BAB II

LANDASAN TEORI

2.1 Kompresi Data

Kompresi data (pemampatan data) merupakan suatu teknik untuk memperkecil jumlah ukuran data (hasil kompresi) dari data aslinya. Pemampatan data umumnya diterapkan pada mesin komputer, hal ini dilakukan karena setiap simbol yang muncul pada komputer memiliki nilai bit – bit yang berbeda. Pengiriman data hasil kompresi dapat dilakukan jika pihak pengirim atau yang melakukan kompresi dan pihak penerima memiliki aturan yang sama dalam hal kompresi data. Pihak pengirim harus menggunakan algoritma kompresi data yang sudah dipilih dan pihak penerima juga menggunakan teknik dekompresi data yang sama dengan pengirim sehingga data yang diterima dapat dibaca kembali dengan benar. Kompresi data menjadi sangat penting karena memperkecil kebutuhan penyimpanan data, mempercepat pengiriman data, memperkecil kebutuhan *bandwith.* [1]

2.1.1 Jenis Kompresi data

Jenis kompresi data berdasarkan mode penerimaan data oleh manusia dapat dibagi menjadi dua, yaitu :

1. *Dialoque Mode*

   Yaitu proses penerimaan data dimana pengirim dan penerima seakan berdialog (*real time*), seperti pada contoh video *conference* dimana kompresi data harus berada dalam batas penglihatan dan pendengaran manusia.

2. *Retrieval Mode*

   Yaitu proses penerimaan data tidak dilakukan secara *real time*, dapat dilakukan *random acces* terhadap data dan dapat bersifat interaktif.

Jenis kompresi data berdasarkan outputnya dibagi menjadi dua bagian yaitu:

1. *Lossy Compression*

   Yaitu teknik kompresi dimana data hasil dekompresi tidak sama dengan data sebelum kompresi namun sudah "cukup" untuk digunakan. Contoh : Mp3, *streaming media,* JPEG, MPEG, dan WMA. Kelebihan dari *lossy compression* yaitu ukuran file lebih kecil dibanding *lossless* namun masih tetap memenuhi syarat untuk digunakan. Biasanya teknik ini membuang bagian – bagian data yang sebenarnya tidak berguna, tidak begitu dirasakan, tidak begitu dilihat oleh manusia sehingga manusia masih beranggapan bahwa data tersebut masih bisa digunakan walaupun sudah dikompresi. Misal terdapat *image* asli berukuran 12,249 *bytes,* kemudian dilakukan kompresi dengan JPEG kualitas 30 dan berukuran 1,869 *bytes* berarti *image* tersebut 85% lebih kecil dan *ratio* kompresi 15%.

2. *Lossless Compression*

   Yaitu teknik kompresi dimana data hasil kompresi dapat didekompres lagi dan hasilnya tepat sama seperti data sebelum proses kompresi. Teknik ini digunakan jika dibutuhkan data setelah dikompresi harus diekstrak/dekompres lagi tepat sama. Teknik kompresi *lossless* ini banyak digunakan untuk berkas – berkas dokumen, dimana sama sekali tidak diperkenankan adanya perbedaan antara informasi awal (sebelum proses kompresi) dan informasi yang diperoleh setelah proses dekompresi. Teknik kompresi ini mempertahankan kebutuhan informasi yang dikandung oleh data, sehingga informasi yang terkandung pada *file* yang telah terkompresi tetap terjaga meskipun ukurannya telah berubah dari ukuran data aslinya.

2.1.2 Metode Kompresi Data

Berdasarkan tipe peta kode yang digunakan untuk mengubah pesan awal (isi *file input*) menjadi sekumpulan *codeword,* metode kompresi dibagi menjadi dua kelompok, yaitu :

1. Metode Statik

   Menggunakan peta kode yang selalu sama. Metode ini membutuhkan dua fase (*two-pass*) : fase pertama untuk menghitung probabilitas kemunculan tiap simbol/karakter dan menentukan peta kodenya dan fase kedua untuk mengubah pesan menjadi sekumpulan kode yang akan ditransmisikan.

   Contohnya pada *Huffman static, arithmatic coding.*

2. Metode Dinamik (adaptif)

   Menggunakan peta kode yang dapat diubah dari waktu ke waktu. Metode ini disebut adaptif karena peta kode mampu beradaptasi terhadapa karakteristik isi *file* selama proses kompresi berlangsung. Metode ini bersifat *onepass,* karena isi *file* selama dikompres hanya diperlakukan satu kali pembacaan terhadap isi *file.* Contohnya pada algoritma LZW dan DMC.

Berdasarkan teknik pengkodean atau pengubahan simbol yang digunakan, metode kompresi dapat dibagi ke dalam tiga kategori, yaitu :

1. Metode *simbolwise*

   Menghitung peluang kemunculan dari tiap simbol dalam *file input,* lalu mengkodekan satu simbol dalam satu waktu, dimana simbol yang lebih sering muncul diberi kode lebih pendek dibandingkan simbol yang lebih jarang muncul. Contohnya pada *Huffman coding, arithmatic coding.*

2. Metode *dictionary*

   Menggantikan karakter/fragmen dalam *file input* dengan indeks lokasi dari karakter/fragmen tersebut dalam sebuah kamus (*dictionary*). Contohnya pada algoritma LZW.

3. Metode *predictive*

   Menggunakan model *finite-context* atau *finite-state* untuk memprediksi distribusi dari simbol – simbol selanjutnya. Contohnya pada algoritma DMC.

2.1.3 Klasifikasi Teknik Kompresi

Teknik kompresi dapat diklasifikasikan menjadi tiga kelompok, antara lain :

1. *Entropy Encoding*

   Merupakan klasifikasi teknik kompresi yang bersifat *lossless compression.* Teknik ini tidak berdasarkan media dengan spesifikasi dan karakteristik tertentu, namun berdasarkan urutan data. *Statistical encoding* pada klasifikasi teknik ini tidak memperhatikan semantik data. Contoh kelompok teknik kompresi ini adalah *Run-length coding, Huffman coding, Arithmatic coding.*

2. *Source Coding*

   Merupakan teknik kompresi yang bersifat *lossy compression*. Teknik ini berkaitan dengan data semantik (arti data) dan media. Contoh kelompok teknik kompresi ini ada *prediction (DPCM, DM), transformation (FFT, DCT), layered coding (bit position, subsampling, sub-band coding), vector quantization.*

3. *Hybrid Coding*

   Merupakan klasifikasi teknik kompresi yang bersifat gabungan antara *lossy compression* dan *lossy compression.* Contoh kelompok teknik kompresi ini adalah pada JPEG, MPEG, H.261, DVI.

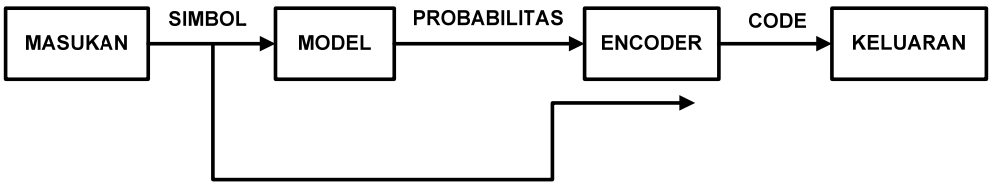
2.1.4 Encoder dan Decoder

Encoding merupakan teknik untuk mendapatkan kode – kode tertentu (*encoder*). Dari kode – kode tersebut dapat diaplikasikan untuk pemampatan data dan keamanan data. Dari data – data yang telah dikodekan tersebut, format –

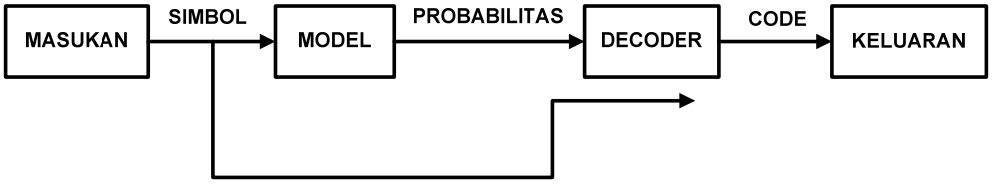
format isi dari data tersebut berbentuk kode – kode yang tidak bisa dibaca oleh *user*. Agar kode – kode tersebut bisa dibaca oleh *user,* maka kita perlu mengkodekan ulang data tersebut. Hal ini dikenal dengan nama *decoding (decoder)*. [2]

Secara umum pemampatan data merubah suatu simbol – simbol menjadi suatu kode – kode. Pemampatan dikatakan efektif jika ukuran perolehan kode – kode tersebut sangat kecil dibandingkan dengan ukuran kode simbol aslinya. Dari suatu kode – kode atau simbol – simbol dasar suatu model akan dinyatakan dalam kode khusus.

Proses *decoding,* yaitu proses pengambilan kode – kode yang telah dibuat menjadi simbol – simbol yang dikenal oleh *user.* Proses *decoder* ini membaca *header* dari kode – kode yang berisi informasi simbol dan jumlah simbol yang digunakan, setelah pembacaan *header* proses *encoder* dengan model ditunjukkan pada gambar 2.1, sedangkan proses *decoder* dengan model ditunjukkan pada gambar 2.2 dibawah ini.



Gambar 2.1 Model proses *Encoder*



Gambar 2.2 Model proses *Decoder*

2.2 Pengkodean Data

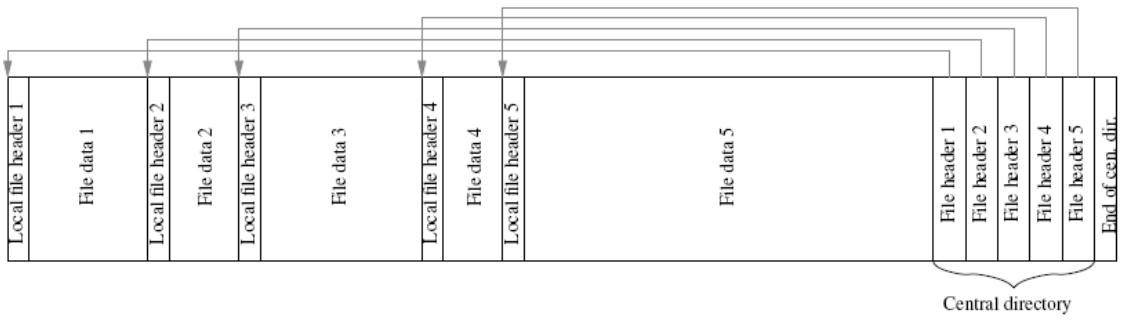
Data disimpan di dalam komputer pada *main memory* untuk diproses. Sebuah karakter data disimpan dalam *main memory* menempati posisi 1 *byte*. Pada komputer generasi yang pertama, 1 *byte* terdiri dari 4 bit, komputer generasi kedua, 1 *byte* terdiri dari 6 bit dan pada komputer generasi sekarang, 1 *byte* terdiri dari 8 bit. Suatu karakter data yang disimpan di *main memory* diwakili dengan kombinasi dari digit *binary* (*binary* digit atau bit). Suatu kode biner dapat digunakan untuk mewakili suatu karakter.

Suatu komputer yang berbeda menggunakan kode biner yang berbeda untuk mewakili suatu karakter. Komputer yang 1 *byte* terdiri dari 4 bit, menggunakan kode binary yang berbentuk kombinasi 4 bit, yaitu BCD (*Binary Coded Decimal*). Komputer yang menggunakan 6 bit untuk 1 *byte* nya, menggunakan kode biner yang terdiri dari 6 kombinasi bit, yaitu SBCDIC (*Standard Binary Coded Decimal Interchange Code*). Komputer yang terdiri dari 8 bit, menggunakan kode biner yang terdiri dari kombinasi 8 bit, yaitu EBCDIC (*Extended Binary Coded Decimal Interchange Code*) atau ASCII (*American Standard Code of Information Interchange*).[2]

2.3 Algoritma kompresi ZLIB

Algoritma ZLIB merupakan turunan dari Agoritma kompresi Deflate. Algortima ini diciptakan oleh Jean-Loup Gailly yang menciptakan proses kompresi data dan Mark Adler yang menciptakan proses dekompresi data. Algoritma Zlib melakukan kompresi dengan pengkompresian data yang terdiri dari serangkaian blok, sesuai dengan blok inputan data tersebut. Setiap blok pada data tersebut dikompresi dengan menggunakan algoritma kompresi data Deflate sebagai kompressor yang merupakan variasi dari Algoritma LZ77 dikombinasikan dengan Huffman Coding.[3]

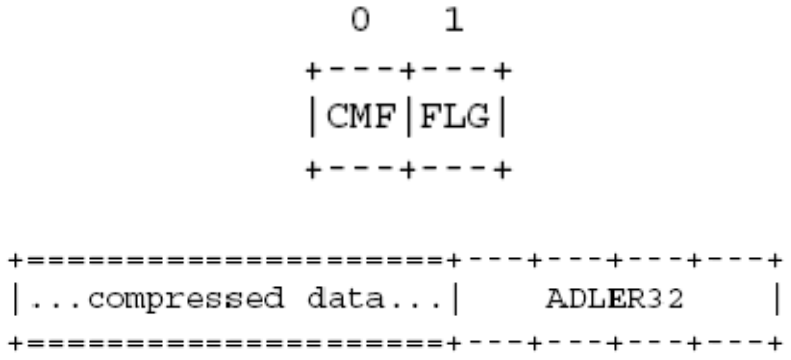
Data yang dikompresi oleh algoritma ZLIB akan diberikan pembungkus berupa data header setelah data tersebut dikompresi oleh algoritma Deflate. Header file yang diberikan pada data yang telah dikompresi tersebut merupakan bagian dari Algoritma ZLIB. Adapun struktur file dari agoritma ZLIB beserta header file yang telah diberikan pada data yeng telah dikompresi ditunjukkan pada gambar 2.3.



