LESTARI            907.140
 183  NELDA ARIANI                  908.650
 695  ISMONO                        908.750
 722  DIPHO MUHARDIAN               911.410
 265  SATIMAH MURNI                 913.310
 180  FAJRI SYAMSUL                 914.530
2868  FISSA NURJANAH                917.410
 603  JUWITA ASTUTI                 949.010
```

Gambar 2.10: Tampilan Perintah **Index**

```
JUMLAH NO_PKAB RESI PILIHAN_1 PILIHAN_2 NAMA_SISWA
  50   11046        201141    201744    ISKANDAR DINATA
  70   10118        201446    203847    ISKANDAR G FAHRI
  60   10860        201543    202047    MESY SEPRIDAWATI
  78   10441        201647    200845    MUHLISIN
  37   10669        201744    200845    INDRA HANAFI
  90   10189        201841    202144    ZAKI TANTOWI
 162   11047        201945    201446    EKO WAHYUDI
 146   11311        202047    204046    DENI AWAL SETIAWAN
  84   11015        202144    200741    DEDI KURNIAWAN
  64   21161        202241    203847    RAHMAWATI
 165   11392        202345    200346    SILVIA DIAN ANGGRAENI

BROWSE      <C:>TESTX      Exclusive      Rec: 11\39           Del
```

Gambar 2.11: Tampilan Data Setelah Dihapus

2. Sintaks : **delete record** [nomor_record]
3. Contoh : delete record 11
4. Tampilan : Perhatikan bahwa record ke 11 (dari 39 record) statusnya terhapus (ada tulisan Del).

2.3.14 Manajemen Basis Data dengan dbu.exe

Jika sistem operasi yang digunakan adalah Windows NT, maka perangkat lunak untuk manajemen basis data yang bisa digunakan adalah *database utiliy* dbu.exe dari Clipper.

Bagian III

Pemrograman Clipper

Bab 3

Program Dasar

3.1 Pemrosesan Program Clipper

Kepopuleran Clipper pada masa jayanya (tahun 1990-an), barangkali tidak dapat dilepaskan dari kesuksesan produk dBase III. Namun kelemahan dBase adalah selalu diperlukannya perangkat dBase itu sendiri untuk menjalankan program yang dibuat menggunakan dBase. Produk Clipper dapat menghasilkan program mandiri yang siap dieksekusi (executable file).

Proses pembuatan program Clipper (.prg) menjadi program yang siap dieksekusi (.exe) melalui beberapa tahapan, yaitu kompilasi dan linking. Hal tersebut bisa dilihat pada gambar berikut :

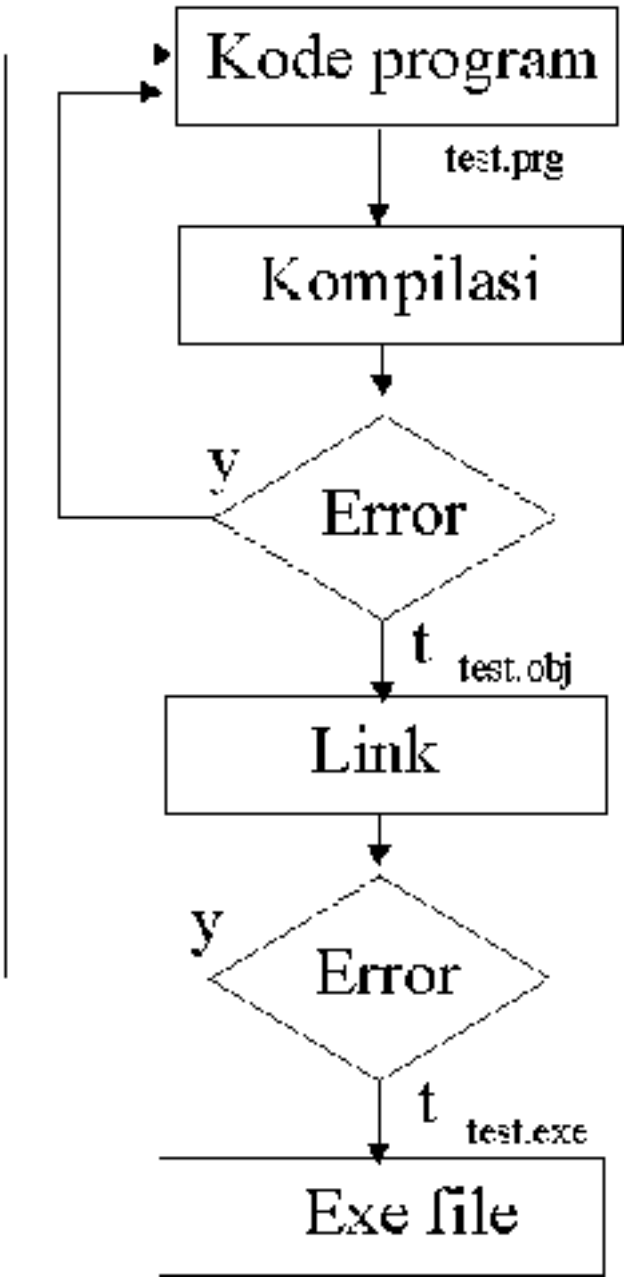
Proses kompilasi dilakukan untuk mencek sintaks penulisan program yang dibuat. Jika ada penulisan yang tidak benar, Clipper akan memberi tahu baris yang salah. Proses kompilasi tidak memberi tahu jika ada kesalahan lain seperti kesalahan logika, pembagian dengan nol, dan sebagainya. Jika tidak ada kesalahan akan dihasilkan file obyek (.obj). Jika masih ada kesalahan, maka program dibetulkan lagi dengan menggunakan teks editor (QEdit : q.exe). Sedangkan proses linking dilakukan untuk mencek kaitan antar program, keberadaan fungsi atau prosedur yang digunakan. Jika tidak ada kesalahan akan dihasilkan program yang bisa dijalankan dari DOS Prompt (.exe). Jika masih ada kesalahan pada program, maka program dibetulkan lagi dengan menggunakan teks editor (QEdit : q.exe). Program yang sudah menjadi .exe bukan berarti bebas dari kesalahan dan inilah keajaiban Clipper ... :)

Sebelum melakukan proses kompilasi dan linking ada variabel lingkungan (*environment variable*) yang harus diatur. Untuk mengatur variabel lingkungan ini, harus diketahui terlebih dahulu dimana program Clipper disimpan. Pengaturan ini dilakukan dari posisi DOS Prompt, jadi Anda harus memilih Start-Program-DOS Prompt. Sebagai contoh dalam kasus ini, program Clipper disimpan di partisi c: dengan subdirektori ca-62. Lebih jelasnya dapat dilihat pada gambar :

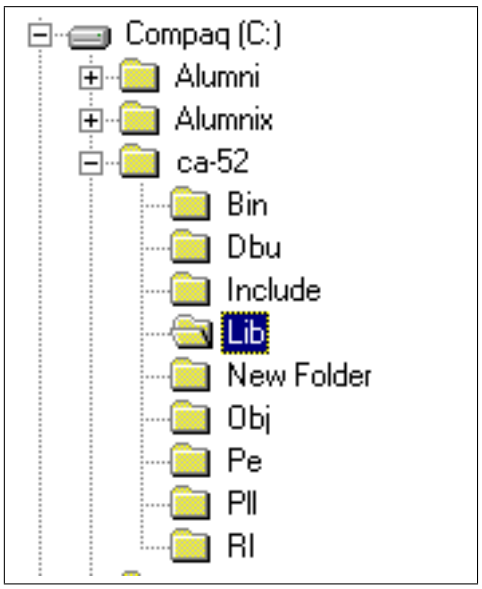
Variabel lingkungan tersebut ada tiga, yaitu :

1. include, untuk menentukan letak direktori file-file include (.ch). Pada contoh ini, maka ketikan $set\ include = c:\backslash ca-52\backslash include[enter]$.

2. obj, untuk menentukan letak direktori file-file obyek (.obj). Pada contoh ini, maka ketikan $set\ obj = c:\backslash ca-52\backslash obj[enter]$.



Gambar 3.1: Proses Program Clipper



Gambar 3.2: Letak Program Clipper

3. lib, untuk menentukan letak direktori file-file pustaka (.lib). Pada contoh ini, maka ketikan $set\ lib = c : \backslash ca - 52 \backslash lib[enter]$.

Nilai variabel lingkungan ini tentu saja bisa berbeda-beda tergantung Anda menyimpan program Clipper pada komputer Anda.

3.2 Proses Kompilasi

Sebagaimana telah disebutkan sebelumnya, proses kompilasi dilakukan untuk mencek sintaks penulisan program. Untuk program yang menggunakan perintah # include, maka variabel lingkungan **include** harus sudah diatur. Misalkan kita memiliki program seperti berikut :

```
clear
@ 10,30 say "Puskom
@ 11,30 say "ADPC "
@ 12,30 say "Unila"
@ 13, say "Lampung"
wait "tekan tombol apa saja !"
return
```

Kemudian program dikompilasi dengan perintah :

```
C:\dwi\clipper>\ca-52\bin\clipper test
```

Hasilnya seperti berikut :

```
Clipper (R) Version 5.2
Copyright (c) 1985-1993, Computer Associates International, Inc.
Microsoft C Floating Point Support Routines
Copyright (c) Microsoft Corp 1984-1987.  All Rights Reserved.
336K available
Compiling TEST.PRG
TEST.PRG(2)    Error C2007  Unterminated string: "Puskom"
TEST.PRG(6)    Error C2005  Statement not recognized,
match failed at: "<end of line>"
2 errors

No code generated
```

Terlihat bahwa masih ada kesalahan pada baris 2 dan baris 6. Pada baris 2, keterangannya `Unterminated string: "Puskom"`, yang artinya kurang petik penutup. Sedangkan pada baris 6, keterangannya `Error C2005 Statement not recognized, match failed at: "<end of line>"`, yang maksudnya kesalahan sintaks penulisan perintah say. Di dalam program tertulis `@ 13, say "Lampung"`, posisi kolom belum ada. Maka diperbaiki menjadi `@ 13,30 say "Lampung"`. Maka program lengkapnya, setelah diperbaiki menjadi :

```
clear
@ 10,30 say "Puskom"
@ 11,30 say "ADPC"
@ 12,30 say "Unila"
@ 13,30 say "Lampung"
wait "tekan tombol apa saja !"
return
```

Jika program tersebut kita kompilasi ulang, maka hasilnya seperti berikut :

```
C:\dwi\clipper>\ca-52\bin\clipper test
Clipper (R) Version 5.2
Copyright (c) 1985-1993, Computer Associates International, Inc.
Microsoft C Floating Point Support Routines
Copyright (c) Microsoft Corp 1984-1987.  All Rights Reserved.
336K available
Compiling TEST.PRG
Code size 144, Symbols 112, Constants 80
```

Karena sudah tidak ada keterangan error, maka proses dilanjutkan dengan proses linking. Maka kita berikan perintah seperti berikut :

```
C:\dwi\clipper>\ca-52\bin\blinker file test
```

Jika proses linking dilakukan dengan DOS Prompt dari Microsoft Windows, tidak akan ada keterangan apapun (meskipun mungkin ada kesalahan). Satu patokan saja, yaitu jika file .exe tidak dihasilkan, berarti masih ada kesalahan. Misal kita berikan perintah :

```
C:\dwi\clipper>test
Bad command or file  name
```

Ini artinya file test.exe belum jadi. Penyebabnya bisa karena masih ada kesalahan dalam program (misal menjalankan fungsi atau prosedur yang tidak ada, atau kita salah dalam memberikan perintah set lib). Jika sudah benar, kita akan mendapatkan file-file berikut :

```
Volume in drive C is COMPAQ
  Volume Serial Number is 1380-0FE3
   Directory of C:\backslash dwi\backslash clipper

   TEST     PRG           160  06-15-02  9:22p TEST.PRG
   TEST     OBJ           986  06-15-02  9:27p TEST.OBJ
   TEST     BAK           160  06-15-02  9:22p TEST.BAK
   TEST     EXE       178,012  06-15-02  9:27p TEST.EXE
   TEST     BIF           354  06-15-02  9:27p TEST.BIF
            5 file(s)        179,672 bytes
                     0 dir(s)        8,327.42 MB free
```

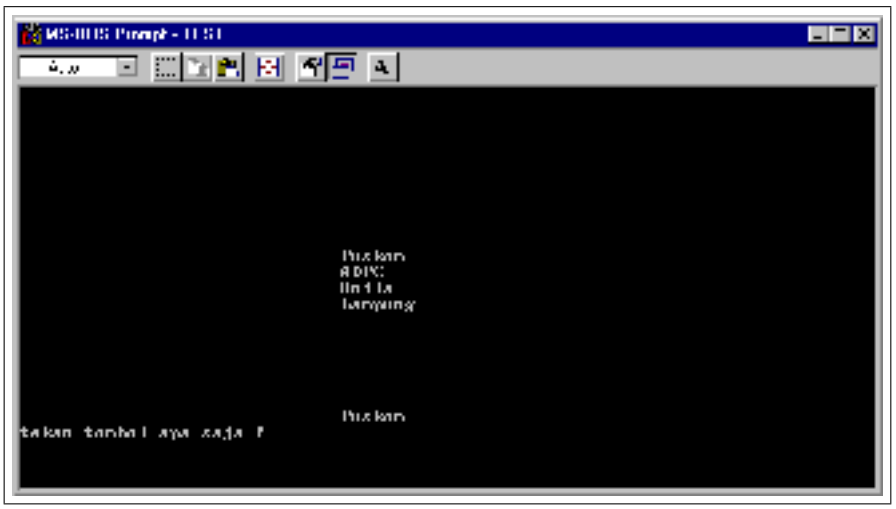
Sekarang kita bisa menjalankan program yang telah kita buat, dalam hal ini file **test.exe**. Untuk itu ketikkan saja : `test [enter]`. **Insyaa Allah** Anda akan mendapatkan hasil seperti ini :

3.3 First Clipper Program

Pada program pertama Clipper ini, kita akan mencoba untuk membuat program yang fungsinya sekedar menampilkan suatu tulisan dan memberhentikan proses sampai pemakai menekan sembarang tombol. Program mirip dengan conton program pada bagian sebelumnya. Perintah-perintah yang digunakan cuma ada 4 macam yaitu clear, say, wait, dan return.

3.3.1 Perintah clear

1. Fungsi : membersihkan layar tampilan
2. Sintaks : clear
3. Contoh : clear



Gambar 3.3: Hasil Program test.prg

3.3.2 Perintah say

1. Fungsi : mencetak sesuatu ke layar atau printer
2. Sintaks : @ baris,kolom say 'tulisan yang mau dicetak'
3. Contoh : @ 12,30 say 'dwi sakethi : dwijim@maiser.unila.ac.id'

3.3.3 Perintah wait

1. Fungsi : menghentikan proses sampai pemakai menekan sembarang tombol
2. Sintaks : wait [tulisan]
3. Contoh :
   (a) wait
   (b) wait 'tekan tombol apa saja !'
4. Keterangan : pada contoh pertama, program akan berhenti sementara dan di layar ditampilkan tulisan 'Press any key to continue', sedangkan pada contoh kedua, akan ditampilkan tulisan 'tekan tombol apa saja'.

3.3.4 Perintah return

1. Fungsi : mengembalikan arah program ke program sebelumnya, dalam hal ini karena program dijalankan dari DOS Prompt, maka program akan kembali ke DOS Prompt. Return ini merupakan akhir dari program, atau bisa juga akhir suatu fungsi atau prosedur.
2. Sintaks : return [hasil]
3. Contoh : return

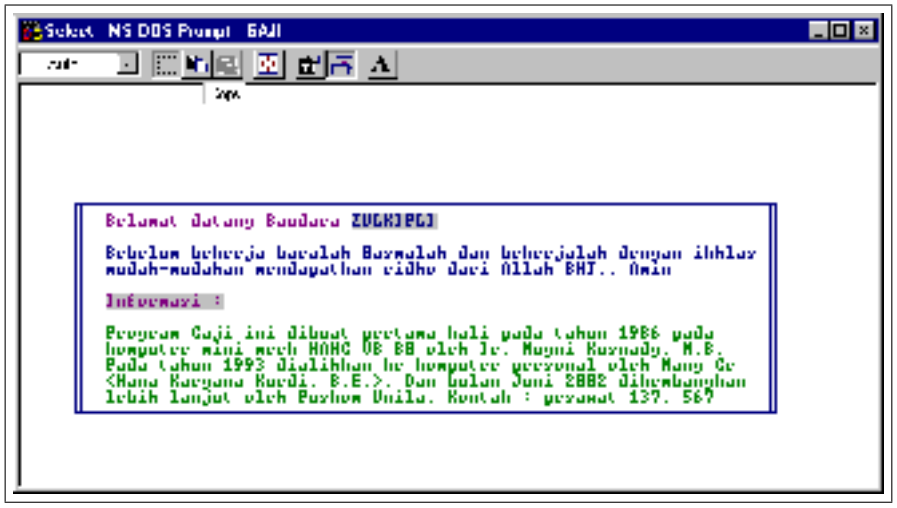
Berikut ini beberapa contoh program yang dapat dicoba dan dilihat hasilnya
:

```
* program test.prg

clear
@ 10,30 say 'Puskom'
@ 11,30 say 'ADPC  '
@ 12,30 say 'Unila '
@ 13,30 say 'Lampung'
@ 14,30 say 'Puskom'
@ 15,30 say 'Puskom'
@ 16,30 say 'Puskom'
@ 17,30 say 'Puskom'
@ 18,30 say 'Puskom'
@ 19,30 say 'Puskom'
@ 20,30 say 'Puskom'
wait "tekan tombol apa saja !"
return
```

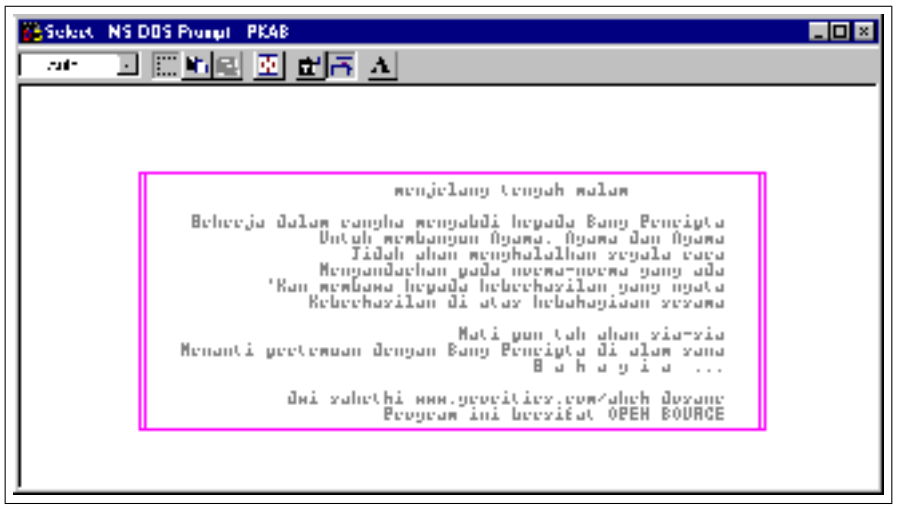
```
* program logo.prg
* ini sebenarnya tulisan unila gede-gede tapi karena karakternya
* berubah waktu di LaTEX, jadi ya begini kelihatannya
clear
@ 03,01 say ' +-------------------------------------------------------+'
@ 04,01 say ' |                                                       |'
@ 05,01 say ' |                          ii                           |'
@ 06,01 say ' |                          ii                           |'
@ 07,01 say ' |      uu    uu  nnn   n      l             aaa         |'
@ 08,01 say ' |      uu    uu  n nn  n  ii  l           aaaaa         |'
@ 09,01 say ' |      uu    uu  n  nn n  ii  l          aaa aaa        |'
@ 10,01 say ' |      uu    uu  n   nnn  ii  l         aaa   aaa       |'
@ 11,01 say ' |      uu    uu  n    nn  ii  l        aaaaaaaaaaa      |'
@ 12,01 say ' |      uuuuuuuu  n     n  ii  lllllll  aaa       aaa    |'
@ 13,01 say ' |      uuuuuuuu  n     n  ii  lllllll aaa         aaa   |'
@ 14,01 say ' |         uu                                            |'
@ 15,01 say ' |                                                       |'
@ 16,01 say ' +-------------------------------------------------------+'
set color to w*/n

@ 17,11 say 'tekan sembarang tombol '

wait ''

return
```

3.4 Tugas Latihan

Buatlah program (tentu saja dengan Clipper), yang hasilnya tampak seperti berikut :



Gambar 3.4: Hasil Program gaji.prg



Gambar 3.5: Hasil Program pkab.prg

Bab 4

Program Input Data dan Menu

4.1 Program Input Data

Suatu nilai dalam program Clipper bisa disimpan dalam variabel atau atribut (*field*. Jika disimpan pada variabel, artinya diletakkan di memory komputer, akibatnya jika komputer mati maka nilai variabel tersebut hilang. Sedangkan jika disimpan dalam atribut *field*, nilai tersebut akan disimpan secara permanen dalam disk berupa file atau tabel.

Variabel dalam Clipper memiliki 4 tipe, yaitu :

1. String, dipakai untuk menyimpan suatu nilai yang terdiri dari campuran karakter huruf, karakter angka, dan karakter lainnya. Tipe ini misal dipakai untuk menyimpan nama, alamat, nama mata kuliah, keterangan jenis kelamin, dan sebagainya. Contoh :

```
nama      = 'Hakan Sukur'
nama_kes  = 'Turki Banteng Bosporus'
alamat   := "Jl. Yang Benar No. 1 Gedong Meneng'
kosong   := space(11)
store 'sejahtera' to tujuan
```

2. Numeric, dipakai untuk menyimpan suatu nilai yang berupa angka. Angka di sini bisa berupa nilai pecahan, nilai bulat, nilai positif atau pun negatif. Dalam Clipper tidak dilakukan pembedaan jenis bilangan.

```
nomor     = 0
nilai     = 87.11
hasil    := -11
store -27 to untung
```

3. Logical, dipakai untuk menyimpan nilai yang hanya terdiri dari nilai .t. (*true*) dan nilai .f. (*false*). Misalnya untuk kondisi lulus tidak lulus, akhir file/tabel, dan sebagainya. Contoh :

```
lulus     = .t.
jawaban   = .f.
akhir    := .t.
store .f. to boleh
```

4. Date, dipakai untuk menyimpan nilai yang berupa tanggal. Untuk menyimpan nilai tanggal ini, bisa digunakan fungsi **date()** atau fungsi **ctod()**. Contoh :

```
tg_lhr    = date()   && tanggal sekarang
tg_lulus  = ctod('11-06-2002')
batas    := ctod('01-01-2003')
```

Pendefinisian suatu variabel dilakukan dengan memberi harga awal variabel. Misal variabel urutan ditentukan memiliki tipe numerik, maka cukup dengan memberikan perintah urutan = 0. Salah satu kelebihan (kalau tidak mau dikatakan sebagai kekurangan) Clipper adalah ketidakberaturan masalah variabel ini. Bagi yang pernah menggunakan bahasa Pascal atau C akan dapat merasakannya. Misal suatu variabel semula ditentukan bertipe numerik, dengan perintah :

```
store 1 to baris
```

Jika tiba-tiba di bagian lain program tipe baris diubah ke string (misalnya), hal itu tidak akan menyebabkan kesalahan program.

```
store 'ini aneh' to baris
```

Demikian juga dengan hasil suatu fungsi, suatu fungsi bisa memberikan hasil dengan tipe yang berbeda. Pada satu sisi, misal return .t. dan pada sisi lain return 1. Apakah ini keajaiban dunia yang kesembilan ?

4.2 Perintah yang Digunakan

Program input data yang akan dibuat belum menggunakan tabel (file data base). Program hanya menyimpan data di memory dan akan hilang setelah program kembali ke MS DOS Prompt. Perintah yang digunakan adalah perintah-perintah pada bagian sebelum ini yaitu : **clear, say, return**. Sedangkan perintah barunya adalah **get, read, store**.

4.2.1 Perintah get

1. Fungsi : memasukkan data melalui keyboard dan biasanya digabung dengan perintah **say**
2. Sintaks : @ baris,kolom get nama_variabel [picture] [valid]
3. Contoh :

```
@ 1,1  say 'Nama Anda  : '
@ 2,1  say 'Tgl. Lahir : '
@ 1,14 get nama picture '@!'  && huruf besar
@ 2,14 get tgl_lahir
read
```

4. Keterangan : program akan menunggu pemakai memasukkan nama dan tanggal lahir. Nama akan diubah langsung ke huruf besar karena ada perintah picture '@!'. Posisi baris dan kolom harus disesuaikan dengan letak baris dan kolom pada perintah **say**.

4.2.2 Perintah read

1. Fungsi : mengakhiri perintah pemasukkan data yang didahului dengan perintah **get**
2. Sintaks : read
3. Contoh :

```
@ 1,1  say 'Nama Anda  : '
@ 2,1  say 'Tgl. Lahir : '
@ 1,14 get nama picture '@!'  && huruf besar
@ 2,14 get tgl_lahir
read
```

4.2.3 Perintah store

1. Fungsi : memasukkan suatu nilai ke variabel
2. Sintaks : store [nilai] to [variabel]
3. Contoh :

```
nama      = 'Hakan Sukur'
nama_kes  = 'Turki Banteng Bosporus'
alamat   := "Jl. Yang Benar No. 1 Gedong Meneng'
kosong   := space(11)
store 'sejahtera' to tujuan
```

4. Keterangan : perintah **store** mempunyai fungsi yang sama dengan operator =.

4.2.4 Perintah set date italian

1. Fungsi : merubah format tanggal ke bentuk yang umum di Indonesia yaitu hari-bulan-tahun (dd-mm-yyyy).
2. Sintaks : set date [format-tanggal]

3. Contoh :

```
set date italian
```

4. Keterangan : Kondisi awal tahun, hanya disediakan sebanyak dua digit dalam pemasukan data. Meskipun demikian, sebenarnya di dalam tabel (*file data base*), tahun disimpan sebanyak 4 digit.

4.2.5 Perintah set century on

1. Fungsi : supaya dalam operasi tanggal, tahun ditampilkan dalam format 4 digit.
2. Sintaks : set century [on/off]
3. Contoh :

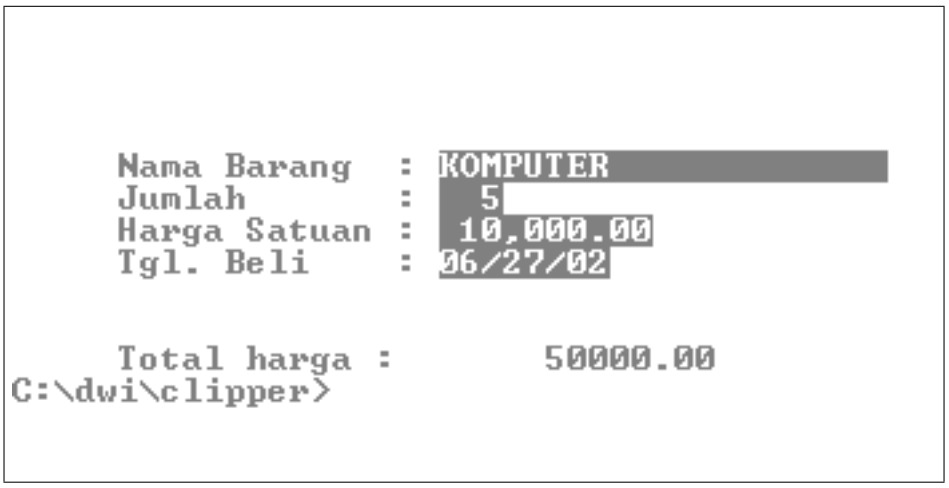
```
set century on
```

4.3 Contoh Program

Dengan perintah-perintah yang sudah diberikan, maka berikut ini ada contoh program untuk menghitung harga barang.

```
clear

store 0 to jumlah,harga
store space(21) to nama_barang
store date() to tgl_beli

@ 10,5 say 'Nama Barang  : '
@ 11,5 say 'Jumlah       : '
@ 12,5 say 'Harga Satuan : '
@ 13,5 say 'Tgl. Beli    : '

@ 10,20 get nama_barang picture '@!'
@ 11,20 get jumlah picture '999'
@ 12,20 get harga picture '999,999.99'
@ 13,20 get tgl_beli
read

bayar = jumlah * harga
@ 16,5 say 'Total harga : '
@ 16,20 say bayar

return
```

Hasil dari program tersebut tampak seperti berikut ini :



Gambar 4.1: Hasil Program harga.prg

4.4 Program Menu

Dari contoh dan latihan sebelumnya, maka kita sudah mempunyai beberapa file program Clipper. Selama ini, untuk menjalankan satu file yang ada, maka file .exe dari program tersebut kita jalankan. Menjalankan program-program tersebut akan menjadi lebih mudah jika program-program tersebut disatukan dalam satu menu program. Program yang sudah kita miliki yaitu :

```
C:\dwi\clipper>dir *.prg

 Volume in drive C is COMPAQ
  Volume Serial Number is 1380-0FE3
   Directory of C:\dwi\clipper

   TEST     PRG           160  06-15-02  9:22p TEST.PRG
   LOGO     PRG         1,126  06-14-02  2:58a LOGO.PRG
   NAMA     PRG           424  06-14-02  3:11a NAMA.PRG
   HARGA    PRG           442  06-27-02 10:26a HARGA.PRG
            4 file(s)          2,152 bytes
                     0 dir(s)       7,749.67 MB free
```

Program-program tersebut akan kita satukan dalam satu menu program.

4.4.1 Perintah set wrap on

1. Fungsi : membuat menu yang ada bisa berbalik arah, jika sudah sampai pada posisi paling bawah, kemudian ditekan panah ke bawan, menu pilihan akan berpindah ke atas.
2. Sintaks : set wrap [on/off]
3. Contoh :

```
set wrap on
```

4. Keterangan : Harga awal **set wrap off**, oleh karenanya kita harus menggantinya jika dalam program kita ingin agar menu bisa bergerak bebas. Dalam artian, jika penunjuk menu sudah berada pada posisi paling bawah, kemudian kita dan kolom menekan panah ke bawah, maka penunjuk menu akan pindah ke posisi paling atas. Demikian juga sebaliknya.

4.4.2 Perintah prompt

1. Fungsi : membuat menu tampilan menu yang bisa dipilih dengan menggerakkan anak panah atau huruf pertama dari tulisan di dalam menu.

2. Sintaks : @ baris,kolom prompt 'tulisan menunya'

3. Contoh :

```
@ 07,20 prompt 'a. Menu Sarapan Pagi'
@ 08,20 prompt 'b. Menu Makan Siang '
@ 09,20 prompt 'c. Menu Makan Malam '
@ 10,20 prompt 'x. Keluar           '
```

4. Keterangan : Untuk memilih menu, dapat menggunakan anak panah kemudian menggantinya tombol Enter, atau huruf pertama masing-masing menu (a, b, c, x). Untuk membuat menu yang menyamping, maka nomor baris harusa sama dan dan kolom kemudian nomor kolom yang berbeda. Kalau pada contoh di atas, nomor baris sama dab nomor kolom berbeda.

4.4.3 Perintah menu to

1. Fungsi : mengaitkan menu pilihan dengan suatu variabel.

2. Sintaks : menu to [nama-variabel]

3. Contoh :

```
@ 07,20 prompt 'a. Menu Sarapan Pagi'
@ 08,20 prompt 'b. Menu Makan Siang '
@ 09,20 prompt 'c. Menu Makan Malam '
@ 10,20 prompt 'x. Keluar           '
menu to makan
```

4. Keterangan : makan akan bernilai 1 jika pemakai memilih 'a. Menu Sarapan menggantinya Pagi', akan bernilai 2 jika pemakai memilih 'b. Menu Makan Siang', demikian seterusnya. Nilai makan inilah yang kemudian diseleksi dan dan kolom kemudian dijalankan nama program yang sesuai dengan pilihan yang yang dipilih oleh pemakai.

4.4.4 Perintah do

1. Fungsi : menjalan suatu program dari program lain, atau menjalankan suatu prosedur.
2. Sintaks : do [nama-program]
3. Contoh :

```
do logo
```

4. Keterangan : menjalankan program logo dari program lain. Hal ini berbeda dengan menjalankan program logo dari program logo itu sendiri ... :)

4.4.5 Perintah do while

1. Fungsi : membuat suatu bagian program dijalankan berulang-ulang sampai kondisi untuk berhenti terpenuhi.
2. Sintaks : do while [kondisi] ... enddo
3. Contoh :

```
do while .t.
   @ 07,20 prompt 'a. Menu Sarapan Pagi'
   @ 08,20 prompt 'b. Menu Makan Siang '
   @ 09,20 prompt 'c. Menu Makan Malam '
   @ 10,20 prompt 'x. Keluar           '
   menu to makan
enddo
```

4. Keterangan : bagian program yang menjalankan menu ini akan dijalankan berulang-ulang (*looping*) sampai terpenuhi kondisi untuk selesai. Kalau contoh di atas, belum ada perintah untuk berhenti. Perintah `do while` harus diakhiri dengan `enddo`.

4.4.6 Perintah do case

1. Fungsi : melakukan seleksi terhadap suatu nilai dan kemudian menentukan tindakan yang akan dilakukan.
2. Sintaks :

```
do case
   case [kondisi-1]
     \dots
   case [kondisi-2]
     \dots
endcase
```

3. Contoh :

```
do while .t.
   @ 07,20 prompt 'a. Menu Sarapan Pagi'
   @ 08,20 prompt 'b. Menu Makan Siang '
   @ 09,20 prompt 'c. Menu Makan Malam '
   @ 10,20 prompt 'x. Keluar           '
   menu to makan
   do case
      case makan=1
           do pagi
      case makan=2
           do siang
      case makan=3
           do malam
      case makan=4
           exit && keluar atau program selesai
   endcase
enddo
```

4. Keterangan : bagian program yang menjalankan menu ini akan dijalankan berulang-ulang (*looping*) sampai pemakai memilih menu 'x. Keluar'. Pada contoh di atas, maka dianggap ada program pagi.prg, siang.prg, dan malam.prg. Perintah `do case` harus diakhiri dengan `endcase`.

4.5 Contoh Program

Berdasarkan contoh-contoh sebelumnya, Anda dianggap sudah memilik program-program seperti berikut :

```
C:\dwi\clipper>dir *.prg

 Volume in drive C is COMPAQ
  Volume Serial Number is 1380-0FE3
   Directory of C:\dwi\clipper

   TEST     PRG           160  06-15-02  9:22p TEST.PRG
   LOGO     PRG         1,126  06-14-02  2:58a LOGO.PRG
   NAMA     PRG           424  06-14-02  3:11a NAMA.PRG
   HARGA    PRG           442  06-27-02 10:26a HARGA.PRG
            4 file(s)          2,152 bytes
                     0 dir(s)        7,749.67 MB free
```

Kita dapat membuat sebuah program yang menggabungkan keempar program tersebut. Contoh lengkap program program tersebut adalah sebagai berikut :

```
* nama program : gabung.prg
```

```
set century on
set date italian
set wrap on

do while .t.
   clear
   @ 07,15 prompt 'a. Logo Unila  '
   @ 08,15 prompt 'b. Entry Nama  '
   @ 09,15 prompt 'c. Harga Barang'
   @ 10,15 prompt 'x. Selesai     '
   menu to pilih
   do case
      case pilih=1
           do logo
      case pilih=2
           do nama
      case pilih=3
           do harga
      case pilih=4
           exit
   endcase
   wait ''
enddo
return
```

Jika program gabung.prg kita kompilasi, terlihat bahwa kompilasi dilakukan juga terhadap program-program yang dijalankan oleh program gabung.prg. Hal tersebut tampak seperti pada tampilan berikut :

```
C:\dwi\clipper>\ca-52\bin\clipper gabung
Clipper (R) Version 5.2
Copyright (c) 1985-1993, Computer Associates International, Inc.
Microsoft C Floating Point Support Routines
Copyright (c) Microsoft Corp 1984-1987.  All Rights Reserved.
331K available
Compiling GABUNG.PRG
Compiling LOGO.PRG
Compiling NAMA.PRG
Compiling HARGA.PRG
Code size 1335, Symbols 688, Constants 1305
```

Jika program gabung.exe kita jalankan, akan didapatkan hasil seperti gambar di samping. Untuk memilih menu, gunakan panah atas atau panah bawah, kemudian tekan tombol Enter. Dapat juga dengan menekan tombol huruf pertama masing-masing menu.

Gambar 4.2: Contoh Program Menu

4.6 Tugas Latihan

Sebagai bahan latihan dan Anda diharapkan untuk mencobanya.

1. Buatlah program untuk menghitung umur seseorang berdasarkan tanggal kelahiran.
2. Gabungkanlah program menghitung tanggal pada latihan sebelumnya ke dalam program gabungan.

Bab 5

Program Input Database

5.1 Pendahuluan

Pada pembahasan sebelumnya, pemasukan data hanya dilakukan di memory. Akibatnya jika komputer dimatikan data pun akan hilang. Pada bahasan berikut ini, akan dilakukan pemasukkan data yang kemudian direkam ke suatu tabel (file data). Dengan adanya perekaman data ini, data tidak akan hilang meskipun komputer dimatikan. Dan memang kondisi seperti inilah yang sebenarnya dilakukan.

Kita merancang untuk membuat suatu menu yang terdiri atas beberapa menu seperti berikut :

1. Pemasukan/Editing Data, digunakan untuk memasukkan data baru atau mengedit data yang sudah ada.
2. Penghapusan Data, digunakan untuk menghapus data yang tidak digunakan atau data yang salah sehingga tidak akan dipakai.
3. Lihat Data, digunakan untuk melihat data-data yang sudah ada di dalam tabel/file data.
4. Selesai

Program untuk pemasukkan data ini sangat bervariasi tergantung perancangannya. Oleh karenanya, boleh jadi kita akan menemukan perbedaan-perbedaan di antara beberapa program. Dalam program pemasukkan data ini, kita akan membuat variabel memory yang berkaitan dengan atribut (*field*). Untuk memudahkannya maka dibuat nama variabel memory sama dengan huruf m ditambah nama atribut (*field*). Contoh : Ini perlu dilakukan karena dalam programnya nanti, kita akan mengganti atribut *field* dengan variabel memory. Contohnya seperti berikut :

```
replace NO_PKAB    with mNO_PKAB
replace PILIHAN_1  with mPILIHAN_1
replace PILIHAN_2  with mPILIHAN_2
replace NAMA_SISWA with mNAMA_SISWA
replace NO_INDUK   with mNO_INDUK
replace JURUSAN    with mJURUSAN
```

No.	Field	Variabel
1	NO_PKAB	mNO_PKAB
2	RESI	mRESI
3	PILIHAN_1	mPILIHAN_1
4	PILIHAN_2	mPILIHAN_2
5	NAMA_SISWA	mNAMA_SISWA
6	NO_INDUK	mNO_INDUK
7	JURUSAN	mJURUSAN
8	KODE_SMA	mKODE_SMA
9	NAMA_SMA	mNAMA_SMA
10	KELAMIN	mKELAMIN

Tabel 5.1: Nama Atribut dan Variabel Memory-nya

```
replace KODE_SMA   with mKODE_SMA
replace KELAMIN    with mKELAMIN
```

5.2 *English Structured*

Kita akan mencoba menggunakan sebuah *tools* yang disebut *English Structured*. *English Structured* adalah ungkapan dalam keseharian yang menunjukkan urutan proses, perhitungan, penyeleksian kondisi terhadap suatu masalah.

1. Sekumpulan blok pernyataan. Contoh :

   ```
   Hitung total
   Set pajak penjualan sama dengan total dikalikan
       bobot pajak penjualan
   Set total sama dengan total ditambah pajak penjualan
   Hitung diskon
   Set total bersih sama dengan total dikurangi diskon
   Cetak total bersih di faktur
   ```

2. Struktur keputusan. Contoh :

   ```
   IF (total lebih besar dari 500000) maka
      Set diskon sama dengan total dikalikan 15%
   jika tidak
      Set diskon sama dengan nol
   ```

3. Struktur perulangan. Contoh :

   ```
   DO WHILE (condition)
      do block

   REPEAT do block UNTIL (condition)
   ```

English Structured untuk program pemasukan data ini kita buat seperti berikut :

1. Program Menu :

```
Set wrap menjadi on
Hapus layar
Kerjakan selama benar
  Tampilkan menu Pemasukan/Editing Data
  Tampilkan menu Penghapusan Data
  Tampilkan menu Browsing Data
  Tampilkan menu Selesai
  Rekam gambar tampilan
  Hapus layar
  Jika pilihan sama dg. 1 jalankan program input data
  Jika pilihan sama dg. 2 jalankan program hapus data
  Jika pilihan sama dg. 3 jalankan program browsing data
  Jika pilihan sama dg. 4 keluar dari program
  Kembalikan gambar tampilan
Selesai
```

2. Program Input Data :

```
Buka file data
Buat indeks sesuai atribut kunci
Beri harga awal variabel kunci indeks
Kerjakan selama benar
  Tampilkan tulisan untuk input/edit data
  Input variabel memory yang menjadi kunci indeks
  Jika menekan tombol ESC maka keluar
  Cari variabel kunci indeks
  Jika ditemukan maka
     Masukkan atribut/field ke variabel memory
   jika tidak
     Isi variabel memory sesuai atribut/field
  Input variabel memory selain kunci indeks
  Jika tidak menekan ESC maka
    Cari variabel kunci indeks
    Jika tidak ketemu
       Tambahkan record kosong
    Ganti field dengan variabelnya
Tutup file data dan index
Selesai
```

3. Program Hapus Data :

```
Buka file data
Buat indeks sesuai atribut kunci
Beri harga awal variabel kunci indeks
Kerjakan selama benar
```

```
  Tampilkan tulisan untuk input/edit data
  Input variabel memory yang menjadi kunci indeks
  Jika menekan tombol ESC maka keluar
  Cari variabel kunci indeks
  Jika ditemukan maka
     Tampilkan atribut/field
     Input pertanyaan mau dihapus
     Jika dihapus maka
       Hapus data tersebut
Tutup file data dan index
Selesai
```

4. Program Brow Data :

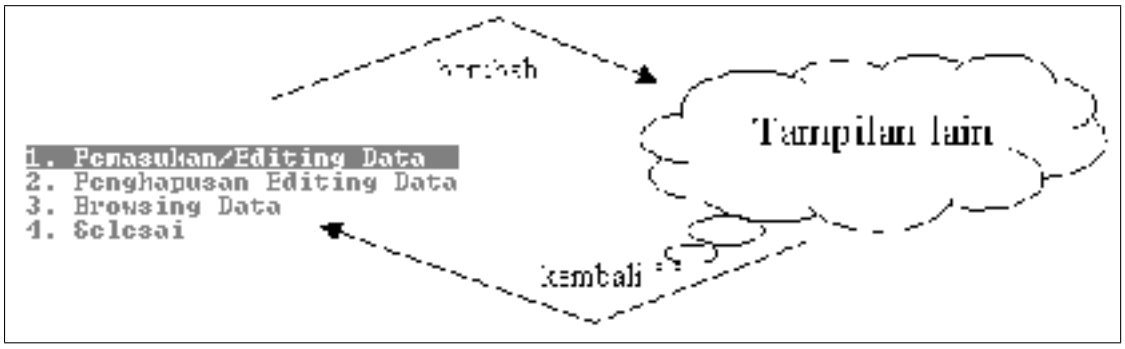
```
Buka file data
Buat indeks sesuai atribut kunci
Tampilkan data dengan dbedit()
Tutup file data dan index
Selesai
```

5.3 Perintah yang Digunakan

Perintah-perintah yang digunakan untuk mengimplementasikan masalah-masalah tersebut di atas ada beberapa perintah. Sebagian perintah sudah dibahas pada bagian-bagian sebelumnya. Di sini hanya akan dibahas perintah-perintah yang baru.

5.3.1 Perintah save screen to

1. Fungsi : Perintah ini dipakai untuk merekam tampilan tulisan di monitor ke memory. Biasanya dilakukan pada suatu proses yang disusul proses lain tapi tampilan akan dikembalikan seperti semula.



Gambar 5.1: Ilustrasi Perubahan Tampilan

2. Sintaks : save screen to nama_variabel

3. Contoh :

```
do while .t.
   @ 07,10 prompt '1. Pemasukan/Editing Data  '
   @ 08,10 prompt '2. Penghapusan Editing Data'
   @ 09,10 prompt '3. Browsing Data           '
   @ 10,10 prompt '4. Selesai                 '
   menu to pilihan
   save screen to lyr_menu
   do case
      case pilihan = 4 .or. lastkey()=K_ESC
           exit
   ...
   endcase
   restore screen from lyr_menu
enddo
```

4. Keterangan : Tampilan menu akan direkam ke variabel lyr_menu. Setelah masuk ke salah satu menu yang dipilih, tampilan layar berubah. Ketika kembali ke menu, maka tampilan dikembalikan ke bentuk semula.

5.3.2 Perintah restore screen from

1. Fungsi : Mengembalikan tampilan ke suatu bentuk tampilan sebelumnya. Untuk bisa mengembalikan tampilan ini, tampilan harus direkam dulu dengan perintah **save screen to**.
2. Sintaks : restore screen from nama_variabel
3. Contoh :

```
save screen to lyr_menu
do case
   case pilihan = 4 .or. lastkey()=K_ESC
        exit
...
endcase
restore screen from lyr_menu
```

4. Keterangan : Perintah ini sangat berhubungan dengan perintah . **save screen to**. Nama_variabel harus sesuai dengan nama_variabel pada **save screen to**.

5.3.3 Perintah seek

1. Fungsi : Melakukan pencarian data sesuai indeks. Hasil pencarian bisa ketemu atau tidak ketemu. Jika ditemukan, maka nilai **found()** adalah .t. (benar) dan jika tidak ditemukan maka nilai **found()** adalah .f. (salah). Dengan adanya mekanisme ini, maka tidak mungkin terjadi data kembar.

2. Sintaks : seek [variabel memory /nilai]

3. Contoh :

```
seek mNO_PKAB
if found()
...
endif

seek '110668'
if found()
...
endif
```

4. Keterangan : Perintah **seek** ini biasanya dilanjutkan dengan perintah **found()** sebagai alat pengecekan apakah pencarian mendapatkan hasil (ketemu) atau tidak mendapatkan hasil (tidak ketemu). Dan sebaliknya, sebagai pendahulu, maka perintah ini harus diawali dengan perintah **index**. Perintah **index** dapat dilihat pada tulisan sebelumnya.

5.3.4 Perintah found()

1. Fungsi : Melakukan pengecekan terhadap hasil pencarian data sesuai indeks. Hasil pencarian bisa ketemu atau tidak ketemu. Jika ditemukan, maka nilai **found()** adalah .t. (benar) dan jika tidak ditemukan maka nilai **found()** adalah .f. (salah). Dengan adanya mekanisme ini, maka tidak mungkin terjadi data kembar.

2. Sintaks : found()

3. Contoh :

```
seek mNO_PKAB
if found()
...
endif

seek '110668'
if found()
...
endif
```

4. Keterangan : Lihat juga contoh pada perintah **if . . . else . . . endif**.

5.3.5 Perintah if . . . else . . . endif

1. Fungsi : Melakukan penyeleksian kondisi sesuai yang dikehendaki dan kemudian melakukan proses sesuai dengan kondisi yang terpenuhi.

2. Sintaks : if [kondisi] . . . else . . . endif

3. Contoh :

```
if .not. found()
   append blank
endif

if found()
   mNAMA_SISWA = NAMA_SISWA
   mNO_INDUK   = NO_INDUK
   mKODE_SMA   = KODE_SMA
   mKELAMIN    = KELAMIN
else
   mNAMA_SISWA = space(30)
   mNO_INDUK   = space(10)
   mKODE_SMA   = space(8)
   mKELAMIN    = space(1)
endif
```

4. Keterangan : Pada contoh pertama, hanya terdapat satu proses yang akan dikerjakan jika kondisi terpenuhi. Yaitu jika data yang dicari tidak ada (**.not. found()**) akan dilakukan penambahan record kosong. Sedangkan pada contoh kedua, ada dua blok proses yang akan dikerjakan tergantung kondisi. Jika ada ditemukan (ada di dalam data), variabel memory akan diisi dengan data yang ada (atribut/*field*). Jika data tidak ada, variabel memory diisi dengan nilai-nilai kosong.

5.3.6 Perintah #include

1. Fungsi : Menyertakan file lain dalam proses kompilasi program. Dengan perintah ini, program atau file yang terpisah dapat saling dikaitkan.

2. Sintaks : #include 'nama_file'

3. Contoh : #include 'inkey.ch'

4. Keterangan : File inkey.ch adalah file yang berisi definisi-definisi tombol. Dengan disertakannya file inkey.ch ini, pembacaaan dan pembuatan program menjadi lebih mudah. Sebagai contoh lihat tabel berikut : Dengan

Nomor	Tombol	Nilai	Nilai inkey.ch
1	Escape	27	K_ESC
2	Enter	13	K_ENTER
3	F1	28	K_F1
4	F2	-1	K_F2
10	...	...	...

Tabel 5.2: Kode Tombol

adanya file inkey.ch, kita tidak perlu menghafal nilai-nilai suatu tombol. Untuk lebih jelasnya bisa buka file inkey.ch.

5.3.7 Perintah lastkey()

1. Fungsi : Mengetahui tombol terakhir yang ditekan.
2. Sintaks : lastkey()
3. Contoh :

```
@05,21 get mNO_PKAB
read
if lastkey()=K_ESC
   exit
endif
```

4. Keterangan : Biasanya digunakan untuk mendeteksi apakah pemakai menekan tombol ESC (misalnya) ketika pemasukan data atau di menu. File inkey.ch adalah file yang berisi definisi-definisi tombol. Dengan disertakannya file inkey.ch ini, pembacaaan dan pembuatan program menjadi lebih mudah. Sebagai contoh lihat tabel berikut : Dengan adanya file inkey.ch, kita tidak perlu menghafal nilai-nilai suatu tombol. Untuk lebih jelasnya bisa buka file inkey.ch.

5.3.8 Perintah dbedit()

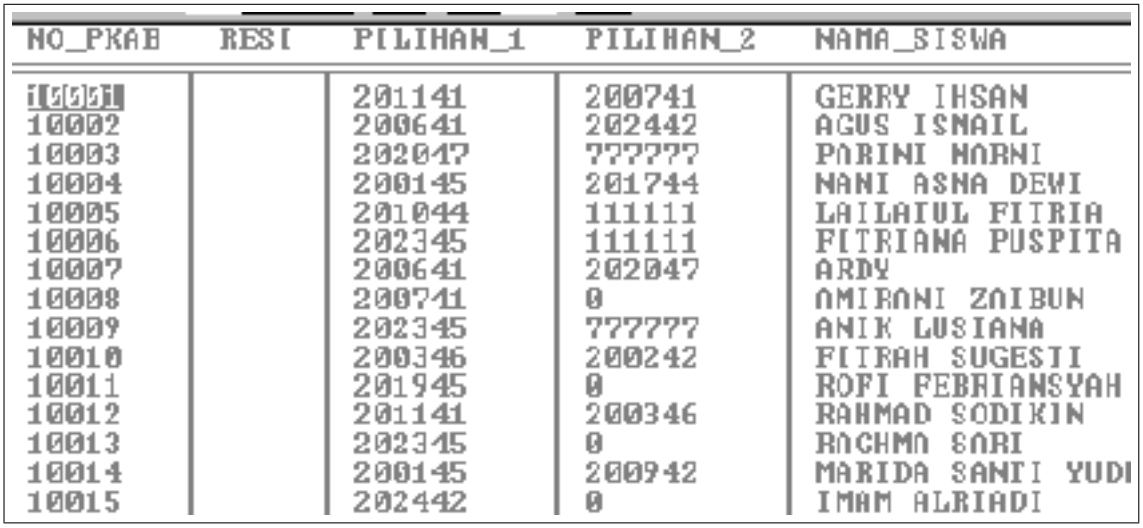
1. Fungsi : Menampilkan isi suatu tabel (file data) dalam bentuk baris dan kolom.
2. Sintaks : dbedit(baris1,kolom1,baris2,kolom2)
3. Contoh :

```
procedure lihat
use ipa2002
index on NO_PKAB to ipa2002
dbedit(0,0,24,78)
close index
close data
return
```

4. Keterangan : Baris1 dan kolom1 menunjukkan posisi kiri atas dan baris2 serta kolom2 menunjukkan posisi kanan bawah. Untuk memindah-mindah penunjuk bisa digunakan tombol panah, pg up, pg dn, home, dan sebagainya.

5.3.9 Perintah delete

1. Fungsi : Menghapus suatu *record* pada posisi *record* aktif.
2. Sintaks : delete
3. Contoh :



NO_PKAB	RESI	PILIHAN_1	PILIHAN_2	NAMA_SISWA
[0000]		201141	200741	GERRY IHSAN
10002		200641	202442	AGUS ISNAIL
10003		202047	777777	PARINI WARNI
10004		200145	201744	NANI ASNA DEWI
10005		201044	111111	LAILAIUL FITRIA
10006		202345	111111	FITRIANA PUSPITA
10007		200641	202047	ARDY
10008		200741	0	AMIRANI ZAIBUN
10009		202345	777777	ANIK LUSIANA
10010		200346	200242	FITRAH SUGESTI
10011		201945	0	ROFI FEBRIANSYAH
10012		201141	200346	RAHMAD SODIKIN
10013		202345	0	RACHMA SARI
10014		200145	200942	MARIDA SANI YUDI
10015		202442	0	IMAM ALRIADI

Gambar 5.2: Tampilan Perintah **dbedit()**

```
seek mNO_PKAB
if found()
   save screen to belum_hapus
   @06,21 say KODE_SMA
   @07,21 say KELAMIN
   @08,21 say NO_INDUK
   @09,21 say NAMA_SISWA
   hapus = 'T'
   @ 11,7 say "Mau dihapus [y/t] : " get hapus
   read
   if hapus='Y' .or. hapus='y' .and. lastkey()#K_ESC
      delete
   endif
   restore screen from belum_hapus
else
   @06,21 say 'Data tidak ada ... !'
   wait ''
   @06,21 say '                    '
endif
```

4. Keterangan : Perintah **delete** diawali dengan perintah **seek** untuk mengarahkan *pointer* ke suatu *record* tertentu. Perintah ini berhubungan dengan perintah **set delete on**.

Contoh selengkapnya dari program untuk pemasukan, editing, penghapusan, dan *browsing* data dapat dilihat seperti berikut :

```
#include 'inkey.ch'
set wrap on
set deleted on
```

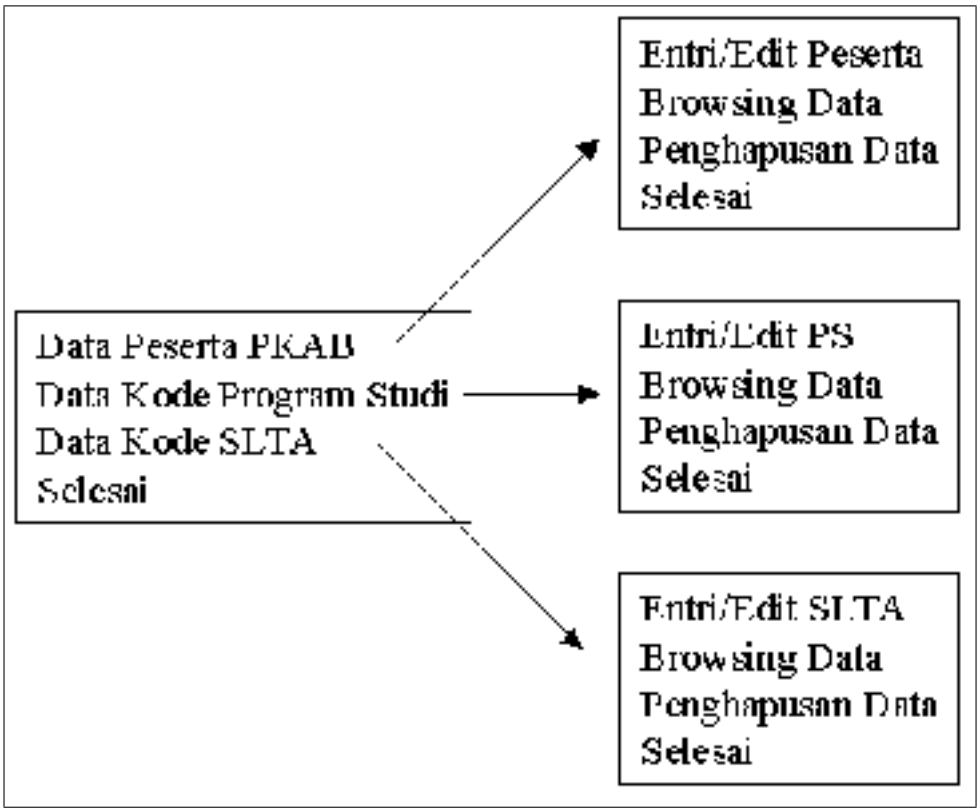
```
clear
do while .t.
   @ 07,10 prompt '1. Pemasukan/Editing Data  '
   @ 08,10 prompt '2. Penghapusan Editing Data'
   @ 09,10 prompt '3. Browsing Data           '
   @ 10,10 prompt '4. Selesai                 '
   menu to pilihan
   save screen to lyr_menu
   clear
   do case
      case pilihan = 4 .or. lastkey()=K_ESC
           exit
      case pilihan = 1
           do entri
      case pilihan = 2
           do hapus
      case pilihan = 3
           do lihat
   endcase
   restore screen from lyr_menu
enddo
return

procedure entri
use ipa2002
index on NO_PKAB to ipa2002
mNO_PKAB = space(5)
do while .t.
   @ 3,7 say 'Pemasukan/Editing Peserta PKAB'
   @ 5,7 say 'No. PKAB    : '
   @ 6,7 say 'Kode SMA    : '
   @ 7,7 say 'Kelamin     : '
   @ 8,7 say 'No. Induk   : '
   @ 9,7 say 'Nama Siswa  : '
   @05,21 get mNO_PKAB
   read
   if lastkey()=K_ESC
      exit
   endif
   seek mNO_PKAB
   if found()
      mNAMA_SISWA = NAMA_SISWA
      mNO_INDUK   = NO_INDUK
      mKODE_SMA   = KODE_SMA
      mKELAMIN    = KELAMIN
   else
      mNAMA_SISWA = space(30)
      mNO_INDUK   = space(10)
      mKODE_SMA   = space(8)
      mKELAMIN    = space(1)
```

```
   endif
   @06,21 get mKODE_SMA
   @07,21 get mKELAMIN
   @08,21 get mNO_INDUK
   @09,21 get mNAMA_SISWA
   read
   if lastkey()#K_ESC
      seek mNO_PKAB
      if .not. found()
         append blank
      endif
      replace NO_PKAB    with mNO_PKAB
      replace NAMA_SISWA with mNAMA_SISWA
      replace NO_INDUK   with mNO_INDUK
      replace KODE_SMA   with mKODE_SMA
      replace KELAMIN    with mKELAMIN
   endif
enddo
close index
close data
return

procedure hapus
use ipa2002
index on NO_PKAB to ipa2002
mNO_PKAB = space(5)
do while .t.
   @ 3,7 say 'Penghapusan Peserta PKAB'
   @ 5,7 say 'No. PKAB    : '
   @ 6,7 say 'Kode SMA    : '
   @ 7,7 say 'Kelamin     : '
   @ 8,7 say 'No. Induk   : '
   @ 9,7 say 'Nama Siswa  : '
   @05,21 get mNO_PKAB
   read
   if lastkey()=K_ESC
      exit
   endif
   seek mNO_PKAB
   if found()
      save screen to belum_hapus
      @06,21 say KODE_SMA
      @07,21 say KELAMIN
      @08,21 say NO_INDUK
      @09,21 say NAMA_SISWA
      hapus = 'T'
      @ 11,7 say "Mau dihapus [y/t] : " get hapus
      read
      if hapus='Y' .or. hapus='y' .and. lastkey()#K_ESC
         delete
```

```
      endif
      restore screen from belum_hapus
   else
      @06,21 say 'Data tidak ada ... !'
      wait ''
      @06,21 say '                    '
   endif
enddo
close index
close data
return

procedure lihat
use ipa2002
index on NO_PKAB to ipa2002
dbedit(0,0,24,78)
close index
close data
return
```

5.4 Tugas Latihan

Setelah mempelajari program tersebut di atas, maka sekarang buatlah program yang memiliki fungsi seperti pada contoh tetapi untuk tabel (data) kode program studi dan kode nama-nama SLTA. Kemudian untuk entri peserta PKAB program juga dilengkapi untuk entri atribut-atribut lainnya. Kemudian program tersebut disatukan dalam program menu sehingga kurang lebih memiliki struktur seperti berikut :



Gambar 5.3: Perancangan Menu Latihan

Bab 6

Validasi *Input* Data

6.1 Pendahuluan

Pada pembahasan sebelumnya, dalam proses pemasukan data tidak dilakukan proses validasi. Proses validasi maksudnya adalah proses pengujian apakah data yang dimasukkan benar. Misalkan pada saat pemasukan Jenis Kelamin, tidak dilakukan pengujian apakah pemakai memasukkan 1 atau 2, atau harus L atau P, dan sebagainya tergantung perancangan. Pemakai dapat mengisi dengan apapun tanpa ada peringatan dari sistem bahwa data yang dimasukkan salah. Juga misalkan pada saat pemasukan kode SMA, pemakai masih dapat memasukkan kode SMA yang tidak ada dalam data. Itulah yang dimaksud dengan belum adanya proses validasi di dalam sistem.

Semestinya sistem memberikan peringatan jika ada kesalahan dalam pemasukan data. Sebab jika tidak dilakukan validasi, maka informasi yang didapat sebagai hasil pengolahan data menjadi tidak akurat. Dengan demikian informasi tidak dapat digunakan dalam pengambilan keputusan.

6.2 Cara Validasi

Secara garis besar ada dua macam cara untuk melakukan validasi. Cara pertama dengan menggunakan perintah **valid** dilanjutkan dengan suatu kondisi. Sedangkan cara kedua menggunakan perintah **valid** dilanjutkan dengan menjalankan suatu fungsi. Pada prinsipnya, kedua model tersebut melakukan prosedur yang sama, yaitu selama perintah **valid** belum menghasilkan nilai .t., maka proses pemasukan data tidak dapat diteruskan ke *attribute* berikutnya.

Seandainya kondisi yang menjadi persyaratan validasi tidak banyak (misal tidak sampai sepuluh), akan lebih mudah menggunakan perintah **valid** kondisinya. Misalnya : validasi kode jenis kelamin, hanya ada laki-laki atau perempuan, validasi pencetakan ke *printer* atau ke layar. Sedangkan untuk validasi yang memiliki banyak kondisi, akan lebih mudah jika kita melakukannya dengan suatu fungsi. Contoh untuk hal ini, misalnya : pengisian kode sekolah, pengisian kode propinsi, pengisian kode kabupaten. Kode-kode yang dimasukkan harus merupakan kode yang benar, artinya kode tersebut ada di dalam kode yang tersedia. Selain untuk validasi, fungsi tersebut juga bisa dikembangkan sehing-

ga dapat dipakai untuk menampilkan kode-kode propinsi, kode-kode sma, kode-kode kabupaten jika pemakai tidak mengetahui kode yang seharusnya.

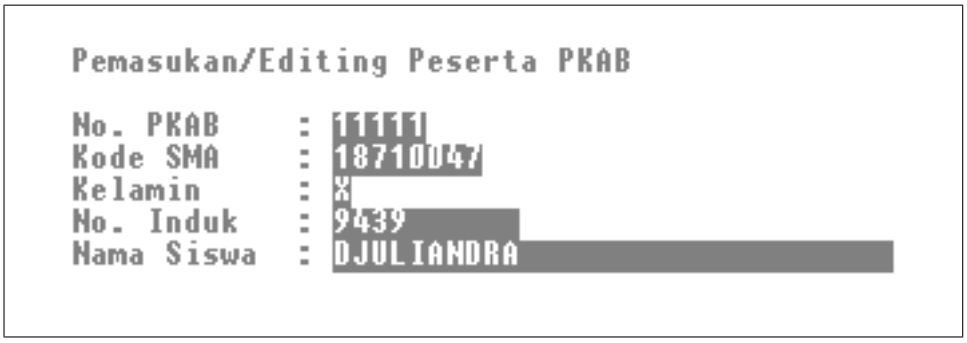
6.3 Validasi dengan Kondisi

Sebelumnya, akan diperlihatkan contoh program yang belum melakukan validasi. Misalkan kita lihat program berikut :

```
* nama program entri.prg
#include 'inkey.ch'
clear
use ipa2002
index on NO_PKAB to ipa2002
mNO_PKAB = space(5)
do while .t.
   @ 3,7 say 'Pemasukan/Editing Peserta PKAB'
   @ 5,7 say 'No. PKAB    : '
   @ 6,7 say 'Kode SMA    : '
   @ 7,7 say 'Kelamin     : '
   @ 8,7 say 'No. Induk   : '
   @ 9,7 say 'Nama Siswa  : '
   @05,21 get mNO_PKAB
   read
   if lastkey()=K_ESC
      exit
   endif
   seek mNO_PKAB
   if found()
      mNAMA_SISWA = NAMA_SISWA
      mNO_INDUK   = NO_INDUK
      mKODE_SMA   = KODE_SMA
      mKELAMIN    = KELAMIN
   else
      mNAMA_SISWA = space(30)
      mNO_INDUK   = space(10)
      mKODE_SMA   = space(8)
      mKELAMIN    = space(1)
   endif
   @06,21 get mKODE_SMA
   @07,21 get mKELAMIN
   @08,21 get mNO_INDUK
   @09,21 get mNAMA_SISWA
   read
   if lastkey()#K_ESC
      seek mNO_PKAB
      if .not. found()
         append blank
      endif
      replace NO_PKAB    with mNO_PKAB
      replace NAMA_SISWA with mNAMA_SISWA
```

```
      replace NO_INDUK   with mNO_INDUK
      replace KODE_SMA   with mKODE_SMA
      replace KELAMIN    with mKELAMIN
   endif
enddo
close index
close data
return
```

Jika program kita jalankan, dan pada item Jenis Kelamin diisi dengan isian yang sembarang, ternyata sistem tidak menolaknya. Apakah hal yang seperti ini akan dibiarkan terjadi ? Ingat bahwa '*Garbage In Garbage Out.* Dari gambar



Gambar 6.1: Sistem Tanpa Validasi

tersebut, terlihat bahwa Jenis Kelamin yang seharusnya diisi dengan 1 untuk laki-laki dan 2 untuk perempuan, ternyata diisi dengan 'X'. Bukan hanya dengan 'X', diisi apa pun, sistem tidak akan menolaknya.

Karena pada pengisian jenis kelamin, hanya ditentukan bahwa jenis kelamin harus diisi dengan laki-laki (1) atau perempuan (2), maka dapat dikatakan bahwa kondisi untuk syarat validasi sedikit (hanya dua kondisi). Oleh karenya akan lebih mudah kalau validasi dilakukan dengan perintah **valid** diikuti dengan kondisi.

6.3.1 Perintah valid

indent

1. Fungsi : Melakukan validasi terhadap data yang dimasukkan oleh pemakai.
2. Sintaks : @ baris,kolom get nama_variabel [pict '9'] [valid ['kondisi'—fungsi]]
3. Contoh :

```
* potongan program ini sama dengan program entri
* hanya ditampilkan yang berbedanya saja
@ 3,7 say 'Pemasukan/Editing Peserta PKAB'
```

```
...
@ 7,7 say 'Kelamin     :        [1:Laki-laki 2:Perempuan]'
...
@ 07,21 get mKELAMIN  picture '9' valid mKELAMIN$'12'

@ 05,21 get mNO_PKAB
read
if lastkey()=K_ESC
   exit
endif
```

4. Keterangan : perintah **picture** untuk membatasi tipe data yang bisa dimasukkan. Perintah **picture** '9' artinya data yang bisa dimasukkan hanya data yang berjenis numeric dan banyaknya satu digit. Sedangkan perintah **valid** mKELAMIN$'12' artinya mKELAMIN harus berupa karakter 1 atau 2. Arti sebenarnya adalah bahwa mKELAMIN harus merupakan karakter yang terkandung dalam karakter atau *string* '12'.

Setelah kita melakukan validasi, maka hasil dari sistem tampak seperti berikut. Dari gambar terlihat bahwa jika *attribute* jenis kelamin diisi dengan data yang

Gambar 6.2: Sistem dengan Validasi

salah, sistem akan menolaknya. Pada contoh, jenis kelamin diisi dengan '9', maka isian tidak dapat diteruskan ke isian berikutnya. Kursor masih tetap pada isian jenis kelamin.

6.4 Validasi dengan Fungsi

Validasi dengan fungsi lebih cocok diterapkan pada proses validasi dengan banyak kondisi. Untuk keperluan itu, kita buat suatu tabel yang berisi data- data. Paling tidak tabel-tabel yang diperlukan berisi atribut KODE dan KETERANGAN. Data yang dimasukkan ke dalam sistem akan dicocokan dengan data di dalam tabel. Jika data yang dimasukkan ternyata tidak ada di dalam tabel, artinya data yang dimasukkan salah dan sistem akan memberi peringatan.

Fungsi yang dibuat harus menghasilkan nilai .t. atau .f. sesuai kondisinya. Nilai .t. didapat jika data yang dimasukkan benar dan sebaliknya. Nilai

yang dimasukkan benar artinya data tersebut ada di dalam tabel. Kemudian KETERANGAN dari KODE yang dimasukkan akan ditampilkan di layar pada posisi baris dan kolom yang ditentukan. Jika ternyata data yang dimasukkan salah, maka sistem akan memberi peringatan 'Kode salah ...!' dan fungsi harus memberikan nilai .f..

6.4.1 *Structured English*

Untuk lebih mempermudah, dapat dilihat proses validasi menggunakan fungsi pada *structured English* berikut :

```
Aktifkan tabel kode SMA
Cari kode yang dimasukkan
Jika ditemukan maka
   Tampilkan nama SMA pada posisi yang ditentukan
   Set nilai hasil fungsi menjadi .t.
jika tidak maka
   Tampilkan penjelasan 'Kode salah' pada
     posisi yang ditentukan
   Set nilai hasil fungsi menjadi .f.
Aktifkan tabel PKAB
Kembalikan nilai hasil fungsi
```

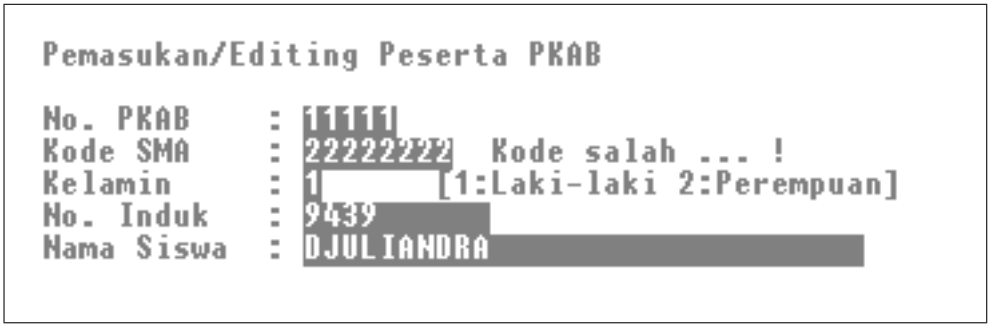
6.4.2 Contoh Program

Contoh program lengkap untuk masalah pemasukkan kode SMA dan jenis kelamin dapat dilihat seperti berikut :

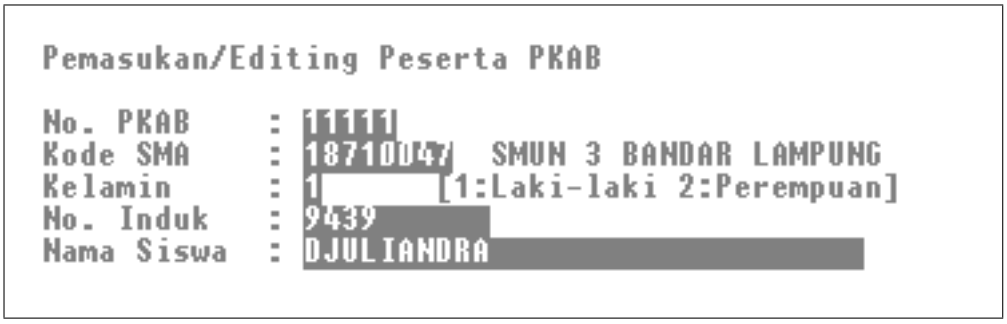
```
#include 'inkey.ch'
clear
select b
use SMA
index on KODE_SMA to SMA

select a
use IPA2002
index on NO_PKAB to ipa2002
mNO_PKAB = space(5)
do while .t.
 @ 3,7 say 'Pemasukan/Editing Peserta PKAB'
 @ 5,7 say 'No. PKAB    : '
 @ 6,7 say 'Kode SMA    : '
 @ 7,7 say 'Kelamin     :    [1:Laki-laki 2:Perempuan]'
 @ 8,7 say 'No. Induk   : '
 @ 9,7 say 'Nama Siswa  : '
 @05,21 get mNO_PKAB
 read
 if lastkey()=K_ESC
    exit
 endif
 seek mNO_PKAB
```

```
 if found()
    mNAMA_SISWA = NAMA_SISWA
    mNO_INDUK   = NO_INDUK
    mKODE_SMA   = KODE_SMA
    mKELAMIN    = KELAMIN
 else
    mNAMA_SISWA = space(30)
    mNO_INDUK   = space(10)
    mKODE_SMA   = space(8)
    mKELAMIN    = space(1)
 endif
 @06,21 get mKODE_SMA pict '99999999' ;
        valid cari_sma(@mKODE_SMA,06,31)
 @07,21 get mKELAMIN  pict '9' valid mKELAMIN$'12'
 @08,21 get mNO_INDUK
 @09,21 get mNAMA_SISWA
 read
 if lastkey()#K_ESC
    seek mNO_PKAB
    if .not. found()
       append blank
    endif
    replace NO_PKAB    with mNO_PKAB
    replace NAMA_SISWA with mNAMA_SISWA
    replace NO_INDUK   with mNO_INDUK
    replace KODE_SMA   with mKODE_SMA
    replace KELAMIN    with mKELAMIN
 endif
enddo
close index
close data
return

function cari_SMA
parameter kode_cari,baris,kolom
select b
seek kode_cari
if found()
   @ baris,kolom say NAMA_SMA
   hasil = .t.
else
   @ baris,kolom say 'Kode salah ... !                    '
   hasil = .f.
endif
select a
return hasil
```

Hasil *running* program jika kode SMA yang dimasukkan salah, tampak seperti berikut : Sedangkan jika kode SMA-nya benar, maka pemasukkan data dapat dilanjutkan ke proses berikutnya.



Gambar 6.3: Sistem dengan Validasi Kode SMA yang Salah



Gambar 6.4: Sistem dengan Validasi Kode SMA yang Benar

6.5 Latihan

Buatlah program untuk memasukkan data-data peserta PKAB sesuai dengan atribut pada tabel yang telah dibuat. Semestinya di sana terdapat pemasukan data program studi pilihan satu dan pilihan dua. Lakukan validasi terhadap pilihan yang dimasukkan ke dalam sistem.

6.6 Apa Resikonya ?

Dengan adanya validasi, pemakai harus memasukkan data benar ke dalam sistem. Masalah yang timbul adalah seandainya pemakai tidak mengetahui dengan benar kode-kode yang harus diisi. Karena jika pemakai belum mengisi dengan isian yang benar, maka selamanya proses pengisian tidak bisa diteruskan.

Untuk itulah perlu disediakan fasilitas bagi pengguna untuk melihat kode-kode yang ada di dalam sistem. Dengan tersedianya fasilitas ini, pemakai dapat melihat daftar kode dan kemudian memilih kode yang ada di dalamnya. Hal ini penting juga, karena tidak jarang data yang tertulis di form isian bukan data yang sebenarnya, walau sudah disediakan buku panduan.

Pembuatan fungsi untuk menampilkan kode-kode yang ada dapat dilakukan dengan banyak variasi. Salah satu yang akan kita lakukan adalah dengan memodifikasi fungsi untuk validasi sehingga dapat digunakan untuk menampilkan

kode. Misal kita menentukan bahwa jika isian kode SMA diisi dengan kosong (spasi), maka sistem akan menampilkan kode-kode yang ada.

6.6.1 *Structured English*

Dengan adanya tambahan fungsi untuk menampilkan kode-kode yang ada, maka fungsi validasi dimodifikasi. Oleh karena itu, fungsi validasi (misal validasi kode SMA) diubah menjadi seperti berikut :

