ALGORITMA RUN-LENGTH HALF-BYTE & HUFFMAN

untuk

PEMAMPATAN FILE

Penyusun:
Herry Sujaini (23299043)
Yessi Mulyani (23299518)

KATA PENGANTAR

Alhamdulillah, puja dan puji syukur kami haturkan kepada Allah SWT, karena dengan hidayah serta inayah-Nya kami dapat menyelesaikan buku kecil yang kami beri judul "Algoritma Run-Length, Half Byte dan Huffman untuk Pemampatan File" ini. Shalawat serta salam tak lupa kami sampaikan kepada Nabi Besar Muhammad SAW.

Sebagai konsekwensi peserta mata kuliah "***Jaringan Komputer (EL-592)***" pada program S-2 di Jurusan Teknik Elektro, Option Teknik Sistem Komputer Institut Teknologi Bandung tahun ajaran 1999/2000, buku ini kami susun dalam rangka tugas yang diberikan oleh Bapak Onno W. Purbo sebagai Dosen Penanggung Jawab mata kuliah tersebut.

Sebagai pemula, tentu saja banyak sekali kekurangan yang terdapat pada buku ini, baik dari segi bahasa, penguraian maupun materinya, untuk itu kritik dan saran yang membangun sangat kami harapkan agar kami dapat memperbaiki buku ini, juga untuk dijadikan masukan saat menulis buku-buku lainnya.

Harapan kami, semoga buku kecil ini dapat bermanfaat bagi kita semua. Aamiin.

Bandung, Mei 2000

Penyusun,

DAFTAR ISI

Halaman

BAB I

PENDAHULUAN

Dalam dunia komputer dan internet, pemampatan file digunakan dalam berbagai keperluan, jika anda ingin mem-backup data, anda tidak perlu menyalin semua file aslinya, dengan memampatkan (mengecilkan ukurannya) file tersebut terlebih dahulu maka kapasitas tempat penyimpanan yang diperlukan akan menjadi lebih kecil. Jika sewaktu-waktu data tersebut anda perlukan, baru dikembalikan lagi ke file aslinya.

Down-load dan *Up-load* file suatu pekerjaan yang kadang mengesalkan pada dunia internet, setelah menghabiskan beberapa waktu kadang-kadang hubungan terputus dan anda harus melakukannya lagi dari awal, hal ini sering terjadi pada file-file yang berukuran besar. Untunglah file-file tersebut dapat dimampatkan terlebih dahulu sehingga waktu yang diperlukan akan menjadi lebih pendek dan kemungkinan pekerjaan *down-load* dan *up-load* gagal akan menjadi lebih kecil.

Dua orang mahasiswa mendapatkan tugas untuk melakukan penelitian mengenai warna baju yang digunakan oleh orang-orang yang lewat di suatu jalan tertentu. Tugasnya mudah saja, jika ada orang lewat dengan baju berwarna merah, mereka cukup menulis "merah" pada buku pencatat, begitu juga dengan warna lain.

Pada suatu saat lewat pada jalan tersebut serombongan tentara yang berjumlah 40 orang, semuanya memakai seragam berwarna hijau. Mahasiswa pertama menulis pada buku pencatat "hijau, hijau, hijau …. " sampai 40 kali, tapi mahasiswa kedua ternyata lebih cerdik, dia hanya menulis pada buku pencatat "hijau 40 x".

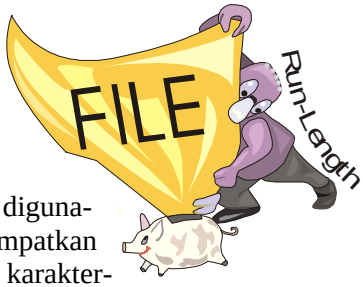
Setelah selesai melaksanakan tugas mereka, ternyata mahasiswa pertama menghabiskan 10 lembar catatan, sedangkan mahasiswa kedua hanya menghabiskan 5 lembar catatan, sedangkan hasil mereka tidak ada bedanya.

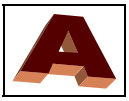
Cara yang digunakan oleh mahasiswa kedua tersebut dapat digunakan pada pemampatan file, dan dikenal dengan algoritma *Run-Length*. Selain algoritma *Run-Length,* buku kecil ini juga akan membahas pemampatan file dengan algoritma *Half-Byte* dan *Huffman* masing-masing pada BAB II, BAB III, dan BAB IV. Pada BAB V akan dijelaskan bagaimana mengimplemen-tasikan algoritma-algoritma tersebut menjadi sebuah program software. Selanjutya pada BAB VI akan dibahas bagaimana hasilnya kalau suatu file hasil pemampatan dimampatkan lagi. Bagi anda yang kurang memahami pengetahuan dasar mengenai dasar bilangan serta operasi logika, kami sarankan untuk mempelajarinya kembali pada BAB VII, hal ini ini penting untuk membantu mempermudah pengertian terhadap penjelasan-penjelasan yang diberikan pada buku ini.

Selain algoritma yang disebutkan di atas sebenarnya masih banyak lagi algoritma-algoritma lain dalam pemamampatan data, seperti algoritma *differential*, algorima *hierarchical*, dan lain-lain, namun mengingat terbatasnya buku ini, kami hanya membahas algoritma *Run-Length, Half-Byte* dan *Huffman.*

BAB II

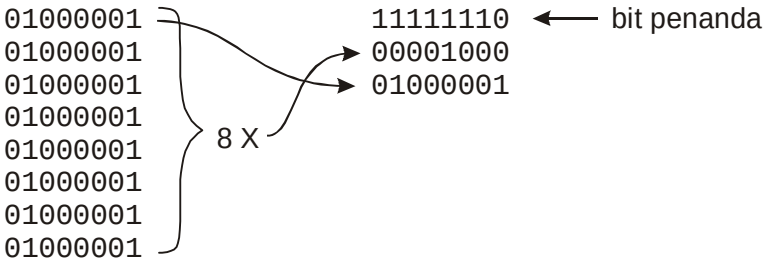
ALGORITMA RUN-LENGTH





lgoritma *Run-length* digunakan untuk memampatkan data yang berisi karakter-karakter berulang. Saat karakter yang sama diterima secara berderet empat kali atau lebih (lebih dari tiga), algoritma ini mengkompres data dalam suatu tiga karakter berderetan. Algoritma *Run-Length* paling efektif pada file-file grafis, dimana biasanya berisi deretan panjang karakter yang sama.

Metode yang digunakan pada algoritma ini adalah dengan mencari karakter yang berulang lebih dari 3 kali pada suatu file untuk kemudian diubah menjadi sebuah bit penanda (*marker bit*) diikuti oleh sebuah bit yang memberikan informasi jumlah karakter yang berulang dan kemudian ditutup dengan karakter yang dikompres, yang dimaksud dengan bit penanda disini adalah deretan 8 bit yang membentuk suatu karakter ASCII. Jadi jika suatu file mengandung karakter yang berulang, misalnya AAAAAAAA atau dalam biner 01000001 sebanyak 8 kali, maka data tersebut dikompres menjadi 11111110 00001000 01000001. Dengan demikian kita dapat menghemat sebanyak 5 bytes. Agar lebih jelas algoritma *Run-Length* dapat digambarkan sebagai berikut :



Deretan data sebelah kiri merupakan deretan data pada file asli, sedangkan deretan data sebelah kanan merupakan deretan data hasil pemampatan dengan algoritma *Run-Length*. Langkah-langkah yang dilakukan adalah :

1. Lihat apakah terdapat deretan karakter yang sama secara berurutan lebih dari tiga karakter, jika memenuhi lakukan pemampatan. Pada contoh di atas deretan karakter yang sama secara berurutan sebanyak 8 karakter, jadi lebih dari tiga dan dapat dilakukan pemampatan.
2. Berikan bit penanda pada file pemampatan, bit penanda disini berupa 8 deretan bit yang boleh dipilih sembarang asalkan digunakan secara konsisten pada seluruh bit penanda pemampatan. Bit penanda ini berfungsi untuk menandai bahwa karakter selanjutnya adalah karakter pemampatan sehingga tidak membingungkan pada saat mengembalikan file yang sudah dimampatkan ke file aslinya. Pada contoh di atas bit penanda ini dipilih 11111110.
3. Tambahkan deretan bit untuk menyatakan jumlah karakter yang sama berurutan, pada contoh diatas karakter yang sama berturutan sebanyak delapan kali, jadi diberikan deretan bit 00001000 (8 desimal).

4. Tambahkan deretan bit yang menyatakan karakter yang berulang, pada contoh diatas karakter yang berulang adalah 01000001 atau karakter A pada karakter ASCII.

Untuk melakukan proses pengembalian ke data asli (decompression), dilakukan langkah-langkah berikut ini :

1. Lihat karakter pada hasil pemampatan satu-persatu dari awal sampai akhir, jika ditemukan bit penanda, lakukan proses pengembalian.
2. Lihat karakter setelah bit penanda, konversikan ke bilangan desimal untuk menentukan jumlah karakter yang berurutan.
3. Lihat karakter berikutnya, kemudian lakukan penulisan karakter tersebut sebanyak bilangan yang telah diperoleh pada karakter sebelumnya (point 2).

Sebagai contoh lain jika sebuah file berisi karakter berturut-turut
00001111
11110000
11110000
11110000
11110000
11110000
11110000
10101010
10101010
10101010

Jika dimampatkan dengan metoda *Run-Length*, hasilnya akan menjadi
00001111
11111110
00000110
10101010

10101010
10101010

Dengan langkah-langkah pengembalian yang telah dijelaskan di atas, akan didapatkan hasil yang sama seperti file aslinya.

Coba anda lakukan sendiri pemampatan dengan metoda *Run-Length* pada deretan karakter berikut :
00001111
11110000
11110000
11110000
11110000
11110000
10101010
10101010
10101010
10101010

Hasilnya akan berjumlah 7 bytes. Kemudian lakukan pengembalian ke file aslinya.

Jika anda akan membuat program pemampatan data dengan algoritma ini, ada beberapa hal yang perlu diperhatikan.

Pemilihan bit penanda diusahakan dipilih pada karakter yang paling sedikit jumlahnya terdapat pada file yang akan dimampatkan, sebab jika pada file asli ditemukan karakter yang sama dengan bit penanda, terpaksa anda harus menulis karakter tersebut sebanyak dua kali pada file pemampatan. Hal ini harus dilakukan untuk menghindari kesalahan mengenali apakah bit penanda pada file pemampatan tersebut benar-benar bit penanda atau memang karakter dari file yang asli. Sebagai contoh jika terdapat deretan data pada file asli seperti berikut ini :

11111110
11110000
11110000
11110000
11110000
11110000
11110000
10101010
10101010
10101010

Dengan cara seperti yang telah dijelaskan sebelumnya kita dapatkan hasil pemampatan sebagai berikut :
11111110
11111110
00000110
10101010
10101010
10101010

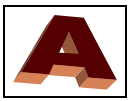
Jika dilakukan proses pengembalian ke file aslinya (decompression) maka akan diperoleh hasil :
11111110
00000110
.
. (sebanyak 11111110 = 254 kali)
.
00000110
10101010
10101010
10101010

Ternyata hasil tersebut tidak sesuai dengan file aslinya. Untuk menjaga agar hal tersebut tidak terjadi, jika pada file asli terdapat karakter yang sama dengan bit penanda, maka

pada file pemampatan karakter tersebut ditulis sebanyak dua kali secara berturutan. Pada saat pengembalian ke file asli, jika ditemukan bit penanda yang berderetan sebanyak dua kali, hal itu berarti karakter tersebut bukan bit penanda, tapi karakter asli dari file aslinya.

BAB III

ALGORITMA HALF-BYTE



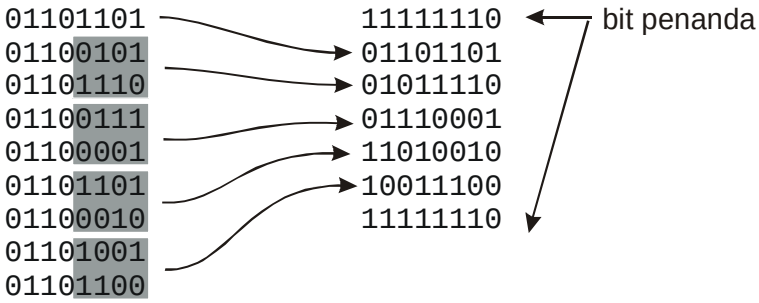
lgoritma *Half-Byte* memanfaatkan empat bit sebelah kiri yang sering sama secara berurutan terutama pada file-file text. Misalnya pada suatu file text berisi tulisan "mengambil", dalam heksadesimal dan biner karakter-karakter tersebut diterjemahkan sebagai :

Karakter	Heksadesimal	Biner
m	6D	01101101
e	65	01100101
n	6E	01101110
g	67	01100111
a	61	01100001
m	6D	01101101
b	62	01100010
i	69	01101001
l	6C	01101100

Jika anda perhatikan karakter-karakter tersebut memiliki empat bit sebelah kiri yang sama yaitu 0110. Gejala seperti inilah yang dimanfaatkan oleh Algoritma *Half-Byte*.

Saat karakter yang empat bit pertamanya sama diterima secara berderet tujuh kali atau lebih, algoritma ini

mengkompres data tersebut dengan bit penanda kemudian karakter pertama dari deretan empat bit yang sama diikuti dengan pasangan empat bit terakhir deretan berikutnya dan ditutup dengan bit penutup. Algoritma ini paling efektif pada file-file text dimana biasanya berisi text-text yang memiliki empat bit pertama yang sama. Agar lebih jelas algoritma *Half-Byte* dapat digambarkan sebagai berikut :



Deretan data sebelah kiri merupakan deretan data pada file asli, sedangkan deretan data sebelah kanan merupakan deretan data hasil pemampatan dengan algoritma *Half-Byte*. Langkah-langkah yang dilakukan adalah :

1. Lihat apakah terdapat deretan karakter yang 4 bit permanya sama secara berurutan tujuh karakter atau lebih, jika memenuhi lakukan pemampatan. Pada contoh di atas deretan karakter yang sama secara berurutan sebanyak 9 karakter, jadi dapat dilakukan pemampatan.
2. Berikan bit penanda pada file pemampatan, bit penanda disini berupa 8 deretan bit (1 byte) yang boleh dipilih sembarang asalkan digunakan secara konsisten pada seluruh bit penanda pemampatan. Bit penanda ini berfungsi untuk menandai bahwa karakter selanjutnya adalah karakter pemampatan sehingga tidak membingungkan pada saat mengembalikan file yang

sudah dimampatkan ke file aslinya. Pada contoh di atas bit penanda ini dipilih 11111110.

3. Tambahkan karakter pertama 4 bit kiri berurutan dari file asli, pada contoh diatas karakter pertama 4 bit kiri berurutan adalah 01101101.
4. Gabungkan 4 bit kanan karakter kedua dan ketiga kemudian tambahkan ke file pemampatan. Pada contoh di atas karakter kedua dan ketiga adalah 01100101 dan 01101110, gabungan 4 bit kanan kedua karakter tersebut adalah 01011110. Lakukan hal ini sampai akhir deretan karakter dengan 4 bit pertama yang sama.
5. Tutup dengan bit penanda pada file pemampatan.

Untuk melakukan proses pengembalian ke data asli (decompression), dilakukan langkah-langkah berikut ini :

1. Lihat karakter pada hasil pemampatan satu-persatu dari awal sampai akhir, jika ditemukan bit penanda, lakukan proses pengembalian.
2. Lihat karakter setelah bit penanda, tambahkan karakter tersebut pada file pengembalian.
3. Lihat karakter berikutnya, jika bukan bit penanda, ambil 4 bit kanannya lalu gabungkan dengan 4 bit kanan karakter di bawahnya. Hasil gabungan tersebut ditambahkan pada file pengembalian. Lakukan sampai ditemukan bit penanda.

Sebagai contoh lain jika sebuah file berisi karakter berturut-turut
01101110
01111111
01111111
01111010
01111100
01111100
01111100
01110000
01110111
01110111
00011000

Jika dimampatkan dengan metoda *Half-Byte*, hasilnya akan menjadi
01101110
11111110
01111111
11111010
11001100
11000000
01110111
11111110
00011000

Dengan langkah-langkah pengembalian yang telah dijelaskan di atas, akan didapatkan hasil yang sama seperti file aslinya.

Coba anda lakukan sendiri pemampatan dengan metoda *Half-Byte* pada deretan karakter berikut :
01101110
01111111
01111111

01111010
01111100
01111100
01111100
01110000
01111100
01110000
01110111
01110111

Hasil pemampatannyanya akan berjumlah 9 bytes. Kemudian lakukan pengembalian ke file aslinya.

Jika anda akan membuat program pemampatan data dengan algoritma ini, ada beberapa hal yang perlu diperhatikan.

Pemilihan bit penanda diusahakan dipilih pada karakter yang paling sedikit jumlahnya terdapat pada file yang akan dimampatkan, sebab jika pada file asli ditemukan karakter yang sama dengan bit penanda, terpaksa anda harus menulis karakter tersebut sebanyak dua kali pada file pemampatan. Hal ini harus dilakukan untuk menghindari kesalahan mengenali apakah bit penanda pada file pemampatan tersebut benar-benar bit penanda atau memang karakter dari file yang asli. Sebagai contoh jika terdapat deretan data pada file asli seperti berikut ini :

01111111
11111110
00000000
01111100
01111100
01111000
01110000
01111100
01110000
01110111

Dengan cara seperti yang telah dijelaskan sebelumnya kita dapatkan hasil pemampatan sebagai berikut :

01111111
11111110
00000000
11111110
01111100
11001000
00001100
00000111
11111110

Jika dilakukan proses pengembalian ke file aslinya (decompression) maka akan diperoleh hasil :

01111111
00000000
01111100
11001000
00001100
00000111
11111110

Ternyata hasil tersebut tidak sesuai dengan file aslinya. Untuk menjaga agar hal tersebut tidak terjadi, jika pada file asli terdapat karakter yang sama dengan bit penanda, maka pada file pemampatan karakter tersebut ditulis sebanyak dua kali secara berturutan. Pada saat pengembalian ke file asli, jika ditemukan bit penanda yang berderetan sebanyak dua kali, hal itu berarti karakter tersebut bukan bit penanda, tapi karakter asli dari file aslinya.

Pada kasus lain dapat terjadi penggabungan 4 bit kanan menghasilkan sebuah karakter yang sama dengan bit penanda sehingga diduga karakter itu adalah bit penutup,

misalnya terdapat deretan data pada file asli seperti berikut ini :
01111100
01111100
01111000
01111111
01111110
01110000
01111100
01110000
01110111

Dengan algoritma *Half-Byte* kita dapatkan hasil pemampatan sebagai berikut :

11111110
01111100
11001000
11111110
00001100
00000111
11111110

Jika dilakukan proses pengembalian ke file aslinya (decompression) maka akan diperoleh hasil :

01111100
01111100
01111000
00001100
00000111
11111110

Ternyata hasil tersebut tidak sesuai dengan file aslinya. Untuk menjaga agar hal tersebut tidak terjadi, jika terdapat

penggabungan 4 bit kanan yang menghasilkan sebuah karakter yang sama dengan bit penanda, maka deretan file tersebut tidak usah dimampatkan.

Pada contoh-contoh di atas, jumlah karakter berurutan yang memiliki 4 bit pertama sama jumlahnya ganjil, jika ditemukan kasus jumlahnya genap maka karakter terakhir tidak perlu dimampatkan, contohnya jika pada file asli terdapat karakter-karakter sebagai berikut :

01111100
01111100
01111000
01111000
01110000
01110000
01111100
01110000

Karena karakter-karakter di atas berjumlah 8 (genap) maka yang dimampatkan hanya karakter 1 sampai 7, sedangkan karakter terakhir (0111000) tidak perlu dimampatkan.

BAB IV

ALGORITMA HUFFMAN

Tidak seperti algoritma *Run-Length* atau *Half-Byte*, algoritma *Huffman* lebih rumit penanganan-nya, tapi jika anda pelajari dengan sabar keterangan berikut ini, penulis yakin anda akan dapat memahaminya dengan baik.

Dasar pemikiran algoritma ini adalah bahwa setiap karakter ASCII biasanya diwakili oleh 8 bits. Jadi misalnya suatu file berisi deretan karakter "ABACAD" maka ukuran file tersebut adalah 6 x 8 bits = 48 bit atau 6 bytes. Jika setiap karakter tersebut di beri kode lain misalnya A=1, B=00, C=010, dan D=011, berarti kita hanya perlu file dengan ukuran 11 bits (10010101011), yang perlu diperhatikan ialah bahwa kode-kode tersebut harus unik atau dengan kata lain suatu kode tidak dapat dibentuk dari kode-kode yang lain. Pada contoh diatas jika kode D kita ganti dengan 001, maka kode tersebut dapat dibentuk dari kode B ditambah dengan kode A yaitu 00 dan 1, tapi kode 011 tidak dapat dibentuk dari kode-kode yang lain. Selain itu karakter yang paling sering muncul, kodenya diusahakan lebih kecil jumlah bitnya dibandingkan dengan karakter yang jarang muncul. Pada contoh di atas karakter A lebih sering muncul (3 kali), jadi kodenya dibuat lebih kecil jumlah bitnya dibanding karakter lain.

Untuk menetukan kode-kode dengan kriteria bahwa kode harus unik dan karakter yang sering muncul dibuat kecil jumlah bitnya, kita dapat menggunakan algoritma *Huffman*.

Sebagai contoh, sebuah file yang akan dimampatkan berisi karakter-karakter "PERKARA". Dalam kode ASCII masing-masing karakter dikodekan sebagai :

P = 50H = 01010000B
E = 45H = 01000101B
R = 52H = 01010010B
K = 4BH = 01001011B
A = 41H = 01000001B

Maka jika diubah dalam rangkaian bit, "PERKARA" menjadi :

01010000010001010101001001001011010000010101001001000001
P E R K A R A

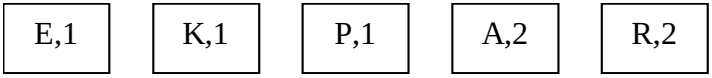
yang berukuran 56 bit.

Tugas kita yang pertama adalah menghitung frekuensi kemunculan masing-masing karakter, jika kita hitung ternyata P muncul sebanyak 1 kali, E sebanyak 1 kali, R sebanyak 2 kali, K sebanyak 1 kali dan A sebanyak 2 kali. Jika disusun dari yang kecil :

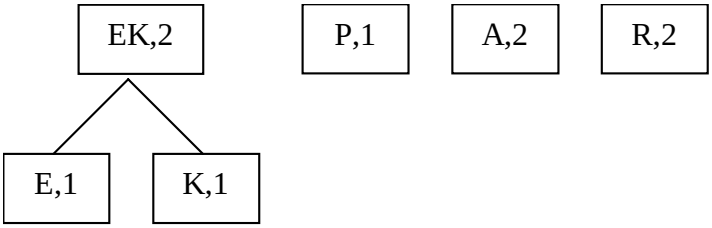
E = 1
K = 1
P = 1
A = 2
R = 2

Untuk karakter yang memiliki frekuensi kemunculan sama seperti E, K dan P disusun menurut kode ASCII-nya, begitu pula untuk A dan R.

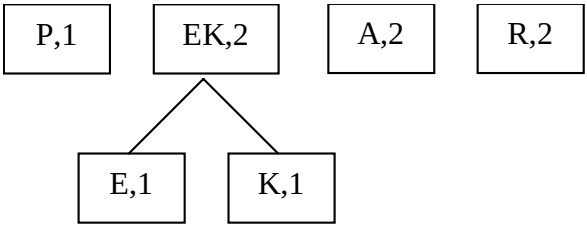
Selanjutnya buatlah node masing-masing karakter beserta frekuensinya sebagai berikut :



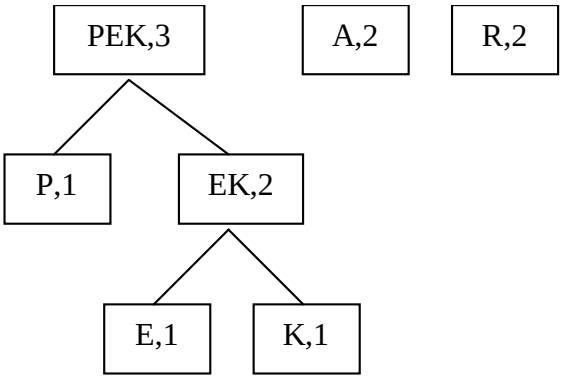
Ambil 2 node yang paling kiri (P dan E), lalu buat node baru yang merupakan gabungan dua node tersebut, node gabungan ini akan memiliki cabang masing-masing 2 node yang digabungkan tersebut. Frekuensi dari node gabungan ini adalah jumlah frekuensi cabang-cabangnya. Jika kita gambarkan akan menjadi seperti berikut ini :



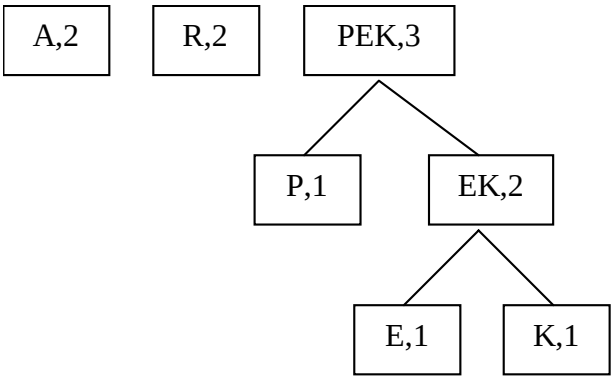
Jika kita lihat frekuensi tiap node pada level paling atas, EK=2, P=1, A=2, dan R=2. Node-node tersebut harus diurutkan lagi dari yang paling kecil, jadi node EK harus digeser ke sebelah kanan node P dan ingat jika menggeser suatu node yang memiliki cabang, maka seluruh cabangnya harus diikutkan juga. Setelah diuurutkan hasilnya akan menjadi sebagai berikut :



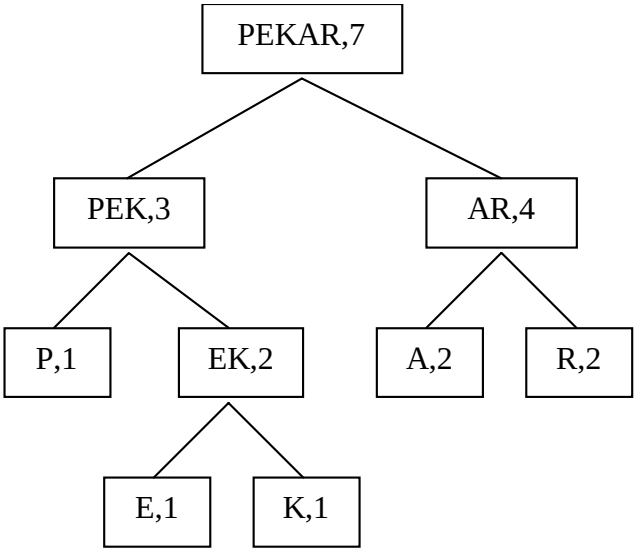
Setelah node pada level paling atas diurutkan (level berikutnya tidak perlu diurutkan), berikutnya kita gabungkan kembali 2 node paling kiri seperti yang pernah dikerjakan sebelumnya. Node P digabung dengan node EK menjadi node PEK dengan frekuensi 3 dan gambarnya akan menjadi seperti berikut ini :



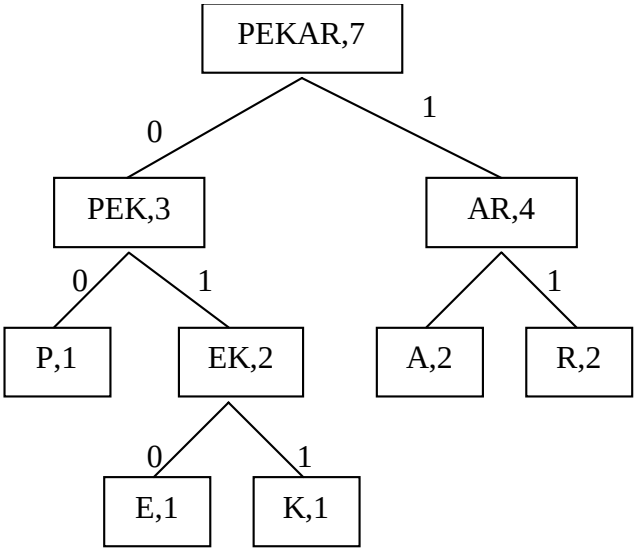
Kemudian diurutkan lagi menjadi :



Demikian seterusnya sampai diperoleh pohon Huffman seperti gambar berikut ini :



Setelah pohon Huffman terbentuk, berikan tanda bit 0 untuk setiap cabang ke kiri dan bit 1 untuk setiap cabang ke kanan seperti gambar berikut :



Untuk mendapatkan kode Huffman masing-masing karakter, telusuri karakter tersebut dari node yang paling atas (PEKAR) sampai ke node karakter tersebut dan susunlah bit-bit yang dilaluinya.

Untuk mendapatkan kode Karakter E, dari node PEKAR kita harus menuju ke node PEK melalui bit 0 dan selanjutnya menuju ke node EK melalui bit 1, dilanjutkan ke node E melalui bit 0, jadi kode dari karakter E adalah 010.

Untuk mendapatkan kode Karakter K, dari node PEKAR kita harus menuju ke node PEK melalui bit 0 dan selanjutnya menuju ke node EK melalui bit 1, dilanjutkan ke node K melalui bit 1, jadi kode dari karakter K adalah 011.

Untuk mendapatkan kode Karakter P, dari node PEKAR kita harus menuju ke node PEK melalui bit 0 dan selanjutnya menuju ke node P melalui bit 0, jadi kode dari karakter P adalah 00.

Untuk mendapatkan kode Karakter A, dari node PEKAR kita harus menuju ke node AR melalui bit 1 dan selanjutnya

menuju ke node A melalui bit 0, jadi kode dari karakter A adalah 10.

Terakhir, untuk mendapatkan kode Karakter R, dari node PEKAR kita harus menuju ke node AR melalui bit 1 dan selanjutnya menuju ke node R melalui bit 1, jadi kode dari karakter R adalah 11.

Hasil akhir kode *Huffman* dari file di atas adalah :

E = 010
K = 011
P = 00
A = 10
R = 11

Dengan kode ini, file yang berisi karakter-karakter "PERKARA" akan menjadi lebih kecil, yaitu :

00 010 11 011 10 11 10 = 16 bit
P E R K A R A

Dengan Algoritma *Huffman* berarti file ini dapat kita hemat sebanyak 56-16 = 40 bit.

Untuk proses pengembalian ke file aslinya, kita harus mengacu kembali kepada kode *Huffman* yang telah dihasilkan, seperti contoh di atas hasil pemampatan adalah :

000101101100 1110

dengan Kode *Huffman* :

E = 010
K = 011
P = 00
A = 10
R = 11

Ambillah satu-persatu bit hasil pemampatan mulai dari kiri, jika bit tersebut termasuk dalam daftar kode, lakukan pengembalian, jika tidak ambil kembali bit selanjutnya dan jumlahkan bit tersebut. Bit pertama dari hasil pemampatan di atas adalah 0, karena 0 tidak termasuk dalam daftar kode kita ambil lagi bit kedua yaitu 0, lalu digabungkan menjadi 00, jika kita lihat daftar kode 00 adalah kode dari karakter P.

Selanjutnya bit ketiga diambil yaitu 0, karena 0 tidak terdapat dalam daftar kode, kita ambil lagi bit keempat yaitu 1 dan kita gabungkan menjadi 01. 01 juga tidak terdapat dalam daftar, jadi kita ambil kembali bit selanjutnya yaitu 0 dan digabungkan menjadi 010. 010 terdapat dalam daftar kode yaitu karakter E. Demikian selanjutnya dikerjakan sampai bit terakhir sehingga akan didapatkan hasil pengembalian yaitu PERKARA.

Supaya lebih jelas kita ambil contoh lain sebuah file yang berisi karakter-karakter "TERTUTUP".

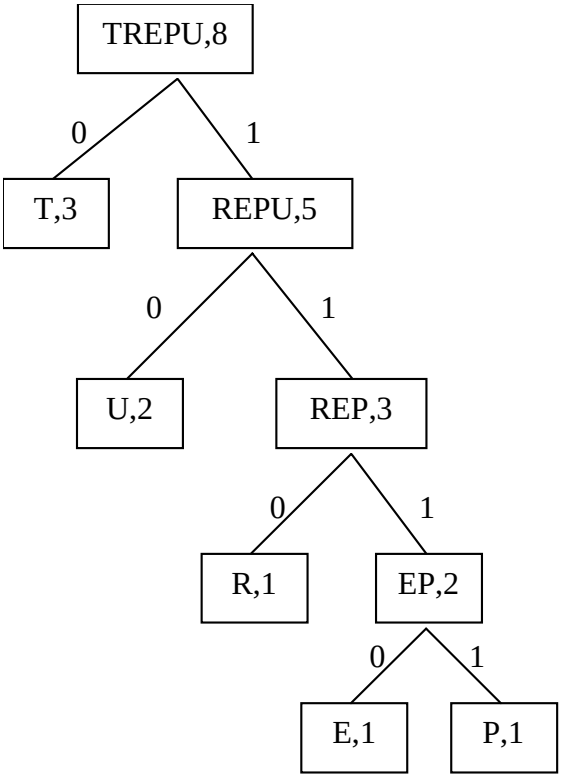
Pertama-tama kita hitung frekuensi kemunculan masing-masing karakter, jika kita hitung ternyata T muncul sebanyak 3 kali, E sebanyak 1 kali, R sebanyak 1 kali, U sebanyak 2 kali dan P sebanyak 1 kali. Jika disusun dari yang kecil (jika sama urutkan berdasarkan kode ASCII):

E = 1
P = 1
R = 1
U = 2
T = 3

Selanjutnya buatlah node masing-masing karakter beserta frekuensinya sebagai berikut :

E,1	P,1	R,1	U,2	T,3

Setelah dilakukan pembuatan pohon *Huffman* seperti yang sudah pernah kita lakukan, hasilnya akan menjadi seperti gambar berikut ini :



Dari pohon Huffman di atas, kita bisa membuat daftar kode untuk masing-masing karakter seperti berikut ini :

E = 1110
P = 1111
R = 110
U = 10
T = 0

Dengan kode ini file pemampatan menjadi berisi :
011101100100101111 = 18 bit
Bandingkan file aslinya yang berukuran 64 bit.

Sekarang coba anda kerjakan sendiri untuk mendapatkan file hasil pemampatan dengan algoritma *Huffman* dari file yang berisi karakter-karakter "PEMAMPATAN". Kerjakanlah dengan tahap-tahap berikut :

1. Hitung frekuensi kemunculan masing-masing karakter.
2. Susun berdasakan jumlah frekuensi dari yang kecil, jika sama lihat kode ASCII-nya.
3. Gambarkan pohon *Huffman* dari karakter-karakter tersebut.
4. Buat daftar Kode *Huffman* dari pohon *Huffman*.
5. Kodekan setiap karakter dalam file asli berdasarkan Kode *Huffman* secara berurutan.

Jika anda lakukan dengan benar, maka daftar kode yang didapatkan adalah :

E = 1010
N = 1011
T = 100
M = 00
P = 01
A = 11

Sehingga file pemampatannya menjadi :

011010001100011110011101 1 = 25 bit.

Coba anda lakukan pengembalian ke file aslinya, apakah menghasilkan file yang sama baik dalam jumlah bit dan isinya ?

Jika anda melakukan pemerograman dengan menggunakan algoritma *Huffman*, jangan lupa menyertakan daftar kodenya dalam file pemampatan, selain itu jumlah byte juga harus disertakan sebagai acuan apakah file pengembalian sama dengan file aslinya. Contoh pemerogramannya lebih jelas dapat dipelajari pada BAB V.

BAB V

IMPLEMENTASI PROGRAM

Program yang akan dijelaskan disini adalah sebuah program sederhana untuk memapatkan file dengan algoritma *Run-Length*, *Half-Byte*, dan *Huffman.* Ketiga algoritma dikemas dalam satu program dimana pemakai dapat memilih algoritma yang diinginkan, selanjutnya program juga dapat mengembalikan file pemampatan ke file aslinya.

Implementasi program untuk masing-masing algoritma dapat dijelaskan sebagai berikut :

1. Algoritma *Run-Length*

   File hasil pemampatan dengan algoritma *Run-Length* harus ditandai pada awal datanya sehingga sewaktu pengembalian ke file asli dapat dikenali apakah file tersebut benar merupakan hasil pemampatan dengan algoritma ini.

   Pada program ini format pengenal file tersebut ditulis pada byte pertama, kedua dan ketiga dengan karakter R, U, dan N. Pembaca dapat mengganti format tersebut dengan karakter lain yang diinginkan, demikian juga dengan jumlahnya.

   Karakter berikutnya (keempat) berisi karakter bit penanda yang telah ditentukan dengan mencari karakter dengan frekuensi kemunculan terkecil. Jika misalnya pada

suatu file bit penandanya adalah X, maka 4 byte pertama isi file pemampatan adalah :

	R	U	N	X	. . .
Bit ke-	1	2	3	4	5 . . .

Karakter kelima dan seterusnya berisi hasil pemampatan dengan algoritma *Run-Length* seperti yang telah dijelaskan pada bab sebelumnya.

2. Algoritma *Half-Byte*

Seperti pada algoritma *Run-Length*, file hasil pemampatan dengan algoritma *Half-Byte* harus ditandai pada awal datanya sehingga sewaktu pengembalian ke file asli dapat dikenali apakah file tersebut benar merupakan hasil pemampatan dengan algoritma ini.

Pada program ini format pengenal file tersebut ditulis pada byte pertama, kedua dan ketiga dengan karakter H, A, dan L. Pembaca dapat mengganti format tersebut dengan karakter lain yang diinginkan, demikian juga dengan jumlahnya.

Karakter berikutnya (keempat) berisi karakter bit penanda yang telah ditentukan dengan mencari karakter dengan frekuensi kemunculan terkecil. Jika misalnya pada suatu file bit penandanya adalah Q, maka 4 byte pertama isi file pemampatan adalah :

	H	A	L	Q	. . .
Bit ke-	1	2	3	4	5 . . .

Karakter kelima dan seterusnya berisi hasil pemampatan dengan algoritma *Half-Byte* seperti yang telah dijelaskan pada bab sebelumnya.

3. Algoritma *Huffman*

Seperti pada kedua algoritma sebelumnya, file hasil pemampatan dengan algoritma *Huffman* harus ditandai pada awal datanya sehingga sewaktu pengembalian ke file asli dapat dikenali apakah file tersebut benar merupakan hasil pemampatan dengan algoritma ini.

Pada program ini format pengenal file tersebut ditulis pada byte pertama, kedua dan ketiga dengan karakter H, U, dan F. Lagi-lagi anda dapat mengganti format tersebut dengan karakter lain yang diinginkan, demikian juga dengan jumlahnya.

Karakter keempat, kelima dan keenam berisi informasi ukuran file asli dalam byte, 3 karakter ini dapat berisi maksimal FFFFFF H atau 16.777.215 byte. Karakter ketujuh berisi informasi jumlah karakter yang memiliki kode *Huffman* atau dengan kata lain jumlah karakter yang frekuensi kemunculannya pada file asli lebih dari nol, jumlah tersebut dikurangi satu dan hasilnya disimpan pada karakter ke tujuh pada file pemampatan

Misalnya suatu file dengan ukuran 3.000 byte dan seluruh karakter ASCII terdapat pada file tersebut, jadi :
Karakter 1-3 : HUF
Karena 3.000 = BB8 H, maka :
Karakter 4-6 : 000B88
Karena seluruh karakter ASCII terdapat pada file tersebut, jadi jumlah karakter yang memiliki kode *Huffman* adalah 256 buah, 256-1=255 = FF Hmaka :
Karakter 7 : FF

Maka 7 byte pertama isi file pemampatan adalah :

	H	U	F	Chr (00H)	Chr (0BH)	Chr (88H)	Chr (FFH)	. . .
Bit ke-	1	2	3	4	5	6	7	8 . . .

Mulai dari karakter ke delapan berisi daftar kode *Huffman* berturut-turut karakter (1 byte), kode *Huffman* (2 byte) dan panjang kode *Huffman* (1 byte). 4 byte berturutan ini diulang untuk seluruh karakter yang dikodekan.

Selanjutnya file diisi hasil pemampatan dengan algoritma *Huffman* seperti yang telah dijelaskan pada bab sebelumnya.

Agar lebih jelas kita ulangi contoh sebelumnya, yaitu file yang berisi karakter "PERKARA". Jika file ini dimampatkan dengan metode *Huffman*, maka file hasil pemampatan akan kita dapatkan sebagai berikut :

Karakter 1-3 : HUF = 48 55 46 H

Karena file tersebut berukuran 7 byte, jadi :

Karakter 4-6 : 00 00 07 H.

Daftar kode Huffman telah kita cari pada bab sebelumnya yaitu :

E = 010
K = 011
P = 00
A = 10
R = 11

Karena jumlah karakter yang memiliki kode *Huffman* ada 5 buah, maka 5-1=4.

Karakter 7 : 04 H.

Karakter 8 : karakter E = 45 H.

Karena kode *Huffman* dari karakter E adalah 010, sedangkan tempat yang disediakan sebanyak 2 byte maka karakter 9 dan 10 menjadi 00000000 00000010 B.

Karakter 9 : 00000000 B = 00 H.

Karakter 10 : 00000010 B = 02 H.

Panjang kode *Huffman* dari karakter E adalah 3 bit (010), jadi :

Karakter 11: 03H.

Cara tersebut di atas diulang untuk karakter K, P, A dan R sehingga didapat :
Karakter 12: karakter K = 4B H.
Karakter 13: 00000000 B = 00 H.
Karakter 14: 00000011 B = 03 H.
Karakter 15: 03H.
Karakter 16: karakter P = 50 H.
Karakter 17: 00000000 B = 00 H.
Karakter 18: 00000000 B = 00 H.
Karakter 19: 02H.
Karakter 20: karakter A = 41 H.
Karakter 21: 00000000 B = 00 H.
Karakter 22: 00000010 B = 02 H.
Karakter 23: 02H.
Karakter 24: karakter R = 52 H.
Karakter 25: 00000000 B = 00 H.
Karakter 26: 00000011 B = 03 H.
Karakter 27: 02H.
Selanjutnya pemampatan karakter-karakter "PERKARA" adalah : 0001011011101110. Jika kita potong-potong menjadi 8 bit tiap bagian akan menjadi :
00010110 = 16 H.
11101110 = EE H.
Jadi :
Karakter 28 : 16 H.
Karakter 29 : EE H.

File hasil pemampatan akan menjadi berukuran 29 byte yang dalam heksadesimal berisi :

48 55 46 00 00 07 04 45 00 02 03 4B 00 03 03 50 00 00 02 41 00 02 02 52 00 03 02 16 EE

Jika kita bandingkan dengan file aslinya yang berukuran 7 byte, hasil ini bukan menjadi lebih kecil, tapi malah

menambah byte sebesar 29-7=22 byte. Hal ini wajar dalam file yang berukuran sangat kecil seperti ini, tapi dalam file-file yang berukuran besar, algoritma *Huffman* ini sangat efektif.

Format yang digunakan pada program ini merupakan format sederhana untuk program pemampatan file, anda dapat memodifikasinya dengan format sendiri atau menambahkan informasi-informasi lainnya, misalnya saja dengan menambahkan informasi nama file asli sehingga pada saat pengembalian ke file aslinya, nama filenya sudah disiapkan.

Program ini dibuat dengan program Borland Delphi versi 5 yang berorientasi pada objek (Object Orientation), mengingat saat ini hampir semua program sudah menggunakan windows sebagai base-nya. Bagi anda yang sudah terbiasa dengan program konvensional (under DOS), tidak akan menjadi masalah karena kami akan menjelaskan langkah demi langkah pembuatan program ini.

1. Buatlah file baru dengan meng-klik File pada menu utama kemudian klik New Application.
2. Ganti Caption dari Form dengan Pemampatan File dan Name dengan FormUtama pada object properties.
3. Tambahkan 5 buah Speed Button pada FormUtama masing-masing :
   a. Name : SButOpen
      Caption : Open
   b. Name : SButSave
      Caption : Save
      Enabled : False
   c. Name : SButSaveOrig
      Caption : Save to Original File
      Enabled : False
      Visible : False
   d. Name : SButDecompress

Caption : Decompress
Enabled : False
Visible : False

e. Name : SButExit
Caption : Exit

4. Tambahkan sebuah GroupBox pada form utama dengan object properties :
Name : GroupMetode
Caption : Kompres File
Enabled: False

5. Di dalam GroupMetode tambahkan 3 buaf Speed Button masing-masing :
   a. Name : SButRunLength
   Caption : Run Length
   b. Name : SButHalfByte
   Caption : Half Byte
   c. Name : SButHuffman
   Caption : Huffman

6. Tambahkan 2 buah Radio Button pada FormUtama masing-masing :
   a. Name : RadioCompress
   Caption : Compress
   Checked : True
   b. Name : RadioDecompress
   Caption : Decompress

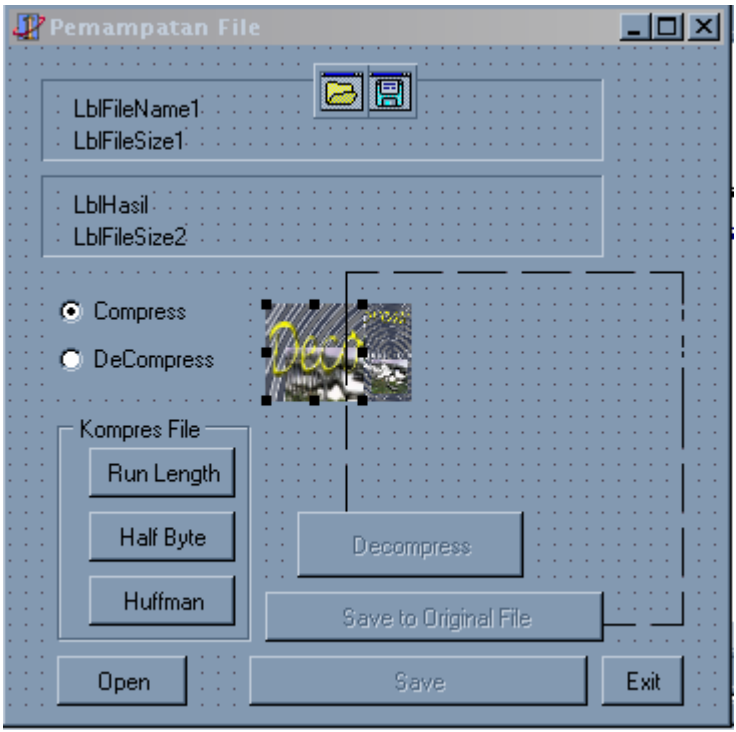
7. Tambahkan 4 buah Label pada FormUtama masing-masing :
   a. Name : LblFileName1
   b. Name : LblFileSize1
   c. Name : LblHasil
   d. Name : LblFileSize2

8. Tambahkan sebuah OpenDialog dan sebuah SaveDialog pada FormUtama.

9. Tambahkan 3 buah Image pada FormUtama masing-masing :
   a. Name : ImageTampil
      Stretch : True
   b. Name : Image1
      Picture : (Pilih sendiri gambar untuk compress)
   c. Name : Image2
      Visible : False
      Picture : (Pilih sendiri gambar untuk decompress)

10. Tambahkan Bevel untuk memperindah tampilan

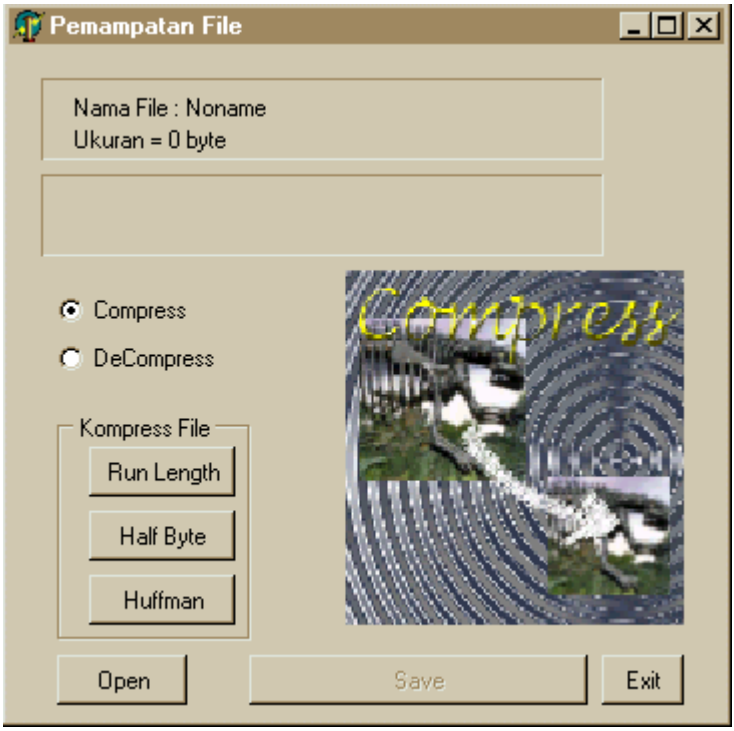
Setelah langkah-langkah tersebut dilakukan, kita akan mendapatkan sebuah form seperti gambar berikut ini, anda dapat menyusun sendiri tata letak tampilan program anda:



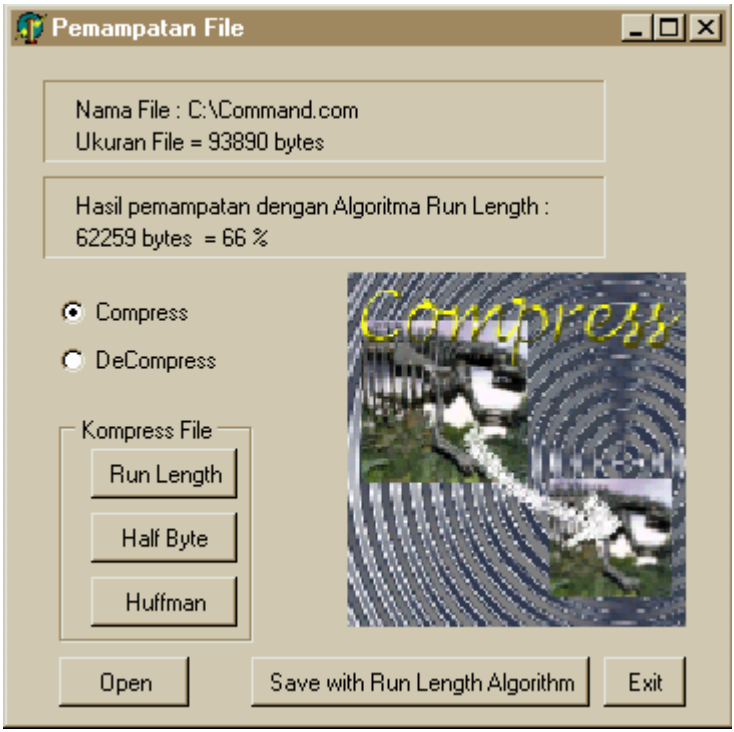
11. Ketik program seperti pada lampiran, kemudian simpan dalam file "pemampatan.pas".
12. Simpan Project dengan nama "FileCompression".
13. Program dapat anda jalankan lewat program Delphi atau di-compile menjadi program tersendiri.

PENJELASAN PROGRAM

Jika anda menjalankan program tersebut, tampilan pertama akan terlihat seperi gambar berikut ini :



Jika anda akan memampatkan suatu file, panggil file tersebut dengan mengklik tombol "Open" sehingga akan tampil o*pen dialog* dan anda dapat memilih file yang akan dimampatkan. Sebagai contoh file yang akan dimampatkan adalah "c:\command.com", anda dapat memilih salah satu algoritma pemampatan file dengan meng-klik salah satu tombol di dalam "Kompress File", pada contoh berikut jika dipilih algoritma *Run-Length*, maka akan tampil seperti gambar berikut :



Pada *bevel* pertama diinformasikan nama file asli serta ukurannya, disana tertulis ukuran file = 93890 bytes, kemudian pada *bevel* kedua tertulis ukuran file hasil pemampatan dengan algoritma *Run-Length*, yaitu sebesar 62259 bytes = 66 %. Hal ini berarti file tersebut dapat dimampatkan sebesar 66 % dengan metode yang dipilih.

Hasil pemampatan ini masih disimpan dalam buffer memori, bila anda akan menyimpannya ke dalam file, silahkan tekan tombol "Save with Run Length Algorithm", maka akan ditamplikan s*ave dialog* dan anda dapat menulis nama file tempat anda menyimpan hasil pemampatan.

Sebelum anda menyimpannya ke dalam file anda dapat mencoba melihat hasil pemampatan dengan algoritma lain

dengan menekan tombol di dalam "Kompress File", misalnya di-klik tombol *Huffman*, maka akan tampak pada pada *bevel* kedua tertulis ukuran file hasil pemampatan dengan algoritma *Huffman*, yaitu sebesar 66803 bytes = 71 %. Hal ini berarti file tersebut dapat dimampatkan sebesar 71 % dengan algoritma *Huffman*. Jika anda mencoba memampatkan file tersebut dengan algoritma *Half-Byte* maka akan didapatkan hasil sebesar 77891 bytes = 82 %.

Terlihat bahwa dalam kasus file ini algoritma yang paling baik untuk memampatkannya adalah algoritma *Run-Length* yaitu sebesar 66%, diikuti algoritma *Huffman* sebesar 71% dan *Half-bye* sebesar 82 %. Namun demikian tidak berarti algoritma *Run-Length* adalah algoritma terbaik untuk file-file lain. Sebagai contoh kami mencoba memampatkan beberapa file dari berbagai tipe file dan hasilnya dapat dilihat seperti pada table berikut ini :

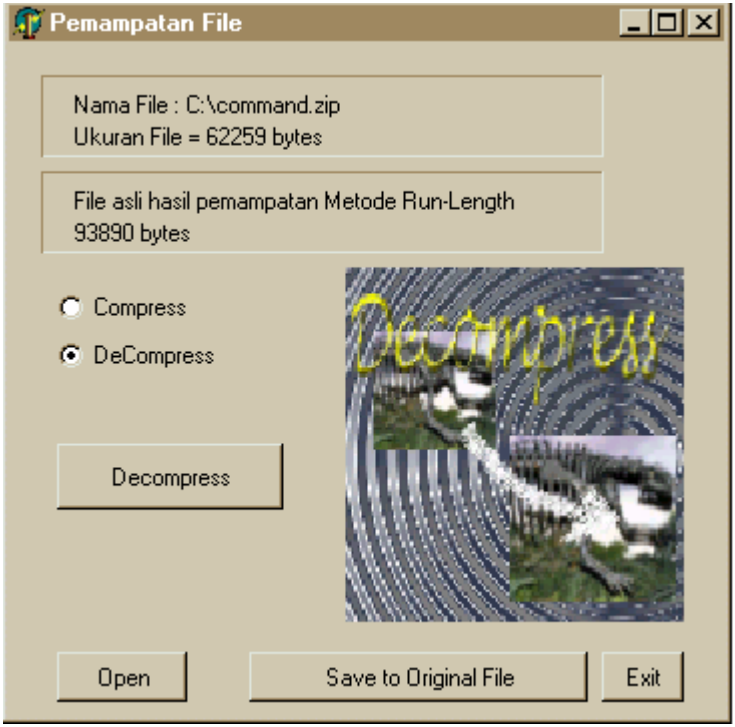
Nama file	Ukuran file asli (bytes)	Ukuran File Hasil Pemampatan Dengan Algoritma : (bytes)		
		Run-Length	Half-Byte	Huffman
suratkuasa.doc	104448	65134 (62 %)	84581 (80 %)	49277 (47 %)
elevator.cdr	193750	190354 (98 %)	192471 (99 %)	194056 (100 %)
fifa2000.exe	1605632	1514726 (94 %)	1549043 (96 %)	1305473 (81 %)
bunga.bmp	360054	195691 (54 %)	278159 (77 %)	70890 (19 %)
Event.dat	13195	1112 (8 %)	7067 (53 %)	2427 (18 %)
Readme.txt	10756	8717 (81 %)	6171 (57 %)	6277 (58 %)

File suratkuasa.doc adalah file dokumen dari hasil pengetikan dengan menggunakan Microsoft office 2000,

setelah dimampatkan ternyata paling kecil hasilnya adalah algoritma *Huffman*, file elevator.cdr merupakan file dokumen dari perogram Corel Draw 9, setelah dimampatkan ternyata paling kecil hasilnya adalah algoritma *Run-Length*, pada file ini ketiga algoritma hanya dapat memampatkannya sebanyak 1-2 %, hal ini karena pada data tersebut sudah dimampatkan oleh program Corel Draw 9, untuk memampatkan file yang sudah dimampatkan akan dibahas pada bab selanjutnya. Berikutnya file fifa2000.exe merupakan suatu program eksekusi, hasil pemampatannya lebih baik jika menggunakan algoritma *Huffman*, file bunga.bmp merupakan file berupa data gambar yang setelah dimampatkan ternyata paling kecil hasilnya adalah dengan menggunakan algoritma *Huffman*, file event.dat merupakan file berupa data text yang setelah dimampatkan ternyata paling kecil hasilnya adalah dengan menggunakan algoritma *Run-length*, dan yang terakhir file readme.txt yang merupakan file text ternyata lebih efektif jika digunakan algoritma *Half-Byte*.

Dari beberapa data di atas menunjukkan bahwa algoritma *Huffman* cukup efektif untuk berbagai macam jenis file, meskipun ada beberapa file yang hasilnya pemampatannya lebih besar dari yang dihasilkan oleh algoritma *Run-Length*. Tapi yang terpenting disini dapat disimpulkan bahwa tidak ada algoritma yang paling efektif untuk setiap file karena hasil pemampatan setiap algoritma tergantung dari isi file yang akan dimampatkannya itu.

Untuk mengembalikan file pemampatan ke file aslinya, anda harus mengubah mode *Compress* menjadi *Decompress* dengan mengklik *Radio Buton Decompress*, setelah itu anda dapat memanggil file yang akan dikembalikan ke file aslinya dengan mengklik tombol *Open*. Setelah anda memanggil file lewat *Open Dialog*, maka anda dapat mengembalikan file tersebut ke file aslinya dengan menekan tombol Decompress, selanjutnya akan program akan menjadi seperti gambar berikut :



Tidak seperti saat memampatkan, kita tiak perlu memilih algoritma untuk pengembalian ke file asli sebab program akan mendeteksi secara otomatis algoritma yang digunakan pada file tersebut. Anda tentu ingat pada saat pemampatan, kita telah menuliskan tiga karakter pertama sebagai karakter identifikasi untuk file pemampatan yaitu "RUN" untuk algoritma *Run-Length*, "HAL" untuk algoritma *Half-Byte*, dan "HUF" untuk algoritma *Huffman*. Andaikan anda memanggil file yang bukan hasil dari pemampatan ataupun hasil pemampatan dari program lain, maka program akan menolak pemanggilan tersebut.

Hasil pemampatan masih disimpan pada buffer memori, jika anda ingin menyimpannya pada media penyimpanan, anda tinggal menekan tombol *Save to Original File*, selanjutnya program akan menampilka *save dialog*, dan anda dapat menuliskan nama filenya.