Gambar 2.3 Struktur *File* ZLIB

2.3.1 Format Data ZLIB

Struktur format data Algoritma ZLIB memiliki struktur seperti pada gambar 2.4, dimana format data yang diletakkan setelah data kompresi merupakan header dari data Algoritma ZLIB :



Gambar 2.4 Struktur format data ZLIB

Dari format data Zlib tersebut, setiap data yang mungkin muncul setelah ADLER32 bukan bagian dari Algoritma ZLIB.

1. CMF (*Compression Method and Flags*)

   Byte ini dibagi menjadi metode kompresi 4-bit dan *field* informasi bergantung pada metode kompresi.

2. CM (*Compression Method*)

   Ini mengidentifikasikan metode kompresi yang digunakan dalam file tersebut. Contohny CM = 8 menunujukkan bahwa metode kompresi yang digunakan adalah metode “Deflate”.

3. CINFO (*Compression info*)

   Untuk CM = 8, CINFO adalah algoritma dasar yang digunakan yaitu algoritma LZ77.

4. FLG (FLaGs)

   Flag pada sebuah *byte* dibagi antara lain sebagai berikut :

   Bit 0 – 4 FCHECK (untuk memeriksa bit CMF dan FLG)

   Bit 5 FDICT

   Bit 6 – 7 FLEVEL (Tingkat kompresi)

5. FDICT (*Preset Dictionary*)

   Jika FDICT diatur, identifier dari kamus DICT diletakkan setelah byte FLG. Dimana kamus dari data tersebut adalah urutan *byte* awal pada kompressor yang tidak menghasilkan output terkomprsi. DICT adalah urutan *byte checksum* Adler-32. Dekompressor ini dapat digunakan untuk menentukan kamus yang telah digunakan oleh kompressor.

6. FLEVEL (Level kompresi)

   Flag ini digunakan untuk menentukan metode kompresi tertentu. Contohnya metode “deflate” (CM = 8) flag ini ditentukan sebagai berikut.

   0 – kompresor menggunakan Algoritma tercepat

   1 – kompresor menggunakan Algoritma cepat

   2 – kompresor menggunakan Algoritma default

   3 – kompresor menggunakan Algoritma kompresi maksimum, algoritma paling lambat.

   Informasi didalam FLEVEL tidak diperlukan untuk melakukan proses dekompresi, melainkan untuk mengindikasikan jika proses kompresi diulangi.

7. Compressed Data

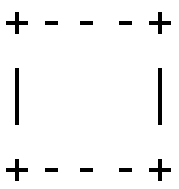
   Bagian ini merupakan file yang telah diberikan proses kompresi. Contohnya algoritma kompresi ZLIB menggunakan algoritma deflate sebagai algoritma kompressor nya.

8. ADLER32

   Ini berisi nilai *checksum* dari data terkompresi (tidak termasuk kamus data) yang dihitung menurut algoritma Adler-32. Algoritma ini adalah ekstensi 32-bit dan perbaikan dari algoritma Fletcher, yang digunakan dalam ITU-T X.224 / standar ISO8073. Adler-32 terdiri dari jumlah dua akumulasi per *byte*: s1 adalah jumlah dari semua *byte*, s2 adalah jumlah dari semua nilai s1. Jumlah dari keduanya terdiri dari 65.521 modul. s1 diinisialisasi dengan 1, s2 diinisialisai dengan 0. Algoritma Adler-32 memeriksa jumlah yang disimpan sebagai s2 * 65.536 + s1 dalam *most-significant-byte* yang pertama.

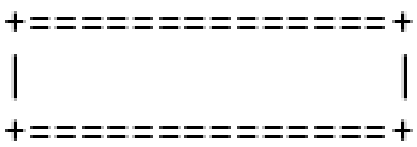
2.3.2 Spesifikasi Detail Algoritma ZLib

Dalam diagram data, dengan merepresentasikan 1 *byte* data digambarkan dalam bentuk kotak seperti yang dibawah ini :



Gambar 2.5 Kotak untuk merepresentasikan nilai 1 *byte*

Sedangkan untuk merepresentasikan perwakilan beberapa variabel *byte* digambarkan dalam bentuk kotak seperti yang ada di bawah ini :



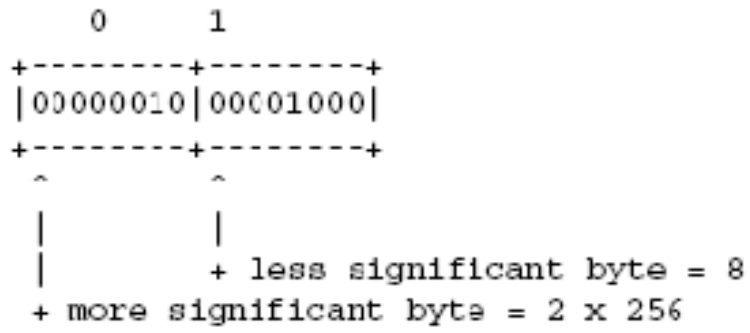
Gambar 2.6 Kotak untuk merepresentasikan nilai beberapa variabel *byte*

Byte disimpan dalam komputer tidak memiliki perintah untuk melakukan "Bit Orde", karena *byte* yang disimpan selalu dianggap sebagai satu unit. Namun, *byte* tersebut dianggap sebagai integer yang bernilai 0 – 255 yang memiliki *most-significant* bit dan *least-significant bit*. Dicontohkan bit number pada gambar 2.6 :



Gambar 2.7 Contoh bit Number

Dalam komputer, nomor dapat menduduki beberapa *byte*. Semua nomor *multi-byte* dalam format yang dijelaskan di sini disimpan dengan *MOST-significant byte* yang pertama (di alamat memori yang lebih rendah). Contoh desimal nomor 520 disimpan sebagai *most-significant byte* :

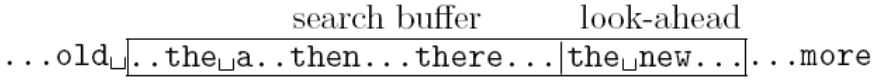


Gambar 2.8 Desimal 520 yang disimpan sebagai *most significant byte*

2.3.3 Algoritma Deflate

Untuk melakukan proses kompresi, Algoritma ZLIB ini menggunakan Algoritma Deflate sebagai kompressornya. Dimana Algoritma Deflate ini merupakan gabungan dari dua buah algoritma kompresi yaitu Algoritma LZ77 dan Algoritma Huffman.

Pada Algoritma LZ77 yang asli mencoba untuk mencocokkan teks dalam yang terlihat didepan *buffer* dari *string* yang telah masuk pada *buffer* pencarian. Contohnya pada Gambar 2.9 yaitu :



Gambar 2.9 Contoh *Buffer string* untuk Algoritma LZ77

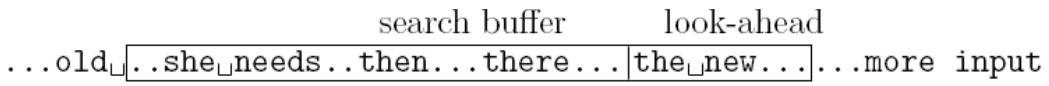
Pada *buffer string* yang terlihat didepan diawali dengan *string* the␣ yang dapat dicocokkan dengan salah satu dari tiga *string* didalam *buffer* yang dicari (*search buffer*). Algoritma LZ77 menulis sebuah token pada aliran kompresi yang terjadi, dimana setiap token merupakan sebuah triplet (*offset, length, next symbol*). Ketiga komponen triplet ini akan selalu diperlukan apabila *string* yang dicari pada search

buffer belum juga menemukan kecocokan, akan tetapi itu akan mengurangi kinerja dari algoritma LZ77 apabila string yang akan dicari kebanyakan tidak memiliki kecocokan pada string yang berada pada *search buffer*. Namun algoritma Deflate yang merupakan variasi dari Algoritma LZ77 ini menghilangkan ketiga komponen triplet pada LZ77 dan menggantinya menjadi sepasang komponen saja yaitu *offset* dan *length* pada kompresi stream. Ketika *string* yang dicari tidak cocok dengan *string* yang ada di *search buffer*, karakter yang tidak memiliki kecocokan tersebut akan dikompresi stream oleh deflate dan bukan dianggap sebagai sebuah token. Sehingga aliran kompresi yang terjadi terdiri dari tiga jenis entitas yaitu : *Literal* (karakter yang tidak ditemukan), O*ffsets* (disebut jarak didalam Deflate Literatur), dan juga panjang. Deflate sebenarnya menulis *Huffman Code* didalam kompresi stream untuk entitas dan menggunakan dua tabel *code*. Satu tabel untuk *literal* dan *length* dan satu tabel lagi digunakan untuk jarak. Proses pencarian akan dapat dilakukan karena *byte literal* yang normal memiliki interval [0,255], sedangkan *length* dan *byte* yang dimiliki oleh Deflate memiliki batas hingga 258 *byte*. Jarak yang dicari juga dapat bernilai besar karena Deflate dapat melakukan pencarian *buffer* hingga 32*Kbytes*.

Ketika jarak dan *length string* telah ditentukan, *encoder* akan mencari pada tabel *length* untuk menemukan kode panjang. Kode ini (diistilahkan dengan "*edoc*") nantinya akan diganti dengan kode huffman yang tertulis pada kompresi stream. *Encoder* kemudian melakukan pencarian pada tabel kode *distance* untuk mencari kode jarak dan ditulis didalam kompresi stream. *Decoder* akan

mengetahui kapan harus menggunakan kode jarak, karena kode jarak ini akan selalu terhubung dengan kode panjang

Varian dari algoritama LZ77 yang digunakan oleh deflate untuk memilih tingkat kecocokan. Misalkan dua *buffer* berisi :



Gambar 2.9 Contoh *Buffer string* untuk Algoritma LZ77

Length kecocokan yang dilihat pada gambar 2.9 adalah 3. Sebelum memilih *string* yang cocok, *encoder* menyimpan string t dari *look-ahead buffer* dan memulai pencocokan yang kedua dengan mencoba untuk mencocokkan string he␣new.. dengan *search buffer*. Jika menemukan kecocokan, *output* t akan berperan sebagai *literal* mengikuti *length* dari kecocokan. Terdapat 3 mode yang mengatur percobaan nilai kecocokan yang kedua. Pada mode "*normal – compression*", jika nilai kecocokan pertama dirasa sudah cukup, maka nilai kecocokan yang kedua akan dikurangi. Pada mode "*high – compression*", *encoder* akan selalu memprioritaskan kecocokan yang kedua sehingga akan meningkatkan kompresi namun akan memakan banyak waktu untuk melakukan pemilihan hingga cocok. Pada mode "*fast – compression*", nilai kecocokan yang kedua akan dihilangkan.

Kompresi deflate akan memasukkan data yang diinput kedalam blok, dimana setiap blok akan dikompresi secara terpisah. Setiap blok dapat memiliki *length* yang berbeda – beda dan *length* blok ditentukan oleh *encoder* berdasarkan ukuran kode yang digunakan (yang panjangnya terbatas 15 bit) dan memori yang

tersedia untuk *encoder* (terkecuali blok pada mode 1 terbatas untuk 65.535 *byte* data yang tidak terkompresi). Deflate *decoder* harus mampu untuk membaca sandi blok dari berbagai ukuran. Deflate menawarkan dua mode kompresi dan setiap blok dapat berada dalam metode apapun. Mode yang ditawarkan adalah sebagai berikut :

1. Tidak ada kompresi. Mode ini digunakan untuk *file* atau bagian dari *file* yang sudah dikompresi atau untuk kasus dimana kompresi software yang meminta segmen file tanpa kompresi. Contoh nya jika seorang pengguna yang ingin memindahkan file 200 MB ke komputer yang lain yang hanya memiliki *zip drive* 100MB. Pengguna tersebut mungkin ingin segmen *file* tersebut menjadi dua segmen 100MB tanpa kompresi. Mode ini tidak menggunakan tabel kode.
2. Kompresi dengan kode tabel yang sudah ditetapkan. Dua tabel kode yang dibangun kedalam *encoder* dan *decoder* yang selalu digunakan. Hal ini mempercepat kompresi dan dekompresi dan memiliki keuntungan bahwa kode tabel tidak perlu ditulis pada kompresi stream. Performa kompresi dapat berkurang jika data yang terkompresi secara statistik berbeda dari data yang digunakan untuk mengatur kode tabel. *Literal* dan *length* yang cocok berada pada tabel pertama dan digantikan oleh kode ("*edoc*"). Blok ditulis pada kompresi stream dalam mode ini dimulai dengan sebuah header khusus untuk menunjukkan metode 2.

2.3.4 Detail mode kompresi Algoritma Deflate

Setiap blok dimulai dengan sebuah header 3 bit dimana bit pertama adalah 1 untuk blok terakhir yang ada didalam file dan 0 untuk semua blok. Dengan memperhatikan bahwa blok data yang terkompresi tidak selalu berakhir pada batas *byte*. Sehingga informasi di blok tersebut buat *decoder* untuk membaca semua potongan – potongan blok yang terkompresi dan menjelaskan ujung pada blok. *Header* 3 bit blok berikutnya kan mengikuti blok ada dan mungkin terletak pada posisi dalam byte pada kompresi stream.

Format blok pada mode 1 adalah sebagai berikut :

1. *Header* 3-bit 000 atau 100
2. Sisa *byte* yang sedang berlangsung akan dilewati dan empat *byte* yang berikutnya berisi LEN dan satu komplemen LEN (sebagai nomor 16-bit unsigned), dimana LEN adalah jumlah dari *byte* data didalam blok. Inilah yang menyebabkan ukuran blok dalam mode ini terbatas untuk 65.535 *byte*.
3. LEN *byte* data.

Format blok pada mode 2 adalah sebagai berikut :

1. *Header* 3-bit 001 atau 101
2. Blok ini diikuti langsung dengan kode awalan yang tetap untuk *literal/length* dan kode awalan khusus dari jarak.
3. Kode 256 yang menunjuk akhir dari blok

Mode 2 menggunakan dua kode tabel, 1 untuk *literal/length* dan yang lain untuk *distance*. Kode dari tabel pertama sebenarnya tidak ditulis didalam

kompresi stream, sehingga untuk menghilangkan ambiguitas, istilah "*edoc*" disini digunakan untuk menunjuk pada kode di tabel pertama. Tabel yang pertama dialokasikan *edoc* dari 0 sampai 255 untuk *literal*, *edoc* 256 untuk *end-of-block*, dan *edoc* 257-285 untuk *length*. Kumpulan dari *edoc* yang berjumlah 29 terakhir tidak cukup untuk mewakili 256 kecocokan dari 3 sampai 258, sehingga bit ekstra akan ditambahkan ke beberapa *edoc*. Tabel 2.1 menunujukkan 29 daftar *edoc*, *extra* bit, dan *length* yang mewakili dari tabel *literal/length*.

Tabel 2.1 Tabel *Literal/length edoc* untuk mode 2

Code	Extra bits	*Length*s
257	0	3
258	0	4
259	0	5
260	0	6
261	0	7
262	0	8
263	0	9
264	0	10
265	1	11,12
266	1	13,14

Code	Extra bits	*Length*s
267	1	15,16
268	1	17,18
269	2	19 – 22
270	2	23 – 26
271	2	27 – 30
272	2	31 – 34
273	3	35 – 42
274	3	43 – 50
275	3	51 – 58
276	3	59 - 66

Code	Extra bits	*Length*s
277	4	67 – 82
278	4	83 – 98
279	4	99 – 114
280	4	115 – 130
281	5	131 - 162
282	5	163 - 194
283	5	195 - 226
284	5	227 - 257
285	0	258

Tabel 2.2 Huffman Codes untuk *edoc* mode 2

Edoc	Bits	Prefix codes
0 -143	8	00110000 – 10111111
144 – 255	9	110010000 – 111111111
256 – 279	7	0000000 – 0010111
280 – 287	8	11000000 - 11000111

Setiap *encoder* dan *decoder* deflate termasuk dari sebuah tabel yang memiliki *length* yang saling berurutan. Ada *edoc* dengan kecocokan *length* sampai 258, sehingga *look-ahead buffer* dari *encoder* deflate dapat memiliki ukuran maksimum hingga 258, tetapi bisa juga memiliki nilai yang lebih kecil dari 258.

Contoh nya jika 10 *string* simbol telah dicocokkan dengan algoritma LZ77, deflate akan mempersiapkan pasangan dari panjang dan jarak dimana *length* dari kecocokan yang awalnya 10 menjadi *edoc* 264, yang ditulis sebagai kode awalan 7 bit 0.001.000. Sebuah *length* 12 menjadi *edoc* 266 yang diikuti bit tunggal 1. Ini ditulis sebagai kode awalan 7-bit yang diikuti 0.001.010. *length* 20 dikonversi menjadi *edoc* 269 diikuti oleh dua bit 01. Ini ditulis sebagai bit sembilan 0001101|01. *Length* 256 menjadi *edoc* 284 diikuti oleh lima bit 11.110.ini ditulis sebagai 11000101 | 11.110. Kecocokan *length* 258 ditandai oleh *edoc* 285 yang 8-bit kode prefixny adalah 11.000.110. *Edoc* pada blok akhir 256 ditulis sebagai 7 bit 0.

Tiga puluh kode jarak yang tercantum pada tabel 2.3 adalah prefix code dengan ukuran yang telah ditetapkan 5-bit *prefixs* yang diikuti oleh bit ekstra untuk mewakili jarak pada interval [1,32768]. Ukuran maksimum *buffer* pencarian

adalah 32,768, namun ukuran tersebut dapat lebih kecil. Tabel 2.3 menunjukkan jarak dari 6 yang diwakili oleh 00100|1, jarak dari 21 menjadi kode 01000 | 101, dan jarak 8195 sesuai dengan kode 11010 | 00000000010.

Tabel 2.3 Tabel 30 jarak prefix code pada mode 2

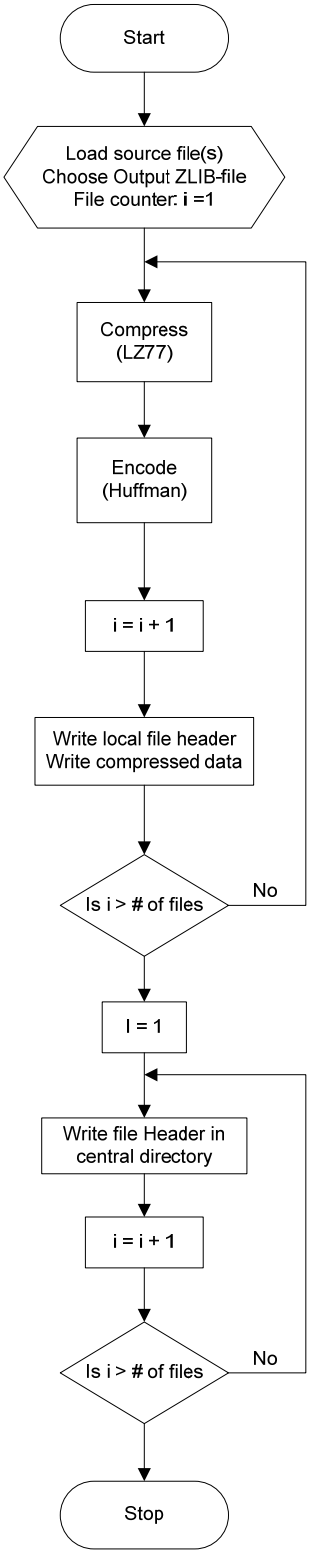
Code	Extra bits	Distance
0	0	1
1	0	2
2	0	3
3	0	4
4	1	5,6
5	1	7,8
6	2	9 – 12
7	2	13 – 16
8	3	17 – 24
9	3	25 - 32

Code	Extra bits	Distance
10	4	33 – 48
11	4	49 – 64
12	5	65 – 96
13	5	97 – 128
14	6	129 – 192
15	6	193 – 256
16	7	257 – 384
17	7	385 – 512
18	8	513 – 768
19	8	769 -1024

Code	Extra bits	Distance
20	9	1025 – 1536
21	9	1537 – 2048
22	10	2049 – 3072
23	10	3073 – 4096
24	11	4097 – 6144
25	11	6145 – 8192
26	12	8193 – 12288
27	12	12289 – 16384
28	13	16385 – 24576
29	13	24577 - 32768

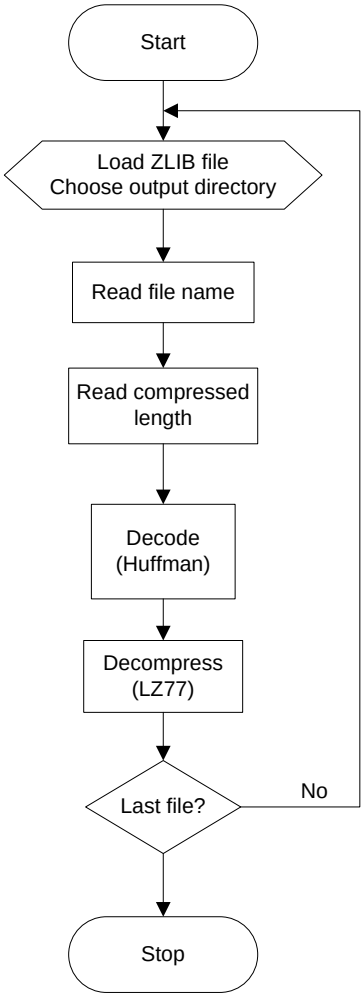
Proses dari Algoritma ZLIB ini dapat ditunjukkan dari diagram yang ditunjukkan pada gambar 2.10 dan 2.11, dimana kedua gambar ini menunjukkan diagram proses kompresi dan dekompresi Algoritma ZLIB.

1. Proses kompresi pada Algoritma ZLIB



Gambar 2.10 Proses Kompresi Algoritma ZLIB

2. Proses Dekompresi Algoritma ZLIB



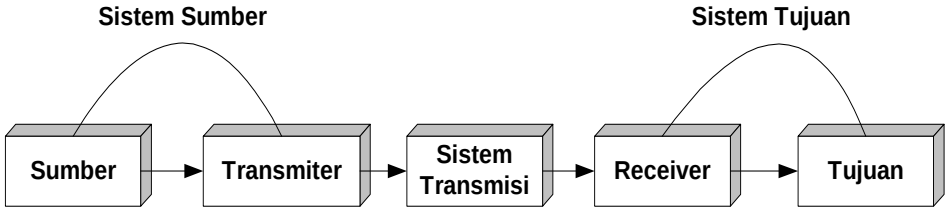
Gambar 2.11 Proses dekompresi Algoritma ZLIB

2.4 Komunikasi Data

Komunikasi data adalah proses pengiriman informasi diantara dua titik menggunakan kode biner melewati saluran transmisi dan peralatan *switching* yang dapat terjadi antara komputer dengan komputer, komputer dengan terminal atau komputer dengan peralatan. Komunikasi data merupakan gabungan dari teknik telekomunikasi dengan teknik pengolahan data.

2.4.1 Model komunikasi Data

Komunikasi data berkaitan dengan pertukaran data diantara dua perangkat yang terhubung secara langsung yang memungkinkan adanya pertukaran data antar kedua pihak. Gambar 2.3 menggambarkan proses komunikasi data.



Gambar 2.12 Proses Komunikasi Data

Pada gambar 2.3 terdapat elemen - elemen dalam kunci model tersebut :

1. *Source* (sumber) : Alat ini membangkitkan data sehingga dapat ditransmisikan, contoh : Telepon, *Personal Computer (PC)*.
2. *Transmitter* (pengirim) : Biasanya data yang dibangkitkan dari sumber tidak ditransmisikan secara langsung dalam bentuk aslinya. Sebuah *transmitter* cukup memindah dan menandai informasi dengan cara yang sama seperti sinyal – sinyal elektromagnetik yang dapat ditransmisikan melewati beberapa sistem transmisi berurutan.
3. Sistem transmisi : Berupa jalur transmisi tunggal (*single transmission*) atau jaringan komplek (*complex network*) yang menghubungkan antara sumber dengan tujuan (*destination*).
4. Tujuan (*destination*) : menangkap data yang dihasilkan oleh *receiver*.

2.4.2 Jaringan Komunikasi Data

Jaringan komunikasi dapat diartikan sebagai suatu sistem yang terbentuk dari interkoneksi fasilitas – fasilitas yang dirancang untuk membawa traffik dari berbagai sumber telekomunikasi.

Suatu jaringan terdiri dari *link* dan *node*. Istilah *node* digunakan untuk merepresentasikan sentral, *junction*, atau keduanya. Istilah *link* digunakan untuk merepresentasikan kabel, peralatan transmisi, dan sebagainya. Sedangkan traffik adalah informasi yang terdapat di dalam jaringan, yang mengalir melalui *link* dan *node*.

Suatu jaringan komunikasi merupakan sumber daya yang dapat dipakai secara bersamaan (*shared*) oleh sejumlah *end user* untuk berkomunikasi dengan *user* lain yang lokasinya berjauhan. Tidak semua *user* menggunakan jaringan pada waktu yang bersamaan, oleh karena itu merupakan suatu hal yang logis apabila sumber daya jaringan yang sangat penting ini dipakai bersama – sama. Penggunaan sumber daya secara bersamaan ini melahirkan konsep sentral.

1. LAN (*Local Area Network*)

LAN digunakan untuk menghubungkan komputer yang berada di dalam suatu area yang kecil, misalnya didalam suatu gedung perkantoran atau kampus. Jarak antar komputer yang dihubungkan bisa mencapai 5 sampai 10 km. suatu LAN biasanya bekerja pada kecepatan mulai 10 Mbps sampai 100 Mbps. LAN menjadi populer karena memungkinkan banyak pengguna untuk memakai sumber

daya yang dapat digunakan misalnya suatu *mainframe, file server,* printer dan sebagainya.

2. MAN (*Metropolitan Area Network*)

MAN merupakan suatu jaringan yang cakupannya meliputi suatu kota. MAN menghubungkan LAN – LAN yang lokasinya berjauhan. Jangkauan MAN meencapai 10 km sampai beberapa ratus km. suatu MAN biasanya bekeja pada kecepatan 1,5 sampai 150 Mbps.

3. WAN (*Wide Area Network*)

WAN dirancang untuk menghubungkan komputer – komputer yang terletak pada suatu cakupan geografis yang luas, seperti hubungan dari suatu kota ke kota yang lain didalam suatu negara. Cakupan WAN bisa meliputi 100 km sampai 1.000 km, dan kecepatan antar kota bisa bervariasi antara 1.5 Mbps sampai 2,4 Gbps. Dalam WAN, biaya untuk peralatan trasmisi sangat tinggi, dan biasanya jaringan WAN dimiliki dan dioperasikan sebagai suatu jaringan *public*.

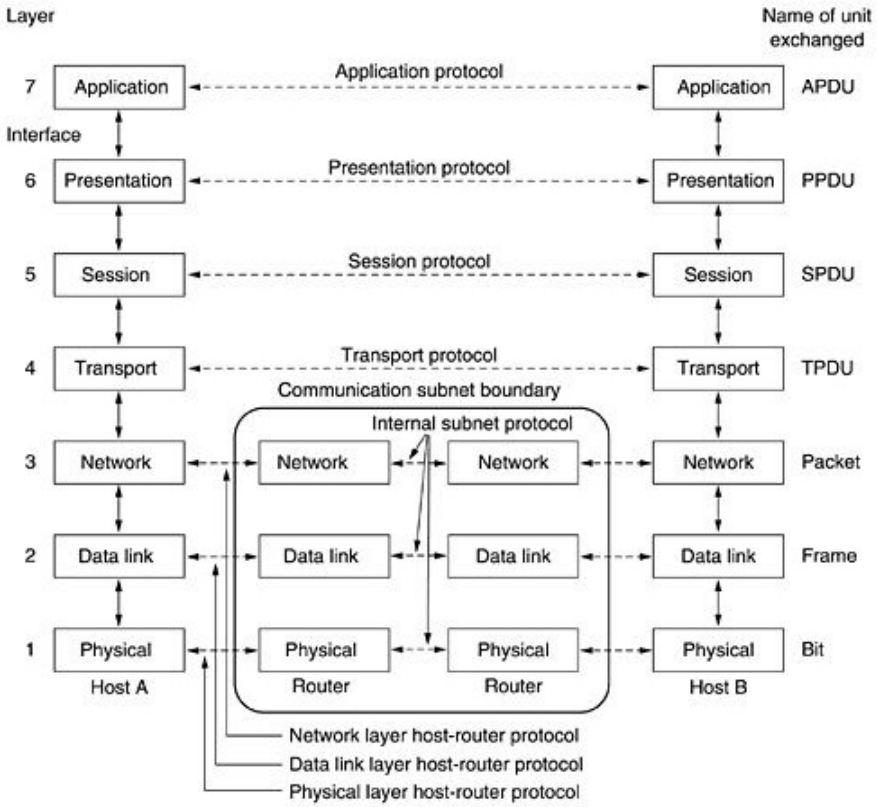
4. GAN (*Global Area Network*)

GAN merupakan suatu jaringan yang menghubungkan negara – negara diseluruh dunia. Kecepatan GAN bervariasi mulai dari 1,5 Mbps sampai dengan 100 Gbps dan cakupannya mencapai ribuan kilometer.

2.4.3 Arsitektur Protokol

Standar yang digunakan untuk melakukan pertukaran informasi diantara sistem – sistem yang terbuka yaitu sistem yang menerapkan standar OSI *(Open*

System Interconnection). Fakta bahwa suatu sistem merupakan sistem yang terbuka tidak berpengaruh terhadap implementasi, teknologi, atau interkoneksi sistem, melainkan berhubungan dengan kesesuaian suatu sistem terhadap standar yang diterapkan. Berdasarkan hal ini, ISO *(Organization for Standarization)* telah menspesifikasikan suatu model referensi OSI yang membagi – bagi fungsi komunikasi ke dalam tujuh lapisan *(layer).* Model OSI menyediakn secara konseptual kerangka kerja untuk komunikasi. Sebenarnya komunikasi dapat terjadi karena penggunaan protokol komunikasi. Dalam konteks jaringan (komunikasi data), sebuah protokol adalah suatu aturan formal dan kesepakatan yang menentukan bagaimana komputer bertukar informasi melalui sebuah media jaringan. Sebuah protokol mengimplementasikan salah satu atau lebih *layer – layer* OSI. Setiap *layer* ditugaskan untuk melaksanakan satu subset dari fungsi komunikasi tertentu yang pada gilirannya diterapkan didalam suatu *data terminal equipment (DTE)* yang berkomunikasi dengan DTE yang lain. Setiap *layer* mengandalkan *layer* yang ada dibawahnya untuk melaksanakan fungsi – fungsi yang lebih primitive serta sekaligus menyediakan layanan untuk mendukung *layer* yang berada diatasnya. *Layer – layer* ini dirancang agar setiap perubahan yang terjadi didalam suatu *layer* tidak mempengaruhi *layer* yang lainnya.



Gambar 2.13 Arsitektur Protokol

a. *Physical Layer*

Layer ini mendefinisikan media transmisi jaringan, metode pensinyalan, sikronisasi bit, arsitektur jaringan (*Ethernet* atau *Token Ring*), topologi jaringan dan pengabelan. Selain itu, *layer* ini mendefinisikan bagaimana *Network Interface Card* (NIC) dapat berinteraksi dengan media kabel atau radio.

b. *Data Link Layer*

Layer ini berfungsi untuk menentukan bagaimana bit – bit data dikelompokkan menjadi format yang disebut sebagai frame. Selain itu, pada level ini terjadi koreksi kesalahan, *flow control,* pengalamatan perangkat

keras (MAC Address) dan menentukan bagaimana perangkat perangkat jaringan seperti *hub*, *bridge, repeater,* dan *switch layer 2* beroperasi. Spesifikasi IEEE 802, membagi *layer* ini menjadi dua sub *layer*, *layer Logical Link Control (LLC)* dan *layer Media Access Control (MAC).*

c. *Network Layer*

Layer ini berfungsi untuk mendefinisikan alamat – alamat IP, membuat *header* untuk paket – paket, dan kemudian melakukan melakukan routing melalui *internetworking* dengan menggunakan *router* dan *switch layer-3.*

d. *Transport Layer*

Layer ini berfungsi untuk memecah data ke dalam paket – paket data serta memberikan nomor urut ke paket – paket tersebut sehingga dapat disusun kembali pada sisi tujuan setelah diterima. Selain itu, pada *layer* ini juga membuat sebuah tanda bahwa paket diterima dengan sukses (*acknowledgement*), dan mentransmisikan ulang terhadap paket – paket yang hilang ditengah jalan.

e. *Session Layer*

Layer ini mengijinkan para pengguna untuk menetapkan *session* dengan pengguna lainnya. *Layer* ini membuka, mengatur dan menutup suatu *session* antara aplikasi – aplikasi.

f. *Presentation Layer*

Layer ini berfungsi untuk mentranslasikan data yang hendak ditransmisikan oleh apikasi ke dalam format yang dapat ditransmisikan melalui jaringan.

g. *Application Layer*

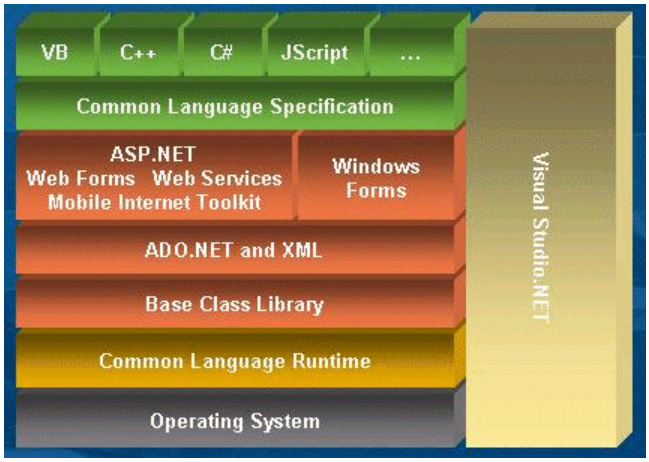
Layer ini berfungsi sebagai antarmuka dengan aplikasi dengan fungsionalitas jaringan, mengatur bagaimana aplikasi dapat mengakses jaringan, dan kemudian membuat pesan – pesan kesalahan. Protokol yang berada dalam lapisan ini adalah HTTP, SMTP, FTP, dan NFS.

2.5 .Net Framework

"*.Net Framework"* adalah suatu komponen *windows* yang terintegerasi yang dibuat dengan tujuan untuk mensupport pengembangan berbagai macam jenis aplikasi serta untuk men-*support* pengembangan berbagai macam jenis aplikasi serta untuk dapat menjalankan berbagai macam aplikasi generasi mendatang termasuk pengembangan *Web Service XML*.[7]

2.5.1 Arsitektur .Net Framework

".Net Framework" terdiri dari dua buah komponen utama, yaitu *Common Runtime Library (CLR)* dan "*.Net Framework" Class Library* atau kadang juga sering disebut dengan *Base Class Library (BCL).*



Gambar 2.14 Arsitektur .Net Framework

2.5.1.1 Common Language Runtime (CLR)

Common Language Runtime (CLR) adalah pondasi utama dari *".Net Framework"*. CLR merupakan komponen yang bertanggung jawab terhadap berbagai macam hal seperti bertanggung jawab untuk melakukan *management memory*, melakukan eksekusi kode, melakukan kompilasi terhadap keamanan kode, melakukan kompilasi kode dan berbagai layanan sistem lainnya.

Dengan adanya fungsi CLR ini, maka aplikasi berbasis *.NET* biasa juga disebut dengan *managed code,* sedangkan aplikasi diluar itu biasa disebut dengan *un-managed code.*

CLR akan melakukan kompilasi kode – kode aplikasi kita menjadi bahasa *assembly* MSIL (*Microsoft Intermediate Language*). Prose kompilasi ini sendiri dilakukan oleh komponen yang bernama *Just In Time* (JIT). JIT hanya akan mengkompilasi metode – metode yang memang digunakan dalam aplikasi dan hasil kompilasi ini sendiri di *chace* di dalam mesin dan akan di*compile* kembali jika memang ada perubahan pada aplikasi kita.

2.5.1.2 ".NET Framework" Class Library (Base Case Library)

".Net Framework" Class Library atau sering juga disebut *Base Case Library* (BCL) adalah koleksi dari *reusable types* yang sangat terintegrasi secara melekat dengan CLR. *Class library* bersifat berorientasi terhadap objek yang akan menyediakan *types* dari fungsi – fungsi *managed code*. Hal ini tidak hanya berpengaruh pada kemudahan dalam hal penggunaan, tetapi juga dapat mengurangi waktu yang diperlukan pada saat eksekusi. Dengan sifat tersebut maka komponen pihak ketiga akan dengan mudah diapikasikan kedalam aplikasi yang dibuat.

Dengan adanya BCL ini maka kita bisa menggunakan .NET Framework untuk membuat berbagai macam aplikasi, seperti :

1. Aplikasi *console*
2. Aplikasi berbasis *windows (Windows Form)*
3. Aplikasi ASP.NET (berbasis web)
4. Aplikasi Web Service XML
5. Aplikasi berbasis *Windows Services*

Jika kita membuat sekumpulan *class* untuk membuat aplikasi berbasis *windows,* maka *class – class* itu bisa kita gunakan untuk jenis aplikasi lain.

2.5.2 Microsoft Visual C#

Visual Studio C# (#, baca = *sharp*) adalah bahasa pemrograman baru yang diciptakan oleh *Microsoft* (dikembangkan dibawah kepemimpinan Anders Hejlsberg yang juga telah menciptakan berbagai macam bahasa pemrograman

termasuk *Borland Turbo C++* dan *Borland Delphi*). Bahasa C# juga telah distandarisasi secara internasional oleh ECMA.

Seperti halnya bahasa pemrograman yang lain, C# bisa digunakan untuk membangun berbagai macam jenis aplikasi, seperti aplikasi berbasis *windows desktop)* dan aplikasi berbasis *web* serta aplikasi berbasis *web services.*

C# bersifat sederhana, karena bahasa ini didasarkan kepada kepada bahasa C dan C++. Beberapa fungsi dari C# diadopsi langsung dari C dan C++ tetapi dengan perbaikan yang membuat bahasanya menjadi lebih sederhana.