```
Aktifkan tabel kode SMA
Jika kode masih kosong maka
   Tampilkan isi tabel dengan dbedit()
   Jika pemakai menekan tombol ENTER maka
     Ambil kode dari tabel ke varibael cari kode
Cari kode yang dimasukkan
Jika ditemukan maka
   Tampilkan nama SMA pada posisi yang ditentukan
   Set nilai hasil fungsi menjadi .t.
jika tidak maka
   Tampilkan penjelasan 'Kode salah' pada
     posisi yang ditentukan
   Set nilai hasil fungsi menjadi .f.
Aktifkan tabel PKAB
Kembalikan nilai hasil fungsi
```

Fungsi untuk validasi selengkapnya menjadi seperti berikut :

```
function cari_SMA
parameter kode_cari,baris,kolom
select b
if kode_cari=space(7)
   @ 6,23 clear to 23,74
   @ 6,23 to 23,74
   dbedit(7,24,22,73)
   if lastkey()=K_ENTER
      kode_cari = KODE_SMA
   endif
endif
seek kode_cari
if found()
   @ baris,kolom say NAMA_SMA
   hasil = .t.
else
   @ baris,kolom say 'Kode salah ... !                    '
   hasil = .f.
endif
return hasil
```

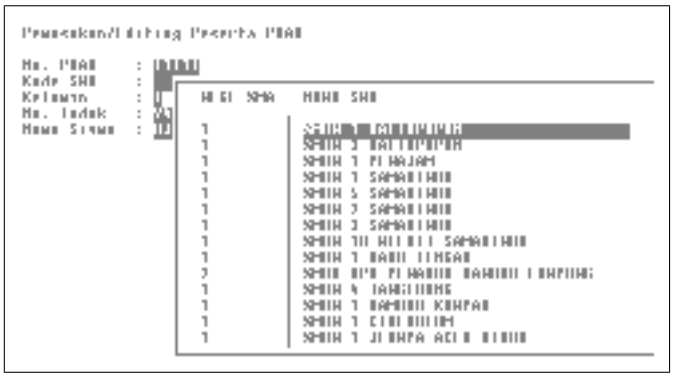
6.6.2 Pengembangan Lebih Lanjut !

Ada beberapa hal yang bisa disebut *bugs* walaupun tidak mengganggu jalannya sistem secara serius. Masalah-masalah tersebut adalah :

1. Tampilan layar yang mengganggu. Jika kita mencoba menampilkan layar tabel kode *help*, semestinya setelah selesai, layar kembali ke tampilan sebelumnya. Contoh tampak seperti berikut :
   Kemudian jika kode SMA diisi dengan isian kosong, maka akan dita-

Gambar 6.5: Tampilan Sebelum Tabel Kode SMA

mpilkan tabel kode SMA seperti berikut :
Setelah selesai pemakai memilih kode SMA yang benar, maka posisi layar



Gambar 6.6: Tampilan Ketika Pencarian Kode SMA

mestinya kembali seperti semula Gambar 6.5, namun pada kenyataannya tampilan malah menjadi seperti berikut :

2. Posisi tabel aktif *data base* semestinya juga kembali ke posisi tabel aktif sebelumnya. Kalau dilihat contoh, maka diawal fungsi tabel aktif adalah IPA2002 dengan nama area **select a** pada akhir fungsi juga harus kembali seperti semula. Pada contoh, memang area kembali ke **select a**. Masalah akan timbul jika area terakhir bukan **select a**.

Adanya masalah-masalah tersebut maka posisi area dan tampilan layar harus disimpan dan pada akhir fungsi dikembalikan lagi. Fungsi masih sama dengan sebelumnya, hanya pada awal dan akhir dimodifikasi dengan memberikan perintah untuk menyimpan nilai area dan tampilan layar.

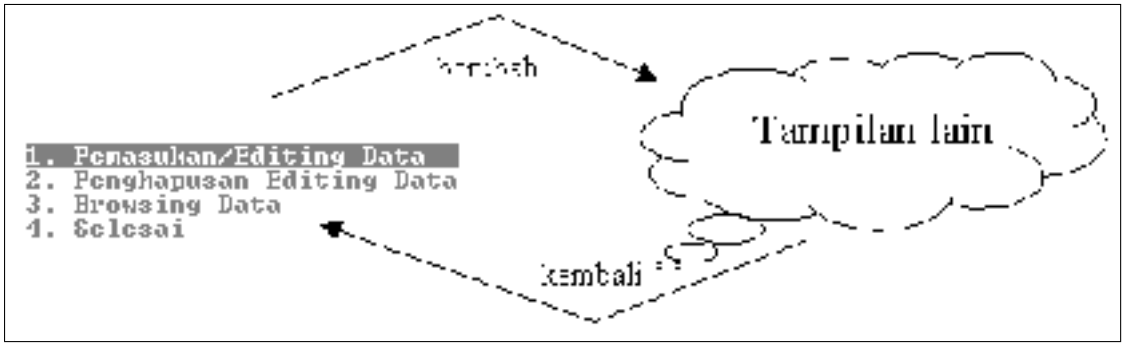
Gambar 6.7: Tampilan Setelah Pencarian Kode SMA

6.6.3 Perintah yang digunakan

Perintah yang digunakan untuk menyimpan nilai area dan tampilan layar seperti berikut :

6.6.4 Perintah save screen to

1. Fungsi : Perintah ini dipakai untuk merekam tampilan tulisan di monitor ke memory. Biasanya dilakukan pada suatu proses yang disusul proses lain tapi tampilan akan dikembalikan seperti semula.



Gambar 6.8: Ilustrasi Perubahan Tampilan

2. Sintaks : save screen to nama_variabel

3. Contoh :

```
do while .t.
   @ 07,10 prompt '1. Pemasukan/Editing Data  '
   @ 08,10 prompt '2. Penghapusan Editing Data'
   @ 09,10 prompt '3. Browsing Data           '
   @ 10,10 prompt '4. Selesai                 '
   menu to pilihan
   save screen to lyr_menu
   do case
      case pilihan = 4 .or. lastkey()=K_ESC
           exit
   ...
   endcase
   restore screen from lyr_menu
enddo
```

4. Keterangan : Tampilan menu akan direkam ke variabel lyr_menu. Setelah masuk ke salah satu menu yang dipilih, tampilan layar berubah. Ketika kembali ke menu, maka tampilan dikembalikan ke bentuk semula.

6.6.5 Perintah restore screen from

1. Fungsi : Mengembalikan tampilan ke suatu bentuk tampilan sebelumnya. Untuk bisa mengembalikan tampilan ini, tampilan harus direkam dulu dengan perintah **save screen to**.

2. Sintaks : restore screen from nama_variabel

3. Contoh :

```
function cari_SMA
parameter kode_cari,baris,kolom
save screen to layar_akhir
select_akhir = select()
select b
if kode_cari=space(7)
   @ 6,23 clear to 23,74
   @ 6,23 to 23,74
   dbedit(7,24,22,73)
   if lastkey()=K_ENTER
      kode_cari = KODE_SMA
   endif
endif
go top
restore screen from layar_akhir
seek kode_cari
if found()
   @ baris,kolom say NAMA_SMA
   hasil = .t.
else
   @ baris,kolom say 'Kode salah ... !                    '
```

```
   hasil = .f.
endif
select(select_akhir)
return hasil
```

4. Keterangan : Perintah ini sangat berhubungan dengan perintah . **save screen to**. Nama_variabel harus sesuai dengan nama_variabel pada **save screen to**.

6.6.6 Perintah select()

1. Fungsi : Menyimpan area *data base* atau tabel yang aktif ke suatu variabel memory. Biasanya perintah ini diletakkan di awal fungsi atau ketika area akan berubah ke area lain. Di samping itu perintah ini juga dapat digunakan untuk mengubah area ke area lain.

2. Sintaks : nama_variabel = select() atau select(nama_area)

3. Contoh :

```
function cari_SMA
parameter kode_cari,baris,kolom
select_akhir=select()
select b
seek kode_cari
if found()
   @ baris,kolom say NAMA_SMA
   hasil = .t.
else
   @ baris,kolom say 'Kode salah ... !                    '
   hasil = .f.
endif
select(select_akhir)
return hasil
```

4. Keterangan : Perintah ini sangat berhubungan dengan perintah **use**. Lihat juga contoh fungsi cari_sma pada bagian sebelumnya.

6.6.7 Perintah restore screen from

Program selengkapnya untuk melakukan pemasukan data, dapat dilihat seperti berikut.

```
* nama file : entriv.prg
#include 'inkey.ch'
clear
select b
use SMA
index on KODE_SMA to SMA
```

```
select a
use IPA2002
index on NO_PKAB to ipa2002
mNO_PKAB = space(5)
do while .t.
   @ 3,7 say 'Pemasukan/Editing Peserta PKAB'
   @ 5,7 say 'No. PKAB    : '
   @ 6,7 say 'Kode SMA    : '
   @ 7,7 say 'Kelamin     :        [1:Laki-laki 2:Perempuan]'
   @ 8,7 say 'No. Induk   : '
   @ 9,7 say 'Nama Siswa  : '
   @05,21 get mNO_PKAB
   read
   if lastkey()=K_ESC
      exit
   endif
   seek mNO_PKAB
   if found()
      mNAMA_SISWA = NAMA_SISWA
      mNO_INDUK   = NO_INDUK
      mKODE_SMA   = KODE_SMA
      mKELAMIN    = KELAMIN
   else
      mNAMA_SISWA = space(30)
      mNO_INDUK   = space(10)
      mKODE_SMA   = space(8)
      mKELAMIN    = space(1)
   endif
   @06,21 get mKODE_SMA pict '99999999' valid cari_sma(@mKODE_SMA,06,31)
   @07,21 get mKELAMIN  pict '9' valid mKELAMIN$'12'
   @08,21 get mNO_INDUK
   @09,21 get mNAMA_SISWA
   read
   if lastkey()#K_ESC
      seek mNO_PKAB
      if .not. found()
         append blank
      endif
      replace NO_PKAB    with mNO_PKAB
      replace NAMA_SISWA with mNAMA_SISWA
      replace NO_INDUK   with mNO_INDUK
      replace KODE_SMA   with mKODE_SMA
      replace KELAMIN    with mKELAMIN
   endif
enddo
close index
close data
return
```

```
function cari_SMA
parameter kode_cari,baris,kolom
save screen to layar_akhir
select_akhir = select()
select b
if kode_cari=space(7)
   @ 6,23 clear to 23,74
   @ 6,23 to 23,74
   dbedit(7,24,22,73)
   if lastkey()=K_ENTER
      kode_cari = KODE_SMA
   endif
endif
go top
restore screen from layar_akhir
seek kode_cari
if found()
   @ baris,kolom say NAMA_SMA
   hasil = .t.
else
   @ baris,kolom say 'Kode salah ... !                    '
   hasil = .f.
endif
select(select_akhir)
return hasil
```

Bab 7

Pemrosesan Data

7.1 Pendahuluan

Apa yang telah dipelajari dalam bagian-bagian terdahulu, masih berhubungan dengan masalah pemasukan data (*entry*). Pembahasan bagian-bagian tersebut dimulai dari persiapan wadah penyimpanan berupa pembuatan tabel (*file .dbf*) sampai kepada masalah validasi pemasukan data. Pada bahasan berikut, kita akan membicarakan masalah pemrosesan data.

Sebagaimana telah disampaikan pada awal buku ini, bahwa masalah yang dibahas dalam buku ini berkaitan dengan simulasi masalah PKAB. Tentu saja apa yang dibahas tidak semuanya sama persis dengan realita. Terutama dalam masalah pemrosesan data ini. Untuk penyederhanaan, penghitungan skor hanya dilakukan dengan menghitung rata-rata mata pelajaran. Skor akhir diperoleh dengan menjumlahkan skor beberapa mata pelajaran kemudian dihitung rata-ratanya. Lantas dimana letak perbedaan dengan kasus sebenarnya ? Dapat dijelaskan di sini, bahwa nilai mata pelajaran yang sama untuk sekolah yang berbeda tentu memiliki bobot yang berbeda pula. Nah ... dalam simulasi ini, tidak dilakukan pembobotan antar sekolah. Hal ini perlu dipahami agar kita dapat membedakan antara simulasi pada materi ini dengan proses yang sebenarnya.

7.2 Modifikasi Program Pemasukan Data

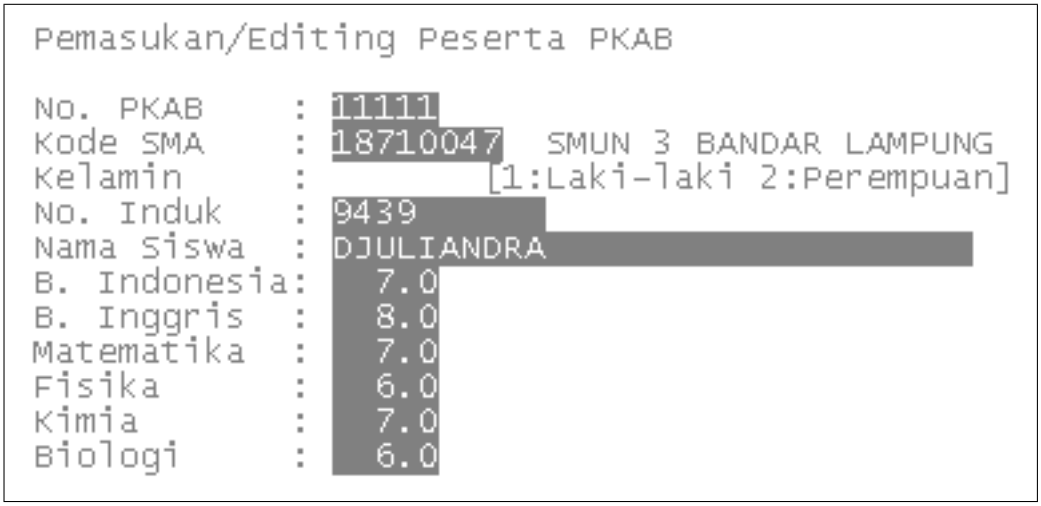
Untuk dapat melanjutkan materi ini, terlebih dahulu struktur file data peserta PKAB yaitu IPA2002.DBF mesti dimodifikasi. Struktur lengkapnya menjadi seperti berikut ini : Dengan demikian, pemasukan data peserta PKAB menjadi seperti berikut :

Field	Field Name	Type	Width	Dec
1	NO_PKAB	Character	5	
2	RESI	Character	1	
3	PILIHAN_1	Character	6	
4	PILIHAN_2	Character	6	
5	NAMA_SISWA	Character	30	
6	NO_INDUK	Character	10	
7	JURUSAN	Character	1	
8	KODE_SMA	Character	8	
9	NAMA_SMA	Character	35	
10	KELAMIN	Character	1	
11	INA_1	Numeric	4	1
12	INGG_1	Numeric	4	1
13	MAT_1	Numeric	4	1
14	FIS_1	Numeric	4	1
15	KIM_1	Numeric	4	1
16	BIO_1	Numeric	4	1
17	SKOR	Numeric	7	3
18	TANDA	Character	10	
19	PILIHAN	Character	6	
20	PIL_LAIN	Character	6	
21	KET_PILIH	Character	1	
22	TAHAP	Character	1	

Tabel 7.1: Atribut Tabel IPA2002.DBF

Contoh program lengkap dapat dilihat seperti berikut :

```
* nama file : ful_ent.prg
#include 'inkey.ch'
clear
select b
use SMA
index on KODE_SMA to SMA
select a
use IPA2002
index on NO_PKAB to ipa2002
mNO_PKAB = space(5)
do while .t.
   @ 3,7 say 'Pemasukan/Editing Peserta PKAB'
   @ 5,7 say 'No. PKAB    : '
   @ 6,7 say 'Kode SMA    : '
   @ 7,7 say 'Kelamin     :        [1:Laki-laki 2:Perempuan]'
   @ 8,7 say 'No. Induk   : '
   @ 9,7 say 'Nama Siswa  : '
   @10,7 say 'B. Indonesia: '
   @11,7 say 'B. Inggris  : '
```



Gambar 7.1: Tampilan Pemasukan Data Peserta PKAB

```
@12,7 say 'Matematika  : '
@13,7 say 'Fisika      : '
@14,7 say 'Kimia       : '
@15,7 say 'Biologi     : '
@05,21 get mNO_PKAB
read
if lastkey()=K_ESC
   exit
endif
seek mNO_PKAB
if found()
   mNAMA_SISWA = NAMA_SISWA
   mNO_INDUK   = NO_INDUK
   mKODE_SMA   = KODE_SMA
   mKELAMIN    = KELAMIN
   mINA_1      = INA_1
   mINGG_1     = INGG_1
   mMAT_1      = MAT_1
   mFIS_1      = FIS_1
   mKIM_1      = KIM_1
   mBIO_1      = BIO_1
else
   mNAMA_SISWA = space(30)
   mNO_INDUK   = space(10)
   mKODE_SMA   = space(8)
   mKELAMIN    = space(1)
   mINA_1      = 0
   mINGG_1     = 0
   mMAT_1      = 0
   mFIS_1      = 0
```

```
      mKIM_1      = 0
      mBIO_1      = 0
   endif
   @06,21 get mKODE_SMA pict '99999999' valid cari_sma(@mKODE_SMA,06,31)
   @07,21 get mKELAMIN  pict '9' valid mKELAMIN$'12'
   @08,21 get mNO_INDUK
   @09,21 get mNAMA_SISWA
   @10,21 get mINA_1   pict '999.9'
   @11,21 get mINGG_1  pict '999.9'
   @12,21 get mMAT_1   pict '999.9'
   @13,21 get mFIS_1   pict '999.9'
   @14,21 get mKIM_1   pict '999.9'
   @15,21 get mBIO_1   pict '999.9'
   read
   if lastkey()#K_ESC
      seek mNO_PKAB
      if .not. found()
         append blank
      endif
      replace NO_PKAB    with mNO_PKAB
      replace NAMA_SISWA with mNAMA_SISWA
      replace NO_INDUK   with mNO_INDUK
      replace KODE_SMA   with mKODE_SMA
      replace KELAMIN    with mKELAMIN
      replace INA_1      with mINA_1
      replace INGG_1     with mINGG_1
      replace MAT_1      with mMAT_1
      replace FIS_1      with mFIS_1
      replace KIM_1      with mKIM_1
      replace BIO_1      with mBIO_1
   endif
enddo
close index
close data
return

function cari_SMA
parameter kode_cari,baris,kolom
save screen to layar_akhir
select_akhir = select()
select b
if kode_cari=space(7)
   @ 6,23 clear to 23,74
   @ 6,23 to 23,74
   dbedit(7,24,22,73)
   if lastkey()=K_ENTER
      kode_cari = KODE_SMA
   endif
endif
go top
```

```
restore screen from layar_akhir
seek kode_cari
if found()
   @ baris,kolom say NAMA_SMA
   hasil = .t.
else
   @ baris,kolom say 'Kode salah ... !                    '
   hasil = .f.
endif
select(select_akhir)
return hasil
```

7.3 Penghitungan Skor

Perlu ditekankan sekali lagi bahwa perhitungan skor yang dilakukan di sini hanya sekedar contoh dan untuk memudahkan maka dilakukan penyederhaan proses perhitungan. Skor akhir diperoleh dengan merata-rata nilai pada beberapa mata pelajaran yaitu Bahasa Indonesia, Bahasa Inggris, Matematika, Fisika, Kimia dan Biologi. Proses ini dilakukan untuk semua data yang ada.

Structured English untuk proses perhitungan tersebut dapat kita susun seperti berikut :

```
Buka file IPA2002
Hitung jumlah data
Set nomor = 0
Kerjakan dari awal sampai akhir data
 {
   Tambahkan nilai counter nomor
   Hitung jumlah nilai
   Hitung nilai rata-rata
   Simpan nilai rata-rata ke file data base
   Tampilkan keterangan proses
   Lanjutkan ke data berikutnya
 }
```

Dengan contoh hasil dan programnya seperti berikut :

```
Nomor PKAB             : 11363
Nama Siswa             : DEDI PRAPAT
Skor Akhir             :   3.33
Sedang proses data ke  :  3323
Prosentasi proses      : 100.00 %
```

Gambar 7.2: Tampilan Pemrosesan Data Peserta PKAB

```
* nama file : proses.prg
#include 'inkey.ch'
clear
use IPA2002
count to jml_data
store 0 to nomor
dbgotop()
do while .not. eof()
   nomor     = nomor + 1
   jml_nilai = INA_1 + INGG_1 + MAT_1 + FIS_1 + KIM_1 + BIO_1
   rata      = jml_nilai/6
   replace SKOR with rata
   @ 09,20 say 'Nomor PKAB            : '+NO_PKAB
   @ 10,20 say 'Nama Siswa            : '+NAMA_SISWA
   @ 11,20 say 'Skor Akhir            : '+str(rata,6,2)
   @ 12,20 say 'Sedang proses data ke : '+str(nomor,5,0)
   @ 13,20 say 'Prosentasi proses     : '+str(nomor/jml_data*100,6,2)+' %'
   dbskip()
enddo
return
```

Selanjutnya kita tinggal merangking peserta berdasarkan pilihan masing- masing dan skornya.

Bab 8

Pencetakan Informasi

8.1 Pendahuluan

Pencetakan informasi menjadi bagian yang sangat penting. Mengapa ? Bagaimana pun rumit dan canggihnya suatu sistem informasi, tetapi manakala sistem itu sendiri tidak dapat memberikan informasi (tentu saja dalam bentuk cetakan), apalah artinya. Seringkali informasi dalam bentuk cetakan menjadi dokumen yang sangat lebih komunikatif bagi pemakai dan itulah yang menjadi bahan pengambilan keputusan.

Sebagaimana umumnya proses pencetakan, pencetakan bisa dilakukan ke *printer* atau ke layar. Pencetakan ke layar berfungsi semacam *preview* sebelum dicetak langsung ke *printer*.

8.2 Perintah-perintah yang Digunakan

Di sini akan dijelaskan perintah-perintah khusus yang berhubungan langsung dengan pencetakan. Perintah-perintah lain, tentu saja ada yang berkaitan namun perintah-perintah tersebut sudah dijelaskan. Sebagai contoh, tidak mungkin mencetak informasi tanpa ada tabel *data base* yang digunakan artinya perintah **user** otomatis harus digunakan.

8.2.1 Perintah set printer to

1. Fungsi : Perintah ini dipakai untuk mengarahkan pencetakan ke suatu *file* teks atau tidak.
2. Sintaks : set printer to [nama_file]
3. Contoh :

```
set printer to hasil.txt
```

4. Keterangan : Sedangkan untuk membatalkan pencetakan ke file, cukup dengan perintah **set printer to** tanpa tambahan apa-apa. Pencetakan ke *file* diperlukan manakala kita akan mencetak ke layar. Sebab pencetakan

ke layar, pada hakekatnya adalah pencetakan ke suatu *file* teks dan kemudian *file* teks tersebut di- *import* ke suatu tabel *file data base*. Dari *file data base* kemudian ditampilkan ke layar dengan perintah **dbedit()**.

8.2.2 Perintah set device to

1. Fungsi : Perintah ini dipakai untuk mengarahkan pencetakan ke suatu *printer* atau layar. Jika perintah **set printer to** diarahkan ke suatu *file* maka pencetakan akan diarahkan ke *file* tersebut, jika tidak maka pencetakan diarahkan ke *printer*.
2. Sintaks : set device to [screen—printer]
3. Contoh :

```
set printer to hasil.txt
set device to printer
```

4. Keterangan : Rangkaian perintah di atas akan mengarahkan pencetakan ke *file* teks bernama hasil.txt. dbedit().

8.2.3 Perintah printer [on—off]

1. Fungsi : Perintah ini dipakai untuk mengaktifkan atau mematikan *printer*.
2. Sintaks : set printer [on—off]
3. Contoh :

```
set printer to hasil.txt
set device to printer
set printer on
```

4. Keterangan : Rangkaian perintah di atas akan mengarahkan pencetakan ke *file* teks bernama hasil.txt.

8.2.4 Perintah append from ... sdf

1. Fungsi : Fungsi **append from ... sdf** digunakan untuk memasukkan isi satu file teks ke tabel aktif (yang sedang di-**use**). File teks yang akan dimasukkan diasumsikan sudah ada sebagai hasil pencetakan ke file. Data yang dimasukkan sesuai nama atribut *field* masing-masing.
2. Sintaks : **append from** [file teks] sdf
3. Contoh :
   (a) use layar
   (b) append from hasil.txt sdf

8.2.5 Perintah dbedit()

1. Fungsi : Menampilkan isi suatu tabel (file data) dalam bentuk baris dan kolom.
2. Sintaks : dbedit(baris1,kolom1,baris2,kolom2)
3. Contoh :

```
do case
   case alat=1
     set device to screen
     set printer off
     set printer to
     use layar
     zap
     append from hasil.txt sdf
     dbgotop()
     dbedit(0,0,24,79)
   case alat=2
     set device to screen
     set printer off
endcase
```

4. Keterangan : Baris1 dan kolom1 menunjukkan posisi kiri atas dan baris2 serta kolom2 menunjukkan posisi kanan bawah. Untuk memindah-mindah penunjuk bisa digunakan tombol panah, pg up, pg dn, home, dan sebagainya.

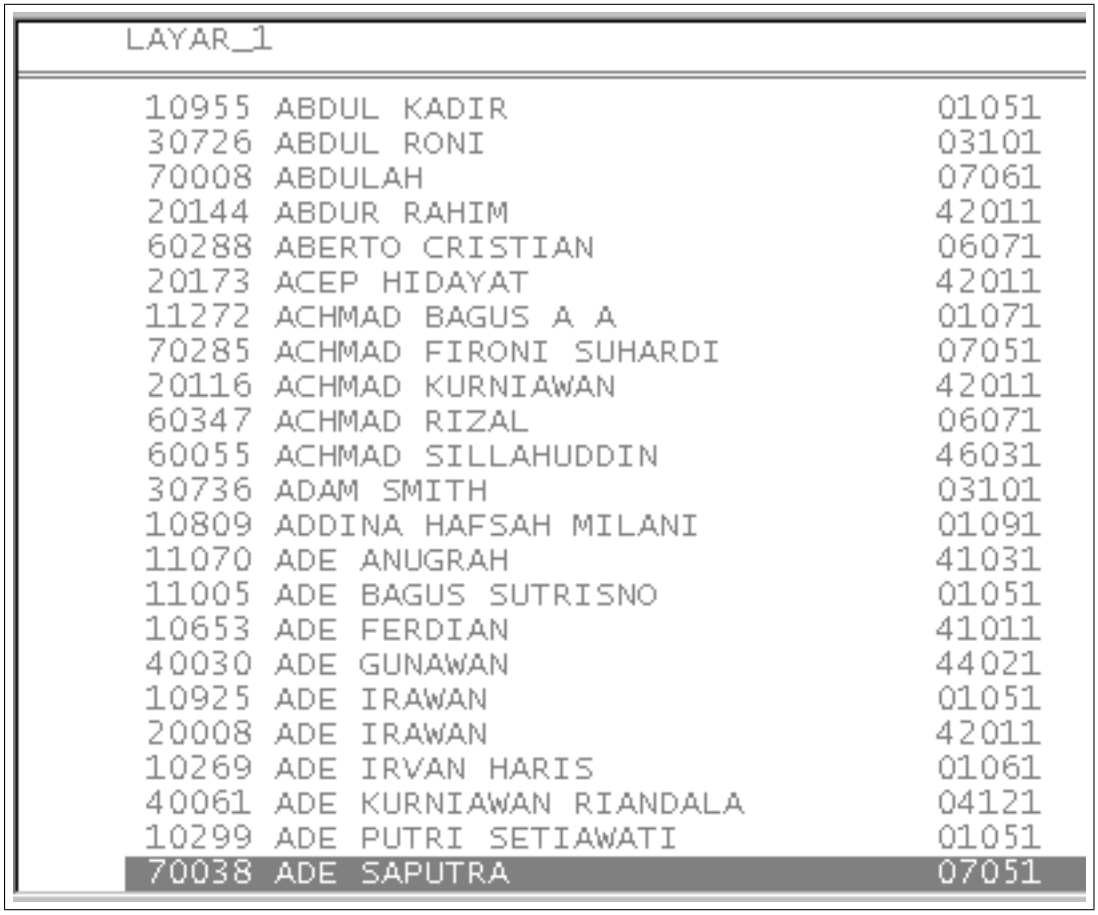
8.2.6 Perintah zap

1. Fungsi : Fungsi **zap** digunakan untuk menghapus seluruh data. Data akan terhapus tapi struktur data masih ada.
2. Sintaks : **zap**
3. Contoh : zap
4. Tampilan : -

8.3 Penting Diperhatikan !

Suatu hal yang penting untuk diperhatikan yaitu bahwa sebelum proses pencetakan harus disiapkan kondisi-kondisi pendahuluan. Kondisi-kondisi tersebut terlihat seperti pada perintah berikut ini :

```
@ 15,11 prompt ' L. Layar   '
@ 16,11 prompt ' P. Printer '
menu to alat
do case
   case alat=1
```



Gambar 8.1: Tampilan Perintah **dbedit()**

```
          set printer to hasil.txt
          set printer on
          set device to print
     case alat=2
          set printer on
          set device to print
endcase
```

Setelah selesai, maka dilakukan proses penutupan kondisi. Proses penutupan ini dilakukan dengan memberi parameter kebalikan dari proses sebelumnya dan penulisan baris program juga dalam urutan terbalik seperti berikut :

```
do case
   case alat=1
          set device to screen
          set printer off
          set printer to
          use layar
```

```
          zap
          append from hasil.txt sdf
          dbgotop()
          dbedit(0,0,24,79)
     case alat=2
          set device to screen
          set printer off
  endcase
```

8.4 *Structured English*

Structured English untuk masalah pencetakan informasi dapat dilihat seperti pada contoh berikut. Tentu saja, karena ini hanya sebagai contoh, pada aplikasi lain harus disesuaikan dengan keperluan.

```
Kerjakan selama benar
{ Buka file data
  Tampilkan menu jenis Urutan Pencetakan
  Jika menekan tombol ESC maka
    Keluar looping
  Tampilkan menu cetak ke layar atau printer
  Jika cetak ke layar maka
    Set pencetakan ke file teks
  Jika cetak ke printer maka
    Siapkan printer
  Jika mencetak berdasarkan urutan nama maka
    {
      Index data menurut urutan nama
      Cetak data yang diperlukan dari awal sampai akhir
    }
  Jika mencetak berdasarkan urutan skor akhir maka
    {
      Index data menurut urutan skor
      Cetak data yang diperlukan dari awal sampai akhir
    }
  Jika cetak ke layar maka
    {
      Import file teks hasil cetakan ke file dbf
      Tampilkan isi file dbf dengan dbedit()
    }
  Jika cetak kr printer maka
    Non aktifkan printer
}
Selesai
```

Dengan contoh program lengkapnya seperti berikut ini :

```
#include 'inkey.ch'
do while .t.
```

```
wall(0,0,24,79)
use ipa2002
@ 09,10 clear to 17,65
@ 10,11 prompt ' A. Cetak Daftar Peserta Urut Nama '
@ 11,11 prompt ' B. Cetak Daftar Peserta Urut Skor '
@ 12,11 prompt ' X. Selesai ...                    '
menu to cetak
if lastkey()=K_ESC .or. cetak=3
   exit
endif
@ 15,11 prompt ' L. Layar   '
@ 16,11 prompt ' P. Printer '
menu to alat
do case
   case alat=1
     set printer to hasil.txt
     set printer on
     set device to print
   case alat=2
     set printer on
     set device to print
endcase
do case
   case cetak=1
     index on NAMA_SISWA to nama
     dbgotop()
     do while .not. eof()
        @ prow()+1,1 say NO_PKAB+' '+NAMA_SISWA+' '+PILIHAN_1+' '+NAMA_SMA
        dbskip()
     enddo
   case cetak=2
     index on PILIHAN_1+str(1000-SKOR) to skor
     dbgotop()
     do while .not. eof()
        @ prow()+1,1 say NO_PKAB+' '+NAMA_SISWA+' '+PILIHAN_1+' '+str(SKOR,8,3)
        dbskip()
     enddo
endcase
do case
   case alat=1
     set device to screen
     set printer off
     set printer to
     use layar
     zap
     append from hasil.txt sdf
     dbgotop()
     dbedit(0,0,24,79)
   case alat=2
     set device to screen
```

```
        set printer off
   endcase
enddo
return
```

Jika program ini dijalankan, kita dapat memilih cetak ke layar atau *printer*. Seandainya terdapat *printer* dan kita memilih cetak ke *printer*, maka hasil pencetakan akan langsung ke *printer*. Untuk dapat menjalankan program ini, maka kita harus memiliki file data IPA2002.DBF sebagaimana telah dibuat diawal dan file data LAYAR.DBF. File data LAYAR.DBF memiliki struktur seperti berikut :

```
Structure for database: LAYAR.DBF
Number of data records:    3800
Date of last update   : 12/23/02
Field  Field Name  Type       Width    Dec
    1  LAYAR_1     Character     70
    2  LAYAR_2     Character     70
    3  LAYAR_3     Character     70
    4  LAYAR_4     Character     70
** Total **                    281
```

Bab 9

Integrasi Program

9.1 Pendahuluan

Meskipun kita sudah membahas pembuatan menu, namun dapat dikatakan bahwa program yang dibuat masih terpisah-pisah. Pada aplikasi yang dipakai oleh pelanggan, sebaiknya aplikasi sudah terintegrasi. Oleh karena itu, pada bagian ini kita akan membahas bagaimana menggabungkan program-program yang sudah dibuat.

Jika Anda mengikuti secara runut dan mengerjakan setiap latihan dengan baik, Anda akan memiliki *file* data dan program seperti berikut :

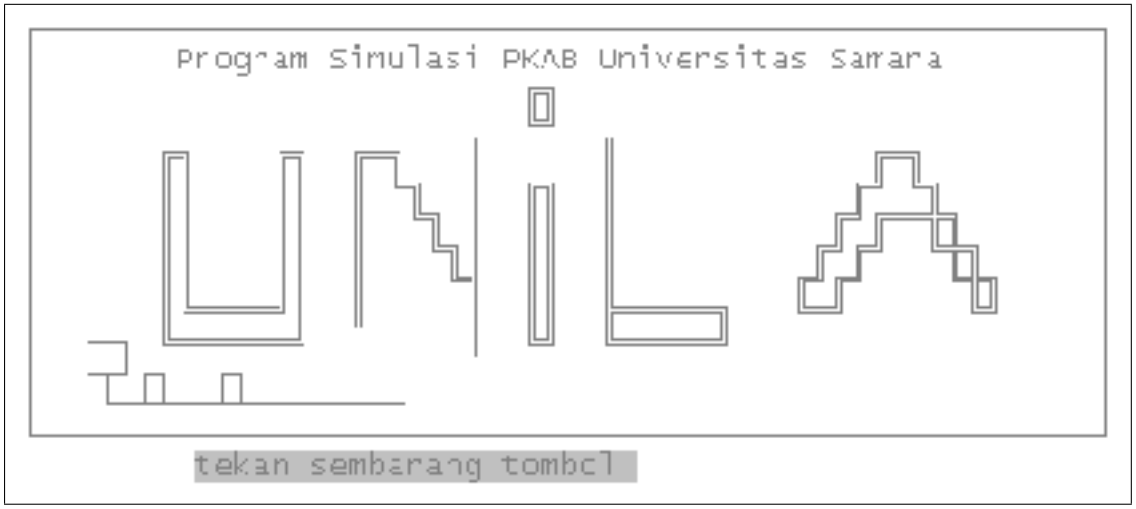
```
 Directory of C:\dwi\clipper

 IPA2002  DBF       529,095  12-26-02 11:14a ipa2002.dbf
 SLTA     DBF       127,648  09-14-02  7:58p SLTA.DBF
 LAYAR    DBF       934,206  12-26-02  7:05p LAYAR.DBF
 PS       DBF         6,760  05-21-02  7:43p PS.DBF

 CETAK    PRG         1,524  12-26-02  6:56p CETAK.PRG
 LOGO     PRG         1,126  06-14-02  2:58a LOGO.PRG
 HAPUS    PRG           958  11-04-02  8:47p HAPUS.PRG
 LIHAT    PRG            66  11-11-02  7:58p LIHAT.PRG
 ENT_FUL  PRG         2,806  12-26-02 10:32a ENT_FUL.PRG
 PROSES   PRG           607  12-26-02 11:14a PROSES.PRG
```

Program menu yang dibuat selengkapnya dapat dilihat pada bagian berikut. Sedangkan isi bagian-bagian program tidak ditampilan ulang karena sama dengan program-program pada bagian sebelumnya.

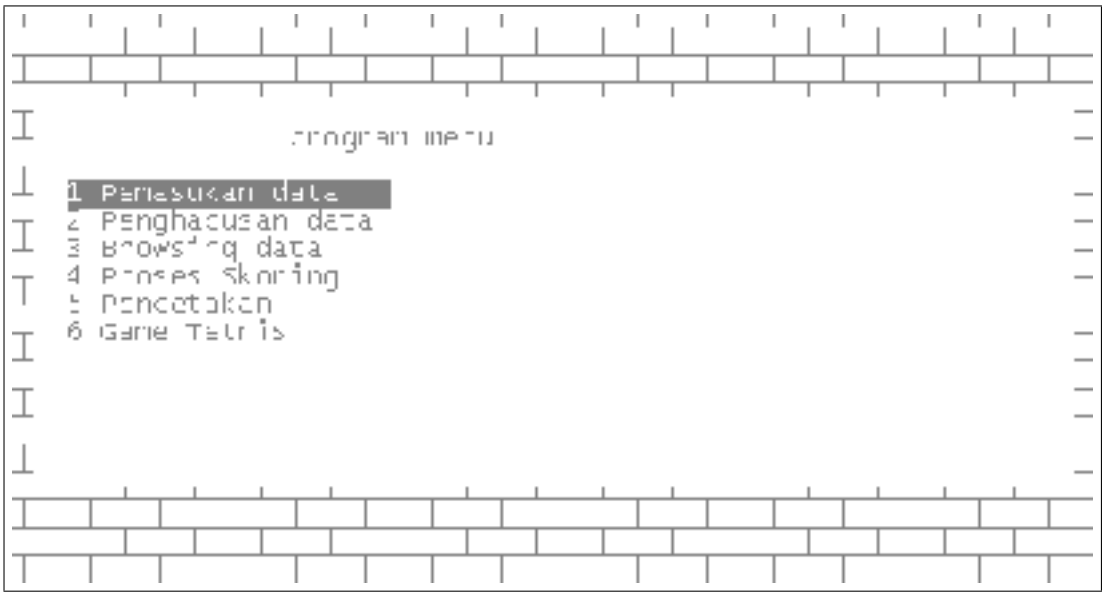
```
* nama file inte.prg
#include 'inkey.ch'
tone(500,10)
set wrap on
do logo
clear
do while .t.
   wall (0,0,24,79)
```

```
   @7,15 clear to 20,75
   @8,30 say 'program menu'
   @10,17 prompt "1 Pemasukan data   "
   @11,17 prompt "2 Penghapusan data "
   @12,17 prompt "3 Browsing data    "
   @13,17 prompt "4 Proses Skoring   "
   @14,17 prompt "5 Pencetakan       "
   @15,17 prompt "6 Game Tetris      "
   menu to pil_utama
   if lastkey() = K_ESC
       exit
   endif
   do case
      case pil_utama =1
           do ent_ful
      case pil_utama = 2
            do  hapus
      case pil_utama = 3
            do lihat
      case pil_utama = 4
            do proses
      case pil_utama = 5
            do cetak
      case pil_utama = 6
            tetris()
    end case
enddo
return
```

Tampilan program tampak seperti berikut :



Gambar 9.1: Tampilan Logo Pembuka



Gambar 9.2: Tampilan Menu Gabungan Program

Bagian IV

Clipper *Advanced*

Bab 10

Optimasi dan Improvisasi

10.1 Improvisasi Program

Pada bagian-bagian sebelumnya, pembuatan program dengan Clipper dilakukan dengan tidak memperhatikan masalah-masalah optimasi dan kesempurnaan program. Oleh karenanya, hal-hal tersebut akan dibahas pada bagian-bagian berikut.

Dalam kondisi tertentu (data masih sedikit, tidak ada kelainan kondisi) maka program akan berjalan normal. Namun hal itu akan menjadi masalah manakala, misalnya data sudah banyak, file data hilang atau tidak ada, dan sebagainya. Dengan demikian, program perlu dioptimasi dan ada improvisasi untuk mengurangi kesalahan-kesalahan program.

10.2 Operasi File

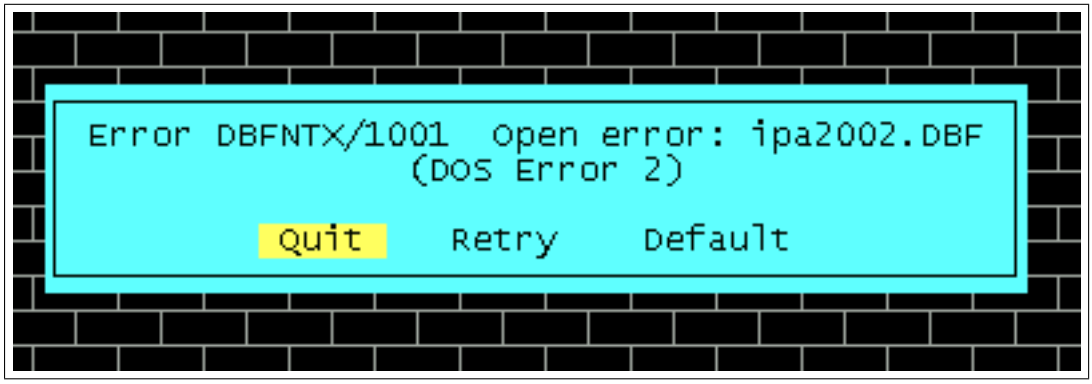
Operasi *file* yang dimaksud di sini adalah membuka dan menutup file data. Sebagaimana sudah kita pahami bersama, secara mudah perintah untuk membuka itshape file data adalah **use** dan untuk menutupnya adalah **close data**. Masalah akan muncul manakala *file* data yang hendak digunakan tidak ada. Misalkan kita lihat contoh berikut :

```
clear
wall(0,0,24,79)
use ipa2002
dbedit(5,7,20,75)
return
```

Jika *file* ipa2002.dbf tidak ada, maka program akan timbul *error*. *Error* yang muncul tampak seperti berikut :

10.3 Perintah-Perintah

Perintah baru yang digunakan untuk improvisasi program adalah :



Gambar 10.1: *Error* Membuka File

10.3.1 Perintah file()

1. Fungsi : Mengetahui ada tidaknya suatu *file*.
2. Sintaks : **file('nama-file.ext')**
3. Contoh : file('ipa2002.dbf')
4. Keterangan : Fung file('ipa2002.dbf') akan menghasilkan nilai .t. jika *file* ipa2002.dbf ada dan sebaliknya.

Bagian V

Penutup Buku

Bab 11

Penutup

11.1 Harapan dan Impian

Apa-apa yang disampaikan dalam buku ini masih sangat sederhana alias apa adanya. Sebagai contoh, dalam suatu proses tidak dilakukan optimalisasi algoritma. Hal-hal tersebut mudah-mudahan akan dibahas pada kesempatan lain (*Insyaa Allah*) jika ada. Mengapa ? Buku ini memang ditujukan untuk pembaca yang baru belajar pemrograman Clipper.

Kalau para pembaca yang budiman pernah memiliki pengalaman buruk mencoba contoh-contoh program tetapi ternyata program tersebut tidak dapat dieksekusi, maka mudah-mudahan semua contoh yang ada pada buku ini dapat dicoba dengan baik dan memberi pelajaran kepada kita semua. Contoh-contoh yang ada sudah dikompilasi dan dieksekusi dan kemudian tidak ada yang disembunyikan.

Dengan penggunaan perangkat lunak LaTeXdalam penulisan buku ini, maka semoga ini menjadi inspirasi pemakaian perangkat lunak yang legal. Selain memang *features* yang tersedia dan perangkat LaTeXyang *powerfull*. Kapan bangsa ini akan terlepas dari predikat 'pembajak perangkat lunak' dan koruptor ?

Kami mengucapkan terima kasih jika Anda berkenan memberi masukan-masukan dalam penulisan buku ini. Semoga bermanfaat ... meskipun nampaknya buku ini hanyalah sebutir pasir di padang pasir nan luas.

Indeks

Buku Panduan

Pemrograman Clipper

Oleh :

Dwi Sakethi, S.Si, M.Kom

http://dwijim.tux.nu

Jurusan Matematika

Fakultas Matematika dan Ilmu Pengetahuan Alam

Universitas Lampung

Bandar Lampung 2003