BAB VI

MEMAMPATKAN FILE YANG SUDAH DIMAMPATKAN

Setelah mempelajari satu-persatu algoritma pemampatan file, sekarang saatnya kita akan membahas bagaimana jika sebuah file yang sudah dimampatkan kita mampatkan lagi, dan jika memungkinkan file yang sudah dua kali dimampatkan kita mampatkan lagi dan seterusnya, apa jadinya ?
Ada dua hal yang pelu dicari jawabannya yaitu :

1. Bagaimana pengaruh pemampatan terhadap file yang telah dimampatkan, kemudian dimampatkan lagi dengan metode yang sama atau yang berbeda ?
2. Jika suatu file dimampatkan 3 kali dengan algoritma yang berbeda, bagaimana pengaruh urutan penggunaan algoritmanya ?

Untuk mecari jawaban persoalan pertama, penulis telah mengadakan percobaan terhadap beberapa file yang telah dimampatkan, kemudian dimampatkan lagi dengan metode yang berbeda. Metode percobaanya dapat dilihat pada table berikut ini:

Algoritma pemampatan	Algortima pemampatan kedua		
Run-Length	*Run-Length*	*Half-Byte*	*Huffman*
Half-Byte	*Run-Length*	*Half-Byte*	*Huffman*
Huffman	*Run-Length*	*Half-Byte*	*Huffman*

Dari metode percobaan di atas akan dapat kita lihat bagaimana hasilnya jika file yang telah dimampatkan dengan suatu algoritma dimampatkan lagi dengan algoritma yang sama, dan bagaima pula hasilnya jika dimampatkan lagi dengan algoritma yang lain.

Percobaan yang dilakukan terhadap beberapa file menunjukkan hasil seperti tabel-tabel berikut ini :

File Asli	Algoritma *Run-Length*	Algortima pemampatan kedua		
		RunLength	*Half-Byte*	*Huffman*
Command.com 93890	62259	62276 (100 %)	62128 (99 %)	57139 (91 %)
Kop-fmpk.doc 41984	28068	28106 (100 %)	27600 (98 %)	24758 (88 %)
Readme.txt 10756	8717	8721 (100 %)	5782 (66 %)	5403 (61 %)

Dari tabel di atas dapat kita lihat bahwa file yang telah dimampatkan dengan algoritma *Run-Length* tidak dapat dimampatkan lagi dengan algoritma yang sama, pada table di atas ketiga file yang dicoba menunjukkan ukurannya malah menjadi lebih besar, hal ini disebabkan penambahan karakter identifikasi serta pengulangan pada karakter yang serupa dengan bit penanda.

Jika dimampatkan lagi dengan algoritma *Half-Byte*, ukuran filenya masih bisa diperkecil, walaupun sedikit pada file pertama dan kedua, sedangkan pada file ketiga hasilnya

cukup baik, yaitu dapat memapatkan sebesar 66% dari file hasil pemampatan dengan algoritma *Run-Length.*

Jika dimampatkan lagi dengan algoritma *Huffman*, ukuran filenya tampak masih bisa diperkecil, hal itu terlihat pada file pertama, kedua dan ketiga, yaitu dapat memampatkan sebesar 91%, 88% dan 61% dari file-file hasil pemampatan dengan algoritma *Run-Length.*

File Asli	Algoritma *Half-Byte*	Algortima pemampatan kedua		
		RunLength	*Half-Byte*	*Huffman*
Command.com 93890	77891	62287 (79 %)	70145 (90 %)	62648 (80 %)
Kop-fmpk.doc 41984	34674	28449 (82 %)	31878 (91 %)	26382 (76 %)
Readme.txt 10756	6171	4453 (72 %)	5490 (88 %)	3921 (63 %)

Dari tabel di atas dapat kita lihat bahwa file yang telah dimampatkan dengan algoritma *Half-Byte* ternyata masih bisa dimampatkan lagi dengan algoritma yang sama, pada tabel di atas ketiga file yang dicoba menunjukkan ukurannya menjadi lebih kecil untuk ketiga file yang dicoba yaitu sebesar 79%, 82% dan 72%.

Jika dimampatkan lagi dengan algoritma *Run-Length*, ukuran filenya tampak masih bisa diperkecil, hal itu terlihat pada file pertama, kedua dan ketiga, yaitu dapat memampatkan sebesar 90%, 91% dan 88% dari file-file hasil pemampatan dengan algoritma *Half-Byte.*

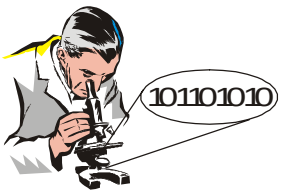
Jika dimampatkan lagi dengan algoritma *Huffman*, ukuran filenya tampak masih bisa diperkecil lebih baik dari algoritma *Run-Length*, hal itu terlihat pada file pertama, kedua dan ketiga, yaitu dapat memampatkan sebesar 80%, 76% dan 63% dari file-file hasil pemampatan dengan algoritma *Half-Byte.*

File Asli	Algoritma *Huffman*	Algortima pemampatan kedua		
		RunLength	*Half-Byte*	*Huffman*
Command.com 93890	66803	59346 (88 %)	63098 (94 %)	64175 (96 %)
Kop-fmpk.doc 41984	25184	23610 (93 %)	24410 (97 %)	25374 (100 %)
Readme.txt 10756	6171	5935 (94 %)	6164 (98 %)	6365 (101 %)

Dari tabel di atas dapat kita lihat bahwa file yang telah dimampatkan dengan algoritma *Huffman* tidak dapat dimampatkan lagi dengan algoritma yang sama, kecuali pada file pertama dengan pemampatan kecil sebesar 96 %.

Jika dimampatkan lagi dengan algoritma *Run-Length*, ukuran filenya tampak masih bisa diperkecil walaupun hanya sedikit, hal itu terlihat pada file pertama, kedua dan ketiga, yaitu dapat memampatkan sebesar 88%, 93% dan 94% dari file-file hasil pemampatan dengan algoritma *Huffman*.

Jika dimampatkan lagi dengan algoritma *Half-Byte*, ukuran filenya tampak masih bisa diperkecil walaupun hanya sedikit, hal itu terlihat pada file pertama, kedua dan ketiga, yaitu dapat memampatkan sebesar 94%, 97% dan 98% dari file-file hasil pemampatan dengan algoritma *Huffman*.

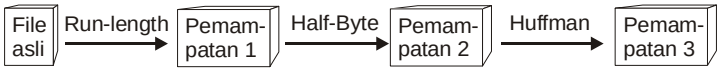


Dari hasil yang didapat pada percobaan di atas, dapat kita simpulkan bahwa :

1. Pada umumnya file yang sudah dimampatkan dengan suatu algoritma sulit untuk dimampatkan lagi dengan algoritma yang sama kecuali pada algoritma *Half-Byte.*
2. File yang sudah dimampatkan dengan suatu algoritma ternyata masih bisa dimampatkan lagi dengan algoritma yang berbeda, kecuali algoritma yang digunakan pada pemampatan pertama adalah algoritma *Huffman* yang hanya menghasilkan perubahan kecil.

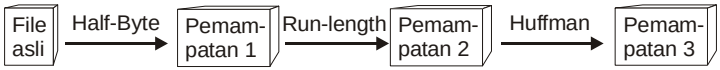
Untuk mecari jawaban persoalan kedua, penulis telah mengadakan percobaan terhadap beberapa file yang telah dimampatkan, kemudian dimampatkan lagi sebanyak dua kali dengan metode yang berbeda. Metode percobaanya dapat dilihat pada gambar berikut ini:

Metode 1

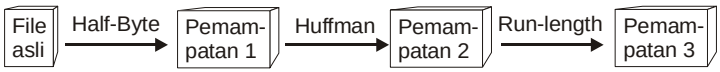


Metode 2

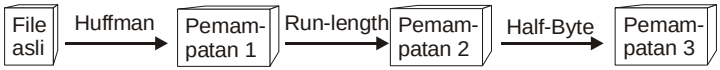
Metode 3



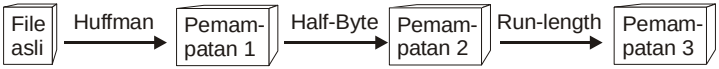
Metode 4



Metode 5



Metode 6



Dari metode percobaan di atas akan dapat kita lihat bagaimana hasilnya jika file yang telah dimampatkan dengan suatu algoritma dimampatkan lagi sebanyak 2 kali dengan algoritma yang lain, dan urutan mana yang menghasilkan hasil yang paling optimal.

Percobaan yang dilakukan terhadap beberapa file menunjukkan hasil seperti tabel-tabel berikut ini :

File Asli	Metode					
	1	2	3	4	5	6
Command.com 93890	57185 (61%)	57272 (61%)	57290 (61%)	59346 (63%)	59424 (63%)	59434 (63%)
Kop-fmpk.doc 41984	22473 (54%)	22446 (54%)	22664 (54%)	23552 (56%)	23644 (56%)	23667 (56%)
Readme.txt 10756	3920 (36%)	5388 (50%)	3120 (29%)	3783 (35%)	5937 (55%)	6009 (56%)
Bunga.bmp 360054	37855 (11%)	29378 (8%)	38637 (11%)	27846 (8%)	52056 (14%)	52851 (15%)
Kuisioner.xls 16896	4854 (29%)	4708 (28%)	4880 (29%)	4852 (29%)	4834 (29%)	4841 (29%)

Jika kita urut dari yang terbaik untuk masing-masing file, data diatas menjadi :

1 – 2 – 3 – 4 – 5 – 6
2 – 1 – 3 – 4 – 5 – 6
3 – 4 – 5 – 1 – 2 – 6
4 – 2 – 1 – 3 – 5 – 6
2 – 5 – 6 – 4 – 1 – 3

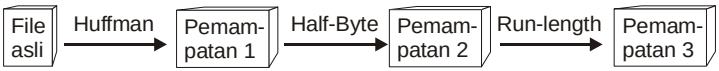
Jika kita beri nilai 5 untuk urutan pertama, 4 untuk urutan kedua, 3 untuk urutan ketiga, 2 untuk urutan keempat,

1 untuk urutan kelima dan 0 untuk urutan keenam, maka nilai yang diperoleh masing-masing metode adalah :

Metode 1 = 5+4+2+3+1 = 15
Metode 2 = 4+5+1+4+5 = 19
Metode 3 = 3+3+5+2+0 = 13
Metode 4 = 2+2+4+5+2 = 15
Metode 5 = 1+1+3+1+4 = 10
Metode 6 = 0+0+0+0+3 = 3

Dari percobaan kecil diatas tampak bahwa metode yang paling baik adalah metode kedua yaitu :

dan metode yang tidak direkomendasikan adalah metode keenam, yaitu:



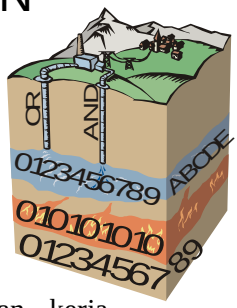
Hasil percobaan diatas hanya digunakan sebagai ilustrasi, jika anda ingin mendapatkan hasil yang lebih baik, anda dapat mengadakan percobaan sendiri dengan menambah jumlah contoh file.

Contoh program pada buku ini hanya menggunakan satu kali pemampatan dengan satu algoritma, anda dapat mengembangkannya dengan menggabungkan ketiga algoritma untuk memampatkan sebuah file. Urutan algoritma yang digunakan dapat ditentukan secara tetap atau dengan mencobakan ke-enam cara pengurutan untuk melihat hasil yang paling optimal, baru kemudian dilaksanakan pemampatan, cara terakhir ini akan menghasilkan

pemampatan yang paling baik, tapi tentu saja akan menambah waktu eksekusi program.

BAB VII

DASAR BILANGAN DAN OPERASI LOGIKA



Setiap bagian di dalam sebuah komputer dapat disederhanakan menjadi sebuah jaringan kerja sakelar-sakelar, yang setiap saatnya dalam keadaan hidup atau mati. Konsep yang sederhana ini pada akhirnya akan menghasilkan kekuatan dan keanekaragaman penggunaan sebuah komputer. Bilangan '0' dinyatakan sebagai sebuah sakelar yang mati dan bilangan '1' dinyatakan sebagai sebuah sakelar yang hidup.

Komputer hanya terbatas dapat menghitung angka 0 dan 1, bagaimana dia dapat bekerja ?, tapi juga harus kita ingat bukankah manusia yang mengenal angka 0 sampai 9 dapat bekerja dengan angka puluhan, ratusan, ribuan bahkan sampai tak terhingga. Konsepnya adalah penggabungan bilangan tersebut, jadi walaupun komputer hanya mengenal angka 0 dan 1 tapi jika dua angka tersebut digabungkan akan memiliki komposisi yang juga tidak terhingga.

Sistem bilangan manusia disebut sebagai sistem bilangan desimalyang dipergunakan di seluruh dunia. Sistem hitungan pada komputer disebut sistem biner yang hanya terdiri dari dua angka yang berbeda yaitu 0 dan 1. Untuk menyatakan bilangan yang lebih besar dari 1, kita mengikuti konsep yang dilakukan pada sistem bilangan desimal. Kalau untuk

menyatakan "sepuluh" pada sistem bilangan desimal menggabungkan angka 1 dan 0, pada sistem biner untuk menyatakan "dua" juga menggabungkan angka 1 dan 0. Untuk bialngan lainnya juga menggunakan konsep yang sama, jadi untuk bilangan 0 sampai 10 menjadi seperti berikut ini :

	Desimal	Biner
Nol	0	0
Satu	1	1
Dua	2	10
Tiga	3	11
Empat	4	100
Lima	5	101
Enam	6	110
Tujuh	7	111
Delapan	8	1000
Sembilan	9	1001
Sepuluh	10	1010

Dengan cara yang sama, sistem biner dapat menyatakan bilangan sampai tidak terhingga.

Dalam penulisan biasanya bilangan biner diberi lambang 2 dengan posisi agak ke bawah, jadi jika ditulis 100_2, berarti bilangan tersebut bukan "seratus", tapi "empat", karena bilangan tersebut adalah bilangan biner. Cara penulisan lain adalah : 100B atau 100_B. Sedangkan pada bilangan desimal biasanya tidak perlu diberi tanda apa-apa atau kadangkala ditulis dengan 4_{10}, 4D atau 4_D.

Selanjutnya pada buku ini, bilangan desimal tidak akan diberi tanda apa-apa sedangkan bilangan biner akan ditandai dengan "$_2$" diakhirnya.

Sekarang bagaimana caranya untuk mengubah dari bilangan desimal ke bilangan biner atau sebaliknya ?

Konversi Desimal ke Biner

Untuk mengkonversi bilangan Desimal ke Biner dapat kita lihat pada contoh di bawah ini :

Misalnya kita akan mengkonversi angka 36 menjadi bilangan biner.

36 : 2 = 18 sisa 0
18 : 2 = 9 sisa 0
9 : 2 = 4 sisa 1
4 : 2 = 2 sisa 0
2 : 2 = 1 sisa 0
1 : 2 = 0 sisa 1

Mula-mula kita bagi angka yang akan dicari yaitu 36 dengan 2, hasilnya adalah 18 dengan sisa 0, selanjutnya hasil bagi tersebut yaitu 18 dibagi lagi dengan 2, hasilnya adalah 9 dengan sisa 0, demikian seterusnya sampai hasilnya sama dengan 0.

Sekarang kita lihat sisa baginya, urutkan dari bawah ke atas, itulah bilangan binernya. Pada contoh di atas, jika kita urutkan sisa baginya dari bawah ke atas akan menjadi 100100_2.

Kita coba sekali lagi, konversikan bilangan 100 menjadi bilangan biner.

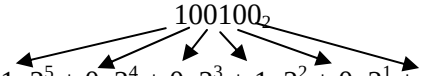
100 : 2 = 50 sisa 0
50 : 2 = 25 sisa 0
25 : 2 = 12 sisa 1
12 : 2 = 6 sisa 0
6 : 2 = 3 sisa 0
3 : 2 = 1 sisa 1
1 : 2 = 0 sisa 1

Jadi $100 = 1100100_2$

Konversi Biner ke Desimal

Untuk mengkonversi bilangan Biner ke Desimal dapat kita lihat pada contoh di bawah ini :

Misalnya kita akan mengkonversi angka 100100_2 menjadi bilangan desimal.



$1x2^5 + 0x2^4 + 0x2^3 + 1x2^2 + 0x2^1 + 0x2^0 =$
$32+0+0+4+0+0$

$= 36$

Angka paling kanan kita kalikan dengan 2^0, angka kedua dari kanan kita kalikan dengan 2^1, angka ketiga dari kanan kita kalikan dengan 2^2, demikian seterusnya sampai angka yang paling kiri. Setelah itu seluruh hasilnya dijumlahkan dan totalnya adalah bilangan desimal dari bilangan biner yang dicari. Pada contoh diatas, 100100_2 sama dengan 36.

Kita coba sekali lagi, konversikan bilangan 1100100_2 menjadi bilangan desimal.

$1x2^6+1x2^5+ 0x2^4+ 0x2^3+1x2^2+ 0x2^1+ 0x2^0=$
$64+32+0+0+4+0+0$

$= 100$

Jadi $1100100_2 = 100$

Selain bilangan desimal dan biner, kita juga mengenal sistem bilangan heksadesimal, bilangan yang memiliki dasar 16 bilangan ini terdiri dari 0 sampai 9 ditambah A, B, C, D, E, dan F untuk menyatakan bilangan 10, 11, 12, 13, 14, dan 15. Jika kita urut dari 0 sampai 15, akan kita dapatkan :

0	0
1	1
2	2
3	3

4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

Dalam penulisan biasanya bilangan heksadesimal diberi lambang 16 dengan posisi agak ke bawah, jadi jika ditulis 10_{16}, berarti bilangan tersebut bukan "sepuluh", tapi "enam belas", karena bilangan tersebut adalah bilangan heksadesimal. Cara penulisan lain adalah : 100H atau 100_H. Selanjutnya pada buku ini, bilangan heksadesimal akan ditandai dengan "$_{16}$" diakhirnya.

Sekarang akan kita bahas bagaimana mengkonversi bilangan desimal ke bilangan heksadesimal dan sebaliknya , juga konversi dari bilangan biner ke heksadesimal dan sebaliknya.

Konversi Desimal ke Heksadesimal

Untuk mengkonversi bilangan desimal ke heksadesimal dapat kita lihat pada contoh di bawah ini :

Misalnya kita akan mengkonversi angka 61719 menjadi bilangan heksadesimal.

61719 : 16 = 3857 sisa 7
3857 : 16 = 241 sisa 1
241 : 16 = 15 sisa 1
15 : 16 = 0 sisa 15 (F)

Mula-mula kita bagi angka yang akan dicari yaitu 61719 dengan 16, hasilnya adalah 3857 dengan sisa 7, selanjutnya hasil bagi tersebut yaitu 3857 dibagi lagi dengan 16, hasilnya adalah 241 dengan sisa 1, demikian seterusnya sampai hasilnya sama dengan 0.

Sekarang kita lihat sisa baginya, jika lebih dari 9 konversikan ke heksadesimal, urutkan dari bawah ke atas, itulah bilangan heksadesimalnya. Pada contoh di atas, jika kita urutkan sisa baginya dari bawah ke atas akan menjadi $F117_{16}$.

Kita coba sekali lagi, konversikan bilangan 100 menjadi bilangan heksadesimal.

100 : 16 = 6 sisa 4 ↑
6 : 16 = 0 sisa 6 |

Jadi $100 = 64_{16}$

Konversi Heksadesimal ke Desimal

Untuk mengkonversi bilangan heksadesimal ke desimal caranya sama dengan konversi dari biner ke desimal, kita lihat pada contoh di bawah ini :

Misalnya kita akan mengkonversi angka $F117_2$ menjadi bilangan desimal.

$F117_{16}$

$15x16^3 + 1x16^2 + 1x16^1 + 7x16^0$ =
61440+256+16+7

= 61719

Angka paling kanan kita kalikan dengan 16^0, angka kedua dari kanan kita kalikan dengan 16^1, angka ketiga dari kanan kita kalikan dengan 16^2, angka tearakhir (F) kita jadikan desimal terlebih dahulu (15) kemudian dikalikan dengan 16^3. Setelah itu seluruh hasilnya dijumlahkan dan

totalnya adalah bilangan desimal dari bilangan biner yang dicari. Pada contoh diatas, $F117_{16}$ sama dengan 61719.

Kita coba sekali lagi, konversikan bilangan $A1_{16}$ menjadi bilangan desimal.

$10x16^1 + 1x16^0 = 160+1$
$= 161$
Jadi $A1_{16} = 161$

Konversi Heksadesimal ke Biner

Untuk mengkonversikan bilangan heksadesimal ke sistem bilangan biner, dapat dilakukan dengan jalan mengkonversi bilangan heksadesimal ke bilangan desimal, selanjutnya hasilnya kita konversikan ke bilangan biner.
Contoh : Konversikan $A1_{16}$ ke bilangan biner.
Pertama kita konversikan ke bilangan desimal seperti yang telah dibahas sebelumnya dan didapatkan $A1_{16} = 161$, selanjutnya kita konversikan hasil tersebut ke bilangan biner, hingga didapat $161 = 10100001_2$.
Jadi $A1_{16} = 10100001_2$

Selain cara di atas, ada cara yang lebih mudah, yaitu dengan mengkonversikan setiap angka dalam bilangan heksadesimal tersebut menjadi 4 angka bilangan biner. Sebelumnya kita akan mengacu pada tabel desima-biner-heksadesimal seperti berikut ini :

desimal	biner	heksadesimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Kita ambil contoh sebelumnya :
Konversikan $A1_{16}$ ke bilangan biner.
Dengan mengacu pada tabel di atas, kita konversikan setiap angka :
A = 1010
1 = 0001
Kemudian kita gabungkan menjadi 10100001.
Jadi $A1_{16} = 10100001_2$.

Contoh lain :
Konversikan $F117_{16}$ ke bilangan biner.
Dengan mengacu pada tabel di atas, kita konversikan setiap angka :
F = 1111
1 = 0001
1 = 0001
7 = 0111

Kita gabungkan menjadi 1111000100010111.
Jadi $F117_{16} = 1111000100010111_2$.

Konversi Biner ke Heksadesimal

Untuk mengkonversikan bilangan biner ke sistem bilangan heksadesimal, dapat dilakukan dengan jalan mengkonversi bilangan biner ke bilangan desimal, selanjutnya hasilnya kita konversikan ke bilangan heksadesimal.

Contoh : Konversikan 10100001_2 ke bilangan heksadesimal.
Pertama kita konversikan ke bilangan desimal seperti yang telah dibahas sebelumnya dan didapatkan $10100001_2 = 161$, selanjutnya kita konversikan hasil tersebut ke bilangan heksadesimal, hingga didapat $161 = A1_{16}$.
Jadi $10100001_2 = A1_{16}$

Selain cara di atas, ada cara yang lebih mudah, yaitu dengan membagi-bagi bilangan tersebut setiap 4 angka kemudian setiap 4 angka tersebut dikonversikan menjadi 1 angka bilangan heksadesimal.

Kita ambil contoh sebelumnya :
Konversikan 10100001_2 ke bilangan heksadesimal.
Pertama kita potong-potong bilangan tersebut atas 4 angka setiap potongnya dari kanan, kemudian konversikan setiap 4 angka tersebut ke bilangan heksadesimal dengan mengacu tabel di atas.

```
1010 | 0001
  A      1
```

Kemudian kita gabungkan menjadi A1.
Jadi $10100001_2 = A1_{16}$

Contoh lain :
Konversikan 11010010101_2 ke bilangan heksadesimal.
Pertama kita bagi empat-empat dari kanan, kemudian dengan mengacu pada tabel di atas, kita konversikan setiap angka.

Karena jumlah angka pada bilangan tersebut sebanyak 11 angka, sisa tiga angka kita tambahkan 0 di sebelah kirinya.

```
  110 | 1001 | 0101
 0110 | 1001 | 0101
    6      9      5
```

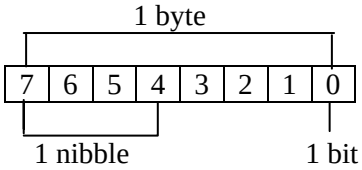
Kita gabungkan menjadi 695.
Jadi $11010010101_2 = 695_{16}$

Bit, Nibble, dan Byte

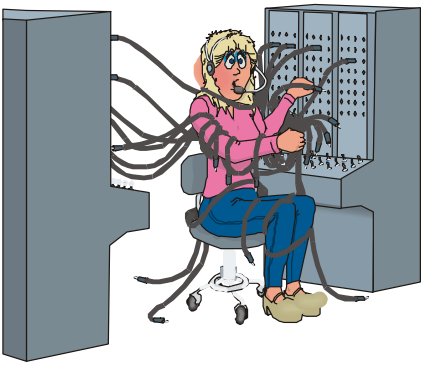
Satu angka biner selalu dinyatakan sebagai satu *bit*, jadi satu bit dapat mempunyai nilai 0 atau 1.

Meskipun bilangan dapat dihasilkan dari sebarang besaran angka biner atau bit, ada besaran bit tertentu yang akan sering anda jumpai, yang paling sering anda akan berurusan dengan kombinasi 8-bit yang biasa dinyatakan sebagai satu *byte.* Dalam dunia komputerbyte adalah selalu satu unit 8-bit, jadi satu byte dapat menyatakan bilangan diantara 0 hingga 255.

Unit dari empat bit kadangkala disebut *nibble,* jadi satu byte terdiri atas dua nibble.



Operasi Logika



Operator NOT

Operator NOT adalah yang paling mudah untuk dimengerti, operator ini hanya mengatakan: pindahkan sakelar ke keadaan sebaliknya, bila sakelar itu hidup membuatnya mati dan bila ia mati, membuatnya hidup. Akibat dari sebuah operator NOT pada sebuah bit adalah ia mengubah 1 menjadi 0 atau 0 menjadi 1.

NOT 0 = 1
NOT 1 = 0
NOT 1011 = 0100

Seringkali satu garis di atas sebuah bilangan digunakan sebagai pengganti NOT.

$\overline{0} = 1$

$\overline{1} = 0$

$\overline{1011} = 0100$

Operator AND

Operator AND menguji apakah dua buah sakelar kedua-duanya dalam keadaan hidup, jadi hasil operator AND akan

1 hanya jika kedua bilangan sama dengan 1. Dengan kata lain jika salah satu bilangan atau kedua-duanya sama dengan 0 maka hasilnya akan menjadi 0.

0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1

Sebuah titik (.) seringkali digunakan sebagai pengganti kata AND.

0 . 0 = 0
0 . 1 = 0
1 . 0 = 0
1 . 1 = 1

Logika operasi AND merupakan bagian yang umum dalam kehidupan kita sehari-hari, misalnya :

"Fahmi akan berbelanja ke pasar jika cuaca baik dan punya uang".

Artinya Fahmi hanya akan berbelanja ke pasar jika kedua keadaan memenuhi yaitu cuaca baik dan punya uang, jika salah satu atau kedua keadaan tersebut tidak memenuhi, maka Fahmi tidak akan berbelanja ke pasar. Misalnya cuaca tidak baik, walaupun Fahmi mempunyai uang, Fahmi tidak akan berbelanja ke pasar, begitu juga sebaliknya walaupun cuaca baik tapi Fahmi tidak mempunyai uang, maka Fahmi tidak akan berbelanja ke pasar, apalagi jika kedua keadaan tersebut tidak memenuhi, cuaca tidak baik dan Fahmi tidak punya uang.

Operator OR

Operator OR menguji apakah dua buah sakelar salah satu atau kedua-duanya dalam keadaan hidup, jadi hasil operator OR akan 1 jika salah satu atau kedua bilangan sama

dengan 1. Dengan kata lain hasilnya akan menjadi 0 hanya jika kedua-duanya sama dengan 0 maka.

0 OR 0 = 0
0 OR 1 = 1
1 OR 0 = 1
1 OR 1 = 1

Sebuah tanda plus (+) seringkali digunakan sebagai pengganti kata OR.

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 1

Logika operasi OR juga merupakan bagian yang umum dalam kehidupan kita sehari-hari, misalnya :

"Adik akan menangis jika dicubit atau dimarahi".

Artinya adik akan menangis jika salah satu atau kedua keadaan memenuhi yaitu dicubit atau dimarahi, jika kedua keadaan tersebut tidak memenuhi, barulah adik tidak akan menangis. Misalnya adik dicubit, walaupun tidak dimarahi adik akan menangis, begitu juga sebaliknya walaupun tidak dicubit tapi dimarahi, maka adik akan menangis. Adik tidak menangis hanya jika dia tidak dicubit juga tidak dimarahi.

Operator XOR

Operator XOR (Exclusive-OR) menguji adanya perbedaan dan perubahan, jika terdapat perbedan antara kedua keadaan maka hasilnya menjadi 1, sebaliknya jika kedua keadaan sama maka hasilnya menjadi 0, XOR dapat didefinisikan sebagai berikut :

0 XOR 0 = 0
0 XOR 1 = 1

1 XOR 0 = 1
1 XOR 1 = 0

Sebuah tanda ⊕ seringkali digunakan sebagai pengganti kata OR.

$0 \oplus 0 = 0$
$0 \oplus 1 = 1$
$1 \oplus 0 = 1$
$1 \oplus 1 = 0$

XOR juga merupakan bagian dari kehidupan kita sehari-hari, ia menunjukkan adanya perbedaan pandangan. Sebagai contoh :

"Persetujuan akan gagal jika kelompok 1 memilih jalan A, sedangkan kelompok 2 memilih jalan B, atau jika kelompok 1 memilih jalan B sedangkan kelompok 2 memilih jalan A".

Persetujuan akan gagal jika kedua kelompok baik kelompok 1 maupun kelompok 2 memilih jalan yang berbeda, apakah kelompok 1 memilih jalan A dan kelompok 2 memilih jalan B, atau kelompok 1 memilih jalan B dan kelompok 2 memilih jalan A. Persetujuan tidak akan gagal jika kedua kelompok memilih jalan yang sama, apakah kedua kelompok sama-sama memilih jalan A atau kedua kelompok sama-sama memilih jalan B.

LAMPIRAN : PEMAMPATAN.PAS

```
unit Pemampatan;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs,
  StdCtrls, Buttons, ExtCtrls;

type
  TFormUtama = class(TForm)
    LblFileName1: TLabel;
    OpenDialog1: TOpenDialog;
    SaveDialog1: TSaveDialog;
    LblFileSize1: TLabel;
    SButExit: TSpeedButton;
    SButOpen: TSpeedButton;
    GroupMetode: TGroupBox;
    SButRunLength: TSpeedButton;
    SButHalfByte: TSpeedButton;
    SButHuffman: TSpeedButton;
    SbutSave: TSpeedButton;
    LblHasil: TLabel;
    LblFileSize2: TLabel;
    RadioDecompress: TRadioButton;
    Image1: TImage;
    Image2: TImage;
    ImageTampil: TImage;
    Bevel1: TBevel;
    Bevel2: TBevel;
    RadioCompress: TRadioButton;
    SButSaveOrig: TSpeedButton;
    SButDeCompress: TSpeedButton;
    procedure FormCreate(Sender: TObject);
    procedure SButExitClick(Sender: TObject);
```

```
    procedure SButOpenClick(Sender: TObject);
    procedure SButRunLengthClick(Sender: TObject);
    procedure SbutSaveClick(Sender: TObject);
    procedure SButHalfByteClick(Sender: TObject);
    procedure SButHuffmanClick(Sender: TObject);
    function BinToDec(Biner:string): LongInt;
    function DecToBin(Desimal:LongInt): string;
    procedure RadioDecompressClick(Sender: TObject);
    procedure RadioCompressClick(Sender: TObject);
    procedure SButDeCompressClick(Sender: TObject);
    procedure SButSaveOrigClick(Sender: TObject);
  private
    { Private declarations }
    FSize,FCSize: LongInt;
    FCLSize: LongInt;
    FCBSize: LongInt;
    FCHSize: LongInt;
    FCSaveSize: LongInt;
    Metode: Integer;
    Buf: array[1..5000000] of Char;
    BufC: array[1..5000000] of Char;
  public
    { Public declarations }
  end;

var
  FormUtama: TFormUtama;

implementation

{$R *.DFM}

procedure TFormUtama.FormCreate(Sender: TObject);
begin
  LblFileName1.caption:='Nama File : Noname';
  LblFileSize1.caption:='Ukuran = 0 byte';
  LblHasil.caption:='';
  LblFileSize2.caption:='';
  ImageTampil.picture:=Image1.picture;
```

```
end;

procedure TFormUtama.SButExitClick(Sender: TObject);
begin
  close;
end;

procedure TFormUtama.SButOpenClick(Sender: TObject);
var
  FromF: file;
  NumRead: Integer;
  Save_Cursor:TCursor;

begin
  if OpenDialog1.Execute then
  begin
    Save_Cursor := Screen.Cursor;
    Screen.Cursor := crHourglass;    { Show hourglass cursor }
    AssignFile(FromF, OpenDialog1.FileName);
    Reset(FromF, 1);           { Record size = 1 }
    Fsize := FileSize(FromF);
     repeat
       BlockRead(FromF, Buf, SizeOf(Buf), NumRead);
     until (NumRead = 0);
     CloseFile(FromF);
    Metode:=0;
    If Buf[1]+Buf[2]+Buf[3]='RUN' then Metode:=1;
    If Buf[1]+Buf[2]+Buf[3]='HAL' then Metode:=2;
    If Buf[1]+Buf[2]+Buf[3]='HUF' then Metode:=3;
    LblFileName1.Caption := 'Nama File :
'+OpenDialog1.FileName;
    LblFileSize1.Caption := 'Ukuran File = '+IntToStr(FSize)+'
bytes';
    Screen.Cursor := Save_Cursor;
    GroupMetode.enabled:=true;
    SButDecompress.enabled:=true;
    LblHasil.caption:='';
    LblFileSize2.caption:='';
  end;
```

```
end;

procedure TFormUtama.SButRunLengthClick(Sender: TObject);
var
   i,ii: LongInt;
   BitPenanda,inbuf,buflong,SameBit,ulang4,uu :integer;
   SigmaMin:LongInt;
   Save_Cursor:TCursor;
   SigmaAscii: Array[0..255] of LongInt;
Begin
  Save_Cursor := Screen.Cursor;
  Screen.Cursor := crHourglass;    { Show hourglass cursor }

  BufC[1]:='R';BufC[2]:='U';BufC[3]:='N';
  for uu:=0 to 255 do
  Begin
    SigmaAscii[uu]:=0;
  end;
  SigmaMin:=1000000;
  for i:=0 to Fsize-1 do   //Jumlah tiap karakter
  begin
    Inbuf:=integer(Buf[1+i]);
    inc(SigmaAscii[Inbuf]);
  end;
  for uu:=0 to 255 do
  begin
    if SigmaAscii[uu]<SigmaMin then
    begin
      SigmaMin:=SigmaAscii[uu];
      BitPenanda:=uu;
    end;
  end;

  BufC[4]:=chr(BitPenanda);  //Bit penanda
  ii:=4;
  i:=1;
  ulang4:=1;
  SameBit:=Integer(Buf[i]);
  While i<FSize do
```

```
 begin
  while SameBit=BitPenanda do
  begin
   ii:=ii+1;
   Bufc[ii]:=chr(BitPenanda);
   ii:=ii+1;
   Bufc[ii]:=chr(BitPenanda);
   i:=1+i;
   samebit:=Integer(Buf[i]);
  end;
  i:=1+i;
  buflong:=Integer(Buf[i]);
  if (buflong=SameBit) and (buflong<>BitPenanda) then
   begin
   if ulang4<255 then ulang4:=ulang4+1
   else
    begin
     Bufc[ii+1]:=chr(BitPenanda);
     Bufc[ii+2]:=chr(ulang4);
     Bufc[ii+3]:=chr(SameBit);
     ii:=ii+3;
     ulang4:=1;
    end;
   end
  else                        {byte selanjutnya tak sama}
   begin
    if ulang4>3 then            { >3}
     begin
      if BitPenanda<>ulang4 then
       begin
        Bufc[ii+1]:=chr(BitPenanda);
        Bufc[ii+2]:=chr(ulang4);
        Bufc[ii+3]:=chr(SameBit);
        ii:=ii+3;
       end
       else
       begin
        Bufc[ii+1]:=chr(BitPenanda);    //Bit Penanda=Jumlah
Ulang
```

```
            Bufc[ii+2]:=chr(ulang4-1);
            Bufc[ii+3]:=chr(SameBit);
            Bufc[ii+4]:=chr(SameBit);
            ii:=ii+4;
          end;
      end
    else                    { <4}
      begin
        for uu:=1 to ulang4 do
          begin
            ii:=ii+1;
            Bufc[ii]:=chr(SameBit);
          end;
      end;
    if Buflong=BitPenanda then
    begin
      ii:=ii+1;
      Bufc[ii]:=chr(BitPenanda);
      ii:=ii+1;
      Bufc[ii]:=chr(BitPenanda);
      if i<FSize then
      begin
        i:=1+i;
        buflong:=Integer(Buf[i]);
      end;
    end;
    SameBit:=Buflong;
    ulang4:=1;
  end;
end;
if ulang4>3 then          {terakhir >3}
begin
  Bufc[ii+1]:=chr(BitPenanda);
  Bufc[ii+2]:=chr(ulang4);
  Bufc[ii+3]:=chr(SameBit);
  ii:=ii+3;
end
else                    { <4}
begin
```

```
    for uu:=1 to ulang4 do
    begin
      if sameBit<>BitPenanda then
      begin
        ii:=ii+1;
        Bufc[ii]:=chr(SameBit);
      end;
    end;
  end;
  FCLSize:=ii;
  LblFileSize2.caption:=inttostr(FCLSize)+' bytes  =
'+inttostr(FCLSize*100 div FSize)+' %';
  Screen.Cursor := Save_Cursor;
  SButSave.caption:='Save with Run Length Algorithm';
  FCSaveSize:=FCLSize;
  SButSave.enabled:=true;
  LblHasil.caption:='Hasil pemampatan dengan Algoritma Run
Length :';
end;

procedure TFormUtama.SbutSaveClick(Sender: TObject);
var
  ToF: Textfile;
  ii: longint;
begin
    if SaveDialog1.Execute then
    begin
      AssignFile(ToF, SaveDialog1.FileName);
       Rewrite(ToF);            { Record size = 1 }
     for ii:=1 to FCSaveSize do
      begin
        Write(ToF, bufC[ii]);
      end;
      CloseFile(ToF);
    end;
end;

procedure TFormUtama.SButHalfByteClick(Sender: TObject);
var
```

```
    SameHB,i,ii,uu: LongInt;
    Inbuf, Genap, BitPenanda: Integer;
    BitKiriS,BitKiriT: string;
    BitKananS,BitKananT: string;
    SigmaMin:LongInt;
    JadiPenanda:Boolean;
    Save_Cursor:TCursor;
    SigmaAscii: Array[0..255] of LongInt;
Begin
  Save_Cursor := Screen.Cursor;
  Screen.Cursor := crHourglass;    { Show hourglass cursor }
  BufC[1]:='H';BufC[2]:='A';BufC[3]:='L';
  for uu:=0 to 255 do
  Begin
    SigmaAscii[uu]:=0;
  end;
  SigmaMin:=1000000;
  for i:=0 to Fsize-1 do   //Jumlah tiap karakter
  begin
    Inbuf:=integer(Buf[1+i]);
    inc(SigmaAscii[Inbuf]);
  end;
  for uu:=0 to 255 do
  begin
    if SigmaAscii[uu]<SigmaMin then
    begin
      SigmaMin:=SigmaAscii[uu];
      BitPenanda:=uu;
    end;
  end;
  BufC[4]:=chr(BitPenanda);  //Bit penanda
  ii:=4;
  BitKiriS:=(inttohex(Integer(Buf[FSize]),2));
  Delete(BitKiriS,2,1);
  if BitKiriS='4' then Buf[FSize+1]:='P' else Buf[FSize+1]:='O';
//O=4F P=50 +akhir file
  i:=1;
  SameHB:=1;
  JadiPenanda:=false;
```

```
  BitKiriS:=(inttohex(Integer(Buf[i]),2));
  Delete(BitKiriS,2,1);
  While i<=FSize do
  Begin
    if (SameHB=1) and (Buf[i]=chr(BitPenanda)) then
JadiPenanda:=True;
    BitKananS:=(inttohex(Integer(Buf[i]),2));
    Delete(BitKananS,1,1);
    i:=i+1;
    BitKiriT:=(inttohex(Integer(Buf[i]),2));
    Delete(BitKiriT,2,1);
    if (BitKiriT=BitKiriS) and (JadiPenanda=false) then
      begin
        SameHB:=SameHB+1;
        if (SameHB mod 2)=1 then
        begin
          BitKananT:=(inttohex(Integer(Buf[i]),2));
          Delete(BitKananT,1,1);
          if StrtoIntDef('$'+BitKananS+BitKananT,255)=BitPenanda
then JadiPenanda:=True;
        end;
      end
    else                        //tidak sama
      begin
        i:=i-SameHB;
        if (SameHB<7) or (JadiPenanda=True) then
        begin
          for uu:=i to i+SameHB-1 do
          begin
            ii:=ii+1;
            BufC[ii]:=Buf[uu];
            if Buf[uu]=chr(BitPenanda) then  //Bit penanda 2X
            begin
              ii:=ii+1;
              BufC[ii]:=Buf[uu];
            end;
          end;
        end
        else                          // SameHB >= 7
```

```
begin
  ii:=ii+1;
  Bufc[ii]:=chr(BitPenanda);
  ii:=ii+1;
  Bufc[ii]:=Buf[i];
  genap:=1;
  for uu:=i+1 to i+SameHB-1 do
  begin
    case genap of
    1:
    begin
      BitKiriS:=(inttohex(Integer(Buf[uu]),2));
      Delete(BitKiriS,1,1);   //ambil kanan
      genap:=0;
    end;
    0:                          //ganjil
    begin
      BitKiriT:=(inttohex(Integer(Buf[uu]),2));
      Delete(BitKiriT,1,1);   //ambil kanan
      ii:=ii+1;
      Bufc[ii]:=char(StrtoIntDef('$'+BitKiriS+BitKiriT,255));
      genap:=1;
    end;
    end;
  end;
  ii:=ii+1;
  Bufc[ii]:=chr(BitPenanda);  //bit penutup
  if genap=0 then
  begin
    ii:=ii+1;
    Bufc[ii]:=char(Integer(Buf[i+SameHB-1])) ;
  end;
 end;
 i:=i+SameHB;
 SameHB:=1;
 JadiPenanda:=false;
 BitKiriS:=(inttohex(Integer(Buf[i]),2));
 Delete(BitKiriS,2,1);
end;
```

```
  end;
  FCBSize:=ii;
  LblFileSize2.caption:=inttostr(FCBSize)+' bytes  =
'+inttostr(FCBSize*100 div FSize)+' %';
  Screen.Cursor := Save_Cursor;
  SButSave.enabled:=true;
  SButSave.caption:='Save with Half Byte Algorithm';
  FCSaveSize:=FCBSize;
  LblHasil.caption:='Hasil pemampatan dengan Algoritma Half
Byte :';
end;

procedure TFormUtama.SButHuffmanClick(Sender: TObject);
var
   i,ii,iii: LongInt;
   u,uu, uuu, NoAlih, JumPindah: Integer;
   KarakterLD0,HighestLevel :integer;
   Save_Cursor:TCursor;
   SigmaAscii: Array[0..255] of LongInt;
   KarakterHuf: array[0..255] of Integer;
   JumlahKarakterHuf: array[0..255] of Integer;
   KodeHuf: array[0..255] of string;
   KodeTem: string;
   JumlahTem, JumlahKiri: Integer;
   Inbuf: Integer;
   AngkaTerkecil: LongInt;
   LevelTree: array[0..1000] of integer;
   FrekTree: array[0..1000] of integer;
   KarTree: array[0..1000] of string;
   JumTree, FrekSource, FrekTarget: Integer;
   Oke, nol: boolean;
Begin
   Save_Cursor := Screen.Cursor;
   Screen.Cursor := crHourglass;    { Show hourglass cursor }
   for uu:=0 to 255 do
   Begin
     SigmaAscii[uu]:=0;
     JumlahKarakterHuf[uu]:=0;
     KarakterHuf[uu]:=0;
```

```
   end;
   KarakterLD0:=0;

   for i:=0 to Fsize-1 do   //Jumlah tiap karakter
   begin
    Inbuf:=integer(Buf[1+i]);
    SigmaAscii[Inbuf]:=SigmaAscii[Inbuf]+1;
   end;
   for u:=0 to 255 do     //Pengurutan
   begin
    AngkaTerkecil:=1000000;
    for uu:=0 to 255 do
    begin
     if (SigmaAscii[uu]>0) and (SigmaAscii[uu]<AngkaTerkecil)
then
      begin
       AngkaTerkecil:=SigmaAscii[uu];
       KarakterHuf[u]:=uu;
       JumlahKarakterHuf[u]:=AngkaTerkecil;
      end;
    end;
    if JumlahKarakterHuf[u]>0 then
     begin
      KarakterLD0:=KarakterLD0+1;
      SigmaAscii[KarakterHuf[u]]:=0;
     end;
   end;

//-------------------- Huffman Tree
   For u:=0 to KarakterLD0-1 do
   Begin
    LevelTree[u]:=1;
    KarTree[u]:=chr(KarakterHuf[u]);
    FrekTree[u]:=JumlahKarakterHuf[u];
   end;

   JumTree:=KarakterLD0;
   For u:=1 to KarakterLD0-1 do          //n-1 kali
   Begin
```

```
    oke:=false;
    uuu:=0;
    repeat    //cari target
     inc(uuu);
     if LevelTree[uuu]=1 then
     begin
       for uu:=JumTree-1 downto 0 do
       begin
         LevelTree[uu+1]:=LevelTree[uu];
         KarTree[uu+1]:=KarTree[uu];
         FrekTree[uu+1]:=FrekTree[uu];
       end;
       LevelTree[0]:=1;
       KarTree[0]:=KarTree[1]+KarTree[uuu+1];
       FrekTree[0]:=FrekTree[1]+FrekTree[uuu+1];
       inc(JumTree);
       for uu:=1 to uuu do          //cabang pohon source
       begin
         inc(LevelTree[uu]);
       end;
       uu:=1;
       repeat                    //target & cabang pohon target
         inc(LevelTree[uu+uuu]);
         inc(uu);
       until (LevelTree[uu+uuu]=1) or (uu+uuu>JumTree);
       oke:=true;
     end;
    until oke=true;

    FrekSource:=FrekTree[0];   //urut
    NoAlih:=0;
    For uuu:=1 to Jumtree-1 do
    begin
      FrekTarget:=FrekTree[uuu];
      if (FrekSource>FrekTarget) and (LevelTree[uuu]=1) then
NoAlih:=uuu;
    end;
    If NoAlih>0 then
    begin
```

```
      uuu:=0;
      JumPindah:=0;
      repeat
        LevelTree[JumTree+uuu]:=LevelTree[uuu];
        KarTree[JumTree+uuu]:=KarTree[uuu];
        FrekTree[JumTree+uuu]:=FrekTree[uuu];
        Inc(JumPindah);
        inc(uuu);
      until levelTree[uuu]=1;
      repeat
        LevelTree[uuu-JumPindah]:=LevelTree[uuu];
        KarTree[uuu-JumPindah]:=KarTree[uuu];
        FrekTree[uuu-JumPindah]:=FrekTree[uuu];
        inc(uuu);
      until (LevelTree[uuu]=1) and (FrekTree[uuu]>=FrekSource);
      For uu:=1 to JumPindah do
      begin
        LevelTree[uuu-JumPindah+uu-1]:=LevelTree[JumTree+uu-
1];
        KarTree[uuu-JumPindah+uu-1]:=KarTree[JumTree+uu-1];
        FrekTree[uuu-JumPindah+uu-1]:=FrekTree[JumTree+uu-1];
      end;
    end;
  end;

//---------------- Huffman Code
  HighestLevel:=1;
  For uu:=0 to JumTree-1 do
  Begin
    If LevelTree[uu]>HighestLevel then HighestLevel:=
LevelTree[uu];
  end;
  For uu:=2 to HighestLevel do  //mulai level 2
  Begin
    Nol:=true;
    For ii:=0 to JumTree-1 do
    Begin
      if LevelTree[ii]=uu then
      begin
```

```
      for uuu:=1 to length(KarTree[ii]) do
      begin
        KodeTem:=copy(KarTree[ii],uuu,1);
        NoAlih:=0;
        for FrekTarget:=0 to KarakterLD0-1 do
        begin
          if KodeTem=chr(KarakterHuf[FrekTarget]) then
NoAlih:=FrekTarget;
        end;
        case nol of
          true: KodeHuf[NoAlih]:=KodeHuf[NoAlih]+'0';
          false: KodeHuf[NoAlih]:=KodeHuf[NoAlih]+'1';
        end;
      end;
      nol:=not(nol);
    end;
  end;
 end;


//-------- Hasil Pemampatan
  BufC[1]:='H';
  BufC[2]:='U';
  BufC[3]:='F';
  KodeTem:=InttoHex(FSize,6);
  BufC[4]:=chr(StrtoIntDef('$'+copy(KodeTem,1,2),255));
//jumlah byte
  BufC[5]:=chr(StrtoIntDef('$'+copy(KodeTem,3,2),255));
  BufC[6]:=chr(StrtoIntDef('$'+copy(KodeTem,5,2),255));
  BufC[7]:=chr(KarakterLD0-1);
  FCHSize:=7;
  For uu:=1 to KarakterLD0 do  // karakter, kode, panjang bit
  begin
    BufC[(uu-1)*4+8]:=chr(KarakterHuf[uu-1]);
    BufC[(uu-
1)*4+9]:=chr(StrtoIntDef('$'+copy(IntToHex(BinToDec(KodeHuf
[uu-1]),4),1,2),225));
```

```
   BufC[(uu-
1)*4+10]:=chr(StrtoIntDef('$'+copy(IntToHex(BinToDec(KodeHu
f[uu-1]),4),3,2),225));
   BufC[(uu-1)*4+11]:=chr(Length(KodeHuf[uu-1]));
   FCHSize:=FCHSize+4;
  end;
  KodeTem:='';
  For ii:=1 to FSize do   // isi file
  begin
   uu:=-1;
   repeat
    inc(uu);
   until chr(KarakterHuf[uu])=Buf[ii];
   KodeTem:=KodeTem+KodeHuf[uu];
   if length(KodeTem)>=8 then
   begin
    repeat
     Inc(FCHSize);
     BufC[FCHSize]:=chr(BinToDec(copy(KodeTem,1,8)));
     If Length(KodeTem)>8 then
KodeTem:=copy(KodeTem,9,Length(KodeTem)-8) else
KodeTem:='';
    until Length(KodeTem)<8;
   end;
  end;
  If Length(KodeTem)>0 then  //sisa
  begin
   KodeTem:=KodeTem+StringofChar('0',8-Length(KodeTem));
   Inc(FCHSize);
   BufC[FCHSize]:=chr(BinToDec(KodeTem));
  end;
   LblFileSize2.caption:=inttostr(FCHSize)+' bytes  =
'+inttostr(FCHSize*100 div FSize)+' %';
   Screen.Cursor := Save_Cursor;
   SButSave.caption:='Save with Huffman Algorithm';
   FCSaveSize:=FCHSize;
   SButSave.enabled:=true;
end;
```

```
function TFormUtama.BinToDec(Biner:string): LongInt;
var
  PanjangKar,uu :Integer;
  Desimal: LongInt;
  SatuBit: String;
begin
  Desimal:=0;
  PanjangKar:=length(biner);
  for uu:=1 to PanjangKar do
  Begin
    SatuBit:=copy(Biner,PanjangKar-uu+1,1);
    Desimal:=Desimal+Trunc((StrToInt(SatuBit))*exp((uu-
1)*ln(2)));
  end;
  BinToDec:=Desimal;
end;

procedure TFormUtama.RadioCompressClick(Sender: TObject);
begin
  ImageTampil.picture:=Image1.picture;
  GroupMetode.Visible:=True;
  SButDeCompress.visible:=false;
  SButSave.visible:=true;
  SButSaveOrig.visible:=false;
end;

procedure TFormUtama.RadioDecompressClick(Sender: TObject);
begin
  ImageTampil.picture:=Image2.picture;
  GroupMetode.Visible:=False;
  SButDeCompress.visible:=true;
  SButSave.visible:=false;
  SButSaveOrig.visible:=true;
end;


procedure TFormUtama.SButDeCompressClick(Sender: TObject);
var
  i,ii: LongInt;
```

```
  BitMin, uu, KarakterLD0, BitPenanda: Integer;
  JumSama:Integer;
  BitKiri,BitKanan,KodeTem,KodeBin: String;
  DapatKode: Boolean;
  KarakterHuf: array[0..255] of Integer;
  JumlahKarakterHuf: array[0..255] of Integer;
  KodeHuf: array[0..255] of string;
  Save_Cursor:TCursor;
begin
  Save_Cursor := Screen.Cursor;
  Screen.Cursor := crHourglass;    { Show hourglass cursor }

 case metode of
  1:        //============================Run Length
  begin
    i:=0;
    BitPenanda:=Integer(Buf[4]);
    ii:=4;
    Repeat
      inc(ii);
      If Integer(Buf[ii])=BitPenanda then
      begin
        If Integer(Buf[ii+1])=BitPenanda then
        begin
          inc(i);
          BufC[i]:=chr(BitPenanda);
          inc(ii);
        end
        else                  // Kompress
        begin
          JumSama:=Integer(Buf[ii+1]);
          for uu:=1 to JumSama do
          begin
            inc(i);
            BufC[i]:=Buf[ii+2];
          end;
          ii:=ii+2;
        end;
      end
```

```
    else                    // Karakter Asli
    begin
      inc(i);
      BufC[i]:=Buf[ii];
    end;
  Until ii>=FSize;
  FCSize:=i;
end;

2:          //============================HalfByte;
begin
  i:=0;
  BitPenanda:=Integer(Buf[4]);
  ii:=4;
  Repeat
    inc(ii);
    If Integer(Buf[ii])=BitPenanda then
    begin
      If Integer(Buf[ii+1])=BitPenanda then
      begin
        inc(i);
        BufC[i]:=chr(BitPenanda);
        inc(ii);
      end
      else                  // Kompress
      begin
        inc(i);
        BufC[i]:=Buf[ii+1];
        BitKiri:=(inttohex(Integer(Buf[ii+1]),2));
        Delete(BitKiri,2,1);
        ii:=ii+2;
        while (Integer(Buf[ii])<>BitPenanda) and (ii<FSize) do
        begin
          BitKanan:=(inttohex(Integer(Buf[ii]),2));
          Delete(BitKanan,2,1);
          inc(i);
          BufC[i]:=chr(StrtoIntDef('$'+BitKiri+BitKanan,255));
          BitKanan:=(inttohex(Integer(Buf[ii]),2));
          Delete(BitKanan,1,1);
```

```
          inc(i);
          BufC[i]:=chr(StrtoIntDef('$'+BitKiri+BitKanan,255));
          inc(ii);
        end;
      end;
    end
    else                    // Karakter Asli
    begin
      inc(i);
      BufC[i]:=Buf[ii];
    end;
  Until ii>=FSize;
  FCSize:=i;
 end;

 3:          //============================Huffman;
 begin

KodeTem:=IntToHex(Integer(Buf[4]),2)+IntToHex(Integer(Buf[5]
),2)+IntToHex(Integer(Buf[6]),2);
   FCSize:=StrtoIntDef('$'+KodeTem,255);
   KodeTem:=IntToHex(Integer(Buf[7]),2);
   KarakterLD0:=StrtoIntDef('$'+KodeTem,255)+1;
   ii:=7;
   BitMin:=256;
   for uu:=1 to KarakterLD0 do          //Kode Huffman
   begin
     ii:=ii+4;
     KarakterHuf[uu-1]:=Integer(Buf[ii-3]);          //byte ke-1
     JumlahKarakterHuf[uu-1]:=Integer(Buf[ii]);      //byte ke-4
     If JumlahKarakterHuf[uu-1]<BitMin then
BitMin:=JumlahKarakterHuf[uu-1];
     i:=Integer(Buf[ii-2])*256+Integer(Buf[ii-1]);   //byte ke-2&3
     KodeTem:=DecToBin(i);
     If length(KodeTem)<JumlahKarakterHuf[uu-1] then
     begin
       KodeTem:=StringofChar('0',JumlahKarakterHuf[uu-1]-
length(KodeTem))+KodeTem;
     end;
```

```
     KodeHuf[uu-1]:=KodeTem;
   end;

//----------- Uncompression Kode Huffman
   KodeTem:='';
   i:=0;
   repeat
     inc(ii);
     KodeBin:=DecToBin(Integer(Buf[ii]));
     KodeBin:=StringOfChar('0',8-length(KodeBin))+KodeBin;
     KodeTem:=KodeTem+KodeBin;
     DapatKode:=True;
     while (Length(KodeTem)>=BitMin) and (i<FCSize) and
(DapatKode=true) do
     begin
       uu := -1;
       DapatKode:=False;
       repeat
         inc(uu);
         if JumlahKarakterHuf[uu]<=Length(KodeTem) then
         begin
           if
copy(KodeTem,1,JumlahKarakterHuf[uu])=KodeHuf[uu] then
DapatKode:=True;
         end;
       until (DapatKode=True) or (uu>=KarakterLD0-1);
       if DapatKode=True then
       begin
         inc(i);
         BufC[i]:=chr(KarakterHuf[uu]);

KodeTem:=copy(KodeTem,JumlahKarakterHuf[uu]+1,Length(Ko
deTem)-JumlahKarakterHuf[uu]);
       end;
     end;
   until ii=FSize;
 end;
end;  //end case
If Metode>0 then
```

```
 begin
   If Metode=1 then LblHasil.Caption := 'File asli hasil
pemampatan Metode Run-Length';
   If Metode=2 then LblHasil.Caption := 'File asli hasil
pemampatan Metode Half-Byte';
   If Metode=3 then LblHasil.Caption := 'File asli hasil
pemampatan Metode Huffman';
   LblFileSize2.Caption := IntToStr(FCSize)+' bytes';
   SButSaveOrig.enabled:=true;
 end
 else MessageDlg('Bukan File Hasil Kompresi',
mtInformation,[mbOK], 0);
 Screen.Cursor := Save_Cursor;
end;

function TFormUtama.DecToBin(Desimal:LongInt): string;
var
  sisabagi: Integer;
  DesBin:LongInt;
  Biner: String;
begin
  Biner:='';
  DesBin:=Desimal;
  repeat
    SisaBagi:=Desbin mod 2;
    Desbin:=Desbin div 2;
    If SisaBagi=0 then Biner:='0'+Biner else Biner:='1'+Biner;
  until DesBin=0;
  DecToBin:=Biner;
end;

procedure TFormUtama.SButSaveOrigClick(Sender: TObject);
var
  ToF: Textfile;
  ii: longint;
begin
    if SaveDialog1.Execute then
    begin
      AssignFile(ToF, SaveDialog1.FileName);
```

```
     Rewrite(ToF);              { Record size = 1 }
    for ii:=1 to FCSize do
     begin
       Write(ToF, bufC[ii]);
     end;
     CloseFile(ToF);
   end;
end;

end.
```

KEPUSTAKAAN

Adam Osborne dan Davis Bunnel, Pengantar Komputer-Mikro, terjemahan oleh Setiyo Utomo, Ir. Jakarta, Erlangga, 1986.

Eniman M. Yunus, Catatan Kuliah Rangkaian Logika Lanjut, ITB, 1999.

G.H. Gonnet, Handbook of Algorithms and Data Structure, London, Addison-Wesley Publishing Company, International Computer Science Series.

Tony Suryanto, Pemampatan File dengan Algoritma Huffman, Jakarta, Dinastindo, 1995.