Bab 3

Struktur Data untuk Citra Digital dan Format Citra *Bitmap*

Citra digital diolah dengan menggunakan komputer, oleh karena itu kita perlu mendefinisikan struktur data untuk merepresentasikan citra di dalam memori komputer. Matriks adalah struktur data yang tepat untuk merepresentasikan citra digital. Elemen-elemen matriks dapat diakses secara langsung melalui indeksnya (baris dan kolom).

Di dalam bab ini kita akan mendefinisikan struktur data matriks untuk citra digital. Notasi algoritmik yang kita gunakan untuk menjelaskan struktur data ini (beserta beberapa primitif operasi citra) adalah notasi Bahasa C (lebih tepatnya notasi ANSI C). Pemrograman citra digital lebih cocok menggunakan Bahasa C karena Bahasa C mempunyai penanganan tipe *pointer* yang lebih dinamis daripada Bahasa Pascal. Pada pembahasan nanti kita akan melihat bahwa struktur data matriks direpresentasikan dengan menggunakan tipe *pointer* mengingat ukuran matriks tidak diketahui sebelum pemrosesan citra digital.

3.1 Matriks

Sebagaimana telah dijelaskan pada Bab 2, citra digital yang berukuran $N \times M$ (tinggi = N, lebar = M) lazim dinyatakan dengan matriks N baris dan M kolom sebagai berikut:

$$f(x, y) \approx \begin{bmatrix} f(0,0) & f(0,1) & \ldots & f(0,M) \\ f(1,0) & f(1,1) & \ldots & f(1,M) \\ \vdots & \vdots & \vdots & \vdots \\ f(N-1,0) & f(N-1,1) & \ldots & f(N-1,M-1) \end{bmatrix}$$

Untuk citra dengan 256 derajat keabuan, harga setiap elemen matriks adalah bilangan bulat di dalam selang [0, 255]. Karena itu, kita dapat menggunakan tipe *unsigned char* untuk menyatakan tipe elemen matriks. *Unsigned char* adalah tipe *integer* positif di dalam Bahasa C yang rentang nilainya hanya dari 0 sampai 255. Kita dapat menggunakan matriks statik untuk merepresentasikan citra digital secara fisik di dalam memori komputer sebagai berikut:

```
unsigned char f[N][M];
```

dengan `N` dan `M` sudah terdefinisi sebelumnya sebagai suatu konstant. Elemen matriks diacu dengan `f[i][j]` , $0 \le i \le N-1$ dan $0 \le j \le M-1$.

Pada kebanyakan kasus, ukuran citra tidak diketahui sebelum pemrosesan dilakukan. Ada kemungkinan ukuran citra yang akan diolah melebihi nilai *N* dan *M* yang sudah ditetapkan di dalam deklarasi struktur data. Oleh karena itu, representasi citra dengan struktur matriks statik menjadi tidak relevan. Tipe data yang cocok untuk citra adalah *pointer*.

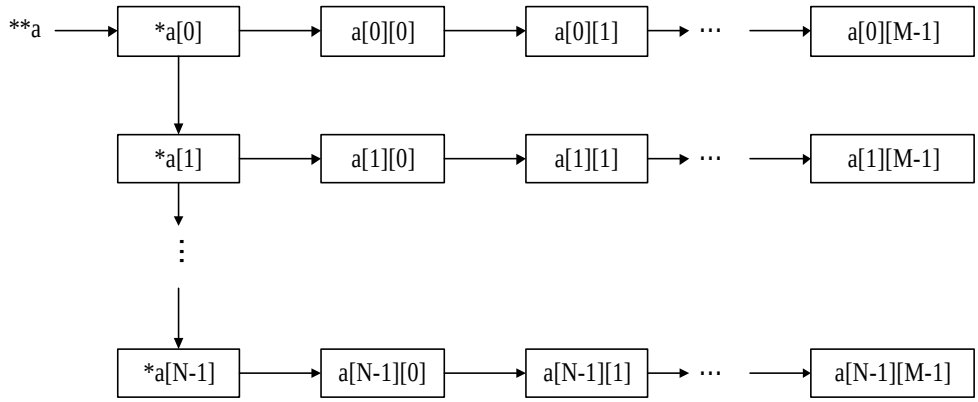
Tinjau tipe *pointer* untuk tabel atau larik (*array*). Di dalam Bahasa C, elemen larik `a[i]` ekivalen dengan `*(a+i)`. Tanda '*' menyatakan *pointer* yang menunjuk pada alamat suatu elemen di memori.

Larik yang berukuran *N* elemen dapat dibuat secara dinamik pada saat *run-time* dengan prosedur alokasi `malloc`:

```
unsigned char *a;
a=(unsigned char*) malloc (N * sizeof(unsigned char));
```

Pernyataan di atas berarti kita meminta komputer mengalokasikan memori untuk elemen larik `a` sebanyak *N* elemen, setiap elemen panjangnya 1 *byte* (yaitu panjang *byte* tipe *unsigned char*). Nilai *N* dapat ditetapkan pada saat *run-time*.

Matriks dapat dianggap sebagai larik satu matra (matra = dimensi) dari vektor seperti yang ditunjukkan pada Gambar 3.1. *Pointer* `**a` menunjuk ke larik *pointer* `*a[0]`, `*a[1]`, …, `*a[N-1]`, yang masing-masing larik menunjuk ke baris-baris dari citra.



Gambar 3.1. *Representasi matriks dengan larik pointer [PIT93]*

Dengan demikian, deklarasi matriks dinamis untuk citra `f` adalah sebagai berikut ini:

```
unsigned char **f;
```

atau dengan menggunakan *user defined type*:

```
typedef unsigned char **citra;
citra f;
```

Alokasi memori untuk matriks citra `f` yang berukuran $N \times M$ dilakukan pada saat *run-time* dengan memanggil fungsi `alokasi` seperti yang ditunjukkan oleh Algoritma 3.1.

```
citra alokasi(int N, int M)
/* Mengalokasikan memori untuk citra f yang berukuran N x M pixel. */
{
  int i;

  f=(unsigned char**)malloc(N * sizeof(unsigned char*));
  if (f==NULL) return(NULL);   /* memori habis */
  for (i=0; i<N; i++)
  {
    f[i]=(unsigned char*)malloc(M*sizeof(unsigned char));
    if (f[i]==NULL)
    { /* memori habis, dealokasi semua elemen baris matriks */
      dealokasi(f, N);
      return(NULL);
    }
  }
  return f;
}
```

Algoritma 3.1. *Alokasi memori untuk matriks citra f*

Cara pemanggilan fungsi `alokasi` adalah dengan menampung keluaran fungsi dalam peubah yang tipenya sama dengan tipe `citra`:

```
f = alokasi(N, M)
```

Untuk citra berwarna, yang mana setiap nilai intensitas merah, hijau, dan biru disimpan di dalam matriks `r`, `g`, dan `b`, maka kita harus mengalokasikan memori untuk ketiga buah matriks tersebut:

```
r = alokasi(N, M)
g = alokasi(N, M)
b = alokasi(N, M)
```

Bila citra selesai diproses, maka memori yang dipakai oleh citra tersebut dikembalikan (dealokasi) kepada sistem. Dealokasi memori untuk matriks citra `f` dapat dilakukan dengan memanggil fungsi `dealokasi` pada Algoritma 3.2.

```
void dealokasi(citra f, int N)
/* Dealokasi memori dari citra f yang mempunyai N baris pixel */

{
  int i;

  for (i=0; i<N; i++)
  {
    free(f[i]); /* bebaskan memori semua elemen pada baris i */
  }
  free(f);
}
```

Algoritma 3.2. *Dealokasi memori untuk dari citra f*

Selain matriks yang merepresentasikan citra, kita mungkin memerlukan matriks lain yang menyimpan nilai-nilai bertipe riil dan matriks yang menyimpan nilai-nilai *integer* (yang rentang nilainya lebih besar daripada *unsigned char*). Oleh karena itu, kita perlu mendedinisikan dua tipe matriks lain sebagai berikut:

```
typedef float **rmatrik;
typedef int **imatriks;
```

Selain menggunakan Algoritma 3.1, kita juga dapat menggunakan algoritma pengalokasian memori untuk matriks secara umum (jadi, tidak hanya untuk matriks citra yang bertipe *unsined char* saja) yang dibentuk pada saat *run-time*. Algoritma pengalokasian memori matriks tersebut ditunjukkan pada Algoritma 3.3. Pada prinsipnya, cara pengalokasian memori untuk matriks pada algoritma 3.3 sama saja dengan Algoritma 3.1, hanya saja Algoritma 3.3 dapat digunakan untuk mengalokasikan memori matriks yang bertipe apa saja. Dengan demikian,

kita tidak perlu menulis tiga prosedur yang berbeda untuk mengalokasikan matriks yang bertipe berbeda pula.

```
void **alokasi(int N, int M, int UkuranElemen)
/* Mengalokasikan memori untuk matriks yang berukuran N x M. Setiap elemen
matriks membutuhkan ruang memori sebesar UkuranElemen byte  */
{
  int i;
  void **larik = (void**)xalloc(N * sizeof(void *)); /* buat array N
                                                        elemen */
  for (i=0; i<N; i++)
    larik[i] = (void*)xalloc(M * UkuranElemen);
  return larik;
}


void *xalloc(unsigned ukuran)
/* Mengalokasikan memori dan memeriksa apakah alokasi memori berhasil */
{
  void *p = malloc(ukuran);
  if (p==NULL)
  {
    printf("Memori tidak cukup untuk alokasi matriks");
    exit(0);
  }
  return p;
}
```

Algoritma 3.3. *Alokasi memori untuk matriks bertipe sembarang*

Untuk mengalokasikan memori untuk citra *Image* yang berukuran $N \times M$ elemen dan setiap elemen bertipe *unsigned char*, perintahnya di dalam program adalah sebagai berikut:

```
Image = (unsigned char**) alokasi (N, M, sizeof(unsined char));
```

Sedangkan untuk mengalokasikan memori matriks *Mat* yang berukuran $N \times M$ elemen dan setiap elemen bertipe *float*, perintahnya di dalam program adalah sebagai berikut:

```
Mat = (float**) alokasi (N, M, sizeof(float));
```

Sebagai catatan, semua algoritma alokasi dan dealokasi matriks yang berbasis stuktur data *pointer* harus menyertakan pustaka *alloc.h* di dalam programnya:

```
#include <alloc.h>
```

3.2 Menampilkan Citra ke Layar

Citra ditampilkan ke layar peraga jika *card* grafik tersedia pada komputer yang digunakan dan *card* tersebut mampu menghasilkan warna untuk setiap komponen *RGB* (*Red*, *Green*, *Blue*). Fungsi baku untuk menampilkan citra adalah `setpixel`:

```
setpixel(unsigned char r,unsigned char g, unsigned char b,
         int i,int j);
/* menampilkan pixel dengan komponen rgb pada koordinat i, j */
```

Prosedur menampilkan citra berwarna yang setiap intensitas merah, hijau, dan biru disimpan di dalam matriks `r`, `g`, dan `b` selengkapnya ditunjukkan pada Algoritma 3.4.

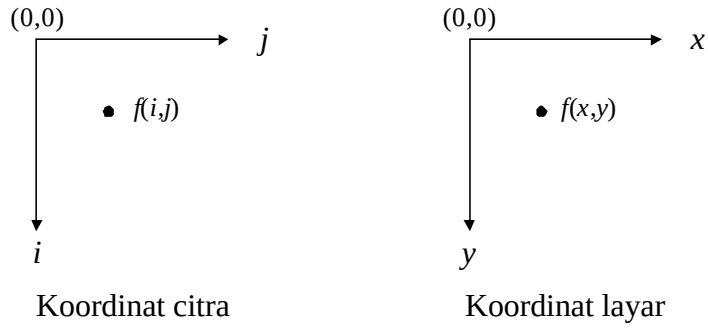
```
void tampilkan_citra(citra r,citra g,citra b, int N,int M)
/* Menampilkan citra yang berukuran N x M pixel ke layar. */
{ int i, j;

  for (i=0; i<N; i++)
     for (j=0; j<M; j++)
        setpixel(r[i][j],g[i][j],b[i][j],j,i);
}
```

Algoritma 3.4. *Prosedur menampilkan citra ke layar*

Jika citra yang ditampilkan adalah citra hitam-putih (matriks intensitas *pixel*-nya adalah *f*), maka perubahan yang dilakukan adalah pada:

```
setpixel(f[i][j],f[i][j],f[i][j], j, i);
```

Pixel (*i*,*j*) ditampilkan pada posisi (*j*,*i*) di layar karena perbedaan sistem koordinat yang digunakan pada representasi citra dan layar peraga (lihat Gambar 3.2).



Gambar 3.2. *Perbedaan antara koordinat citra dan koordinat layar*

Prosedur menampilkan citra pada *platform* Windows berbeda dengan prosedur di atas. Algoritma 3.5 berikut ini adalah prosedur menampilkan citra hitam-putih dengan 256 derajat keabuan di lingkungan Windows.

```
void WIN_tampilkan_citra(citra Image, int N, int M)
/* Menampilkan citra Image yang berukuran N x M di lingkungan Windows */
{
  HDC      MemDC;      /* Handle ke memory device context */
  HBITMAP  mbitmap;    /* Handle ke citra */
  HWND     hwnd;       /* Handle ke window */
  COLORREF TabelWarna[256];   /* Tabel warna (palet) */
  int i, j, palet;

  hwnd = GetActiveWindow();
  MemDC = CreateCompatibleDC(GetDC(hwnd));
  mbitmap = CreateCompatibleBitmap(GetDC(hwnd),M,N);
  SelectObject(MemDC,mbitmap);

  /* Definisikan palet */
  for (i=0; i<256; i++)
    TabelWarna[i]=GetNearestColor(MemDC, RGB(i,i,i));

  /* Isikan pixel ke memori device (layar) */
  for (i=0; i<N; i++)
    for (j=0; j<M; j++)
     {
       palet = Image[i][j];
       SetPixelV(MemDC,j,i,TabelWarna[palet]);
     }

  /* Tembakkan citra ke layar */
  BitBlt(GetDC(hwnd),0,0,M,N,MemDC,0,0,SRCCOPY);
}
```

Algoritma 3.5. *Prosedur menampilkan citra hitam-putih ke layar di lingkungan Windows.*

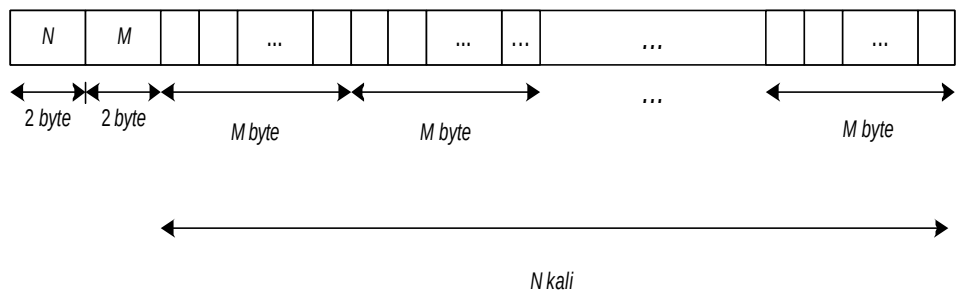
Jika citra yang ditampilkan adalah citra berwarna (matriks intensitas *pixel*-nya masing-masing adalah *r*, *g*, dan *b*), maka perubahan yang dilakukan adalah sebagai berikut:

```
for (i=0; i<N; i++)
 for (j=0; j<M; j++)
 {
   palet = GetNearestColor(MemDC, RGB(b[i][j], g[i][j], r[i][j]);
   SetPixelV(MemDC, j, i, palet);
 }

/* Tembakkan citra ke layar */
BitBlt(GetDC(hwnd), 0, 0, M, N, MemDC, 0, 0, SRCCOPY);
```

3.3 Membaca Citra dari Arsip

Citra disimpan di dalam arsip biner untuk sewaktu-waktu dibuka dan dibaca kembali. Arsip tersebut ada yang mempunyai *header* dan ada yang tanpa *header*. *Header* adalah informasi yang terletak pada bagian awal arsip. *Header* berisi informasi bagaimana citra disimpan. Kita perlu mengetahui *header* agar kita mengetahui cara membaca data citra. Saat ini terdapat bermacam-macam format penyimpanan citra di dalam arsip (misalnya BMP, GIF, TIFF, PCX, JPG, dan lain-lain). Dua format yang populer saat ini dan seolah-olah menjadi standard adalah GIF dan JPG. Pembahasan format-format citra di dalam arsip diluar cakupan buku ini, namun khusus format *bitmap* (BMP) akan diberikan di dalam Bab 3 ini sebagai satu studi kasus.

Algoritma 3.6 berisi prosedur untuk membaca citra hitam-putih (*greyscale*) dari dalam arsip yang berisi data citra mentah (*raw image*). Disebut data citra mentah karena arsip tersebut hanya berisi ukuran citra dan nilai keabuan setiap *pixel*. Tidak ada informasi lain di dalam arsip selain ukuran citra. Ukuran citra panjangnya 32 bit, masing-masing 16 bit (atau 2 *byte*) untuk tinggi (*N*) dan 16 bit untuk lebar (*M*). *Byte-byte* berikutnya berisi nilai keabuan setiap *pixel* di dalam citra. Setiap *pixel* berukuran 1 *byte*. Gambar 3.3 memperlihatkan susunan *byte* di dalam arsip yang berisi data citra mentah.



Gambar 3.3. *Susunan data di dalam arsip data citra mentah*

Mula-mula kita membaca data tinggi (*N*) dan lebar (*M*) citra. Kita bisa menggunakan peubah *N* dan *M* yang bertipe *unsigned short int* karena tipe ini panjangnya 2 *byte*. Selanjutnya, data *pixel-pixel* di dalam citra dibaca "baris per baris". Setiap baris panjangnya *M byte*. Setiap *byte* ke-*j* dari baris *i* menyatakan nilai *pixel* pada koordinat (*i*, *j*). Nilai ini disimpan pada elemen matriks *f*[*i*][*j*].

```
void baca_citra_dari_arsip(char nama_arsip[], citra f)
/* Membaca citra dari arsip nama_arsip. Citra hasil pembacaan disimpan di
   dalam matriks f.
*/
{
  FILE *fp;
  int i, j;
  unsigned short int N, M;

  if((fp=fopen(nama_arsip, "rb"))==NULL)
  {
   printf("Arsip tidak ada");
   exit(0);
  }
  fread(&N, sizeof(unsigned short int), 1, fp);  /* baca tinggi citra */
  fread(&M, sizeof(unsigned short int), 1, fp);  /* baca lebar citra */

  f = alokasi(N, M)  /* alokasi memori matriks untuk citra f */
  if(f==NULL)
  { printf("Memori tidak cukup");
    exit(0);
  }

  /* baca data citra baris demi baris */
  for(i=0; i<N; i++)
  {
    /* baca data citra baris ke-i */
    fread(f[i], sizeof(citraunsigned char), M, fp);
  }
  close(fp);
}
```

Algoritma 3.6. *Prosedur membaca citra dari arsip.*

3.4 Menyimpan Citra ke dalam Arsip

Operasi menyimpan citra ke dalam arsip merupakan kebalikan dari operasi arsip. Citra disimpan di dalam dengan susunan yang sama seperti pada Gambar 3.3. Algoritma 3.7 berisi prosedur menyimpan citra ke dalam arsip. Ukuran citra (*N* dan *M*) disimpan pada awal arsip. Selanjutnya, data citra disimpan baris per baris *pixel*.

```
void tulis_citra_ke_arsip(char nama_arsip[], citra f)
/* Menulis citra f ke dalam arsip nama_arsip. */
{
  FILE *fp;
  int i, j;
  unsigned short int N, M;

  if((fp=fopen(nama_arsip, "wb"))==NULL)
  {
   printf("Arsip tidak dapat dibuat");
```

```
    exit(0);
  }
  fwrite(N, sizeof(unsigned short int), 1, fp);  /* tulis tinggi citra */
  fwrite(M, sizeof(unsigned short int), 1, fp);  /* tulis lebar citra */

  /* baca data citra baris demi baris */
  for(i=0; i<N; i++)
  {
    /* tulis data citra baris ke-i */
    fwrite(f[i], sizeof(unsigned char), M, fp);
  }
  close(fp);
}
```

Algoritma 3.7. *Prosedur menulis citra ke dalam arsip.*

3.5 Format Berkas *Bitmap*

Citra disimpan di dalam berkas (*file*) dengan format tertentu. Format citra yang baku di lingkungan sistem operasi Microsoft Windows dan IBM OS/2 adalah berkas *bitmap* (BMP). Saat ini format BMP memang "kalah" populer dibandingkan format JPG atau GIF. Hal ini karena berkas BMP pada umumnya tidak dimampatkan, sehingga ukuran berkasnya relatif lebih besar daripada berkas JPG maupun GIF. Hal ini juga yang menyebabkan format BMP sudah jarang digunakan.

Meskipun format BMP tidak mangkus dari segi ukuran berkas, namun format BMP mempunyai kelebihan dari segi kualitas gambar. Citra dalam format BMP lebih bagus daripada citra dalam format yang lainnya, karena citra dalam format BMP umumnya tidak dimampatkan sehingga tidak ada informasi yang hilang (pemampatan citra dibahas secara mendalam di dalam Bab 10). Terjemahan bebas *bitmap* adalah pemetaan bit. Artinya, nilai intensitas *pixel* di dalam citra dipetakan ke sejumlah bit tertentu. Peta bit yang umum adalah 8, artinya setiap *pixel* panjangnya 8 bit. Delapan bit ini merepresentasikan nilai intensitas *pixel*. Dengan demikian ada sebanyak $2^8 = 256$ derajat keabuan, mulai dari 0 sampai 255.

Citra dalam format BMP ada tiga macam: citra biner, citra berwarna, dan citra hitam-putih (*greyscale*). Citra biner hanya mempunyai dua nilai keabuan, 0 dan 1. Oleh karena itu, 1 bit sudah cukup untuk merepresentasikan nilai *pixel*. Citra berwarna adalah citra yang lebih umum. Warna yang terlihat pada citra *bitmap* merupakan kombinasi dari tiga warna dasar, yaitu merah, hijau, dan biru. Setiap *pixel* disusun oleh tiga komponen warna: *R* (*red*), *G* (*green*), dan *B* (*blue*). Kombinasi dari ketiga warna *RGB* tersebut menghasilkan warna yang khas untuk *pixel* yang bersangkutan. Pada citra 256 warna, setiap *pixel* panjangnya 8 bit, tetapi komponen warna *RGB*-nya disimpan di dalam tabel *RGB* yang disebut

palet. Setiap komponen panjangnya 8 bit, jadi ada 256 nilai keabuan untuk warna merah, 256 nilai keabuan untuk warna hijau, dan 256 nilai keabuan untuk warna biru. Nilai setiap *pixel* tidak menyatakan derajat keabuannya secara langsung, tetapi nilai *pixel* menyatakan indeks tabel *RGB* yang memuat nilai keabuan merah (*R*), nilai keabuan hijau (*G*), dan nilai keabuan biru (*B*) untuk *pixel* yang bersangkutan. Pada citra hitam-putih, nlai $R = G = B$ untuk menyatakan bahwa citra hitam-putih hanya mempunyai satu kanal warna. Citra hitam-putih umumnya adalah cira 8-bit.

Citra yang lebih kaya warna adalah citra 24-bit. Setiap *pixel* panjangnya 24 bit, karena setiap *pixel* langsung menyatakan komponen warna merah, komponen warna hijau, dan komponen warna biru. Masing-masing komponen panjangnya 8 bit. Citra 24-bit disebut juga citra 16 juta warna, karena ia mampu menghasilkan 2^{24} = 16.777.216 kombinasi warna.

Saat ini beredar tiga versi berkas *bitmap*, (i) berkas *bitmap* versi lama dari *Microsoft Windows* atau IBM OS/2, (ii) berkas *bitmap* versi baru dari Microsoft Windows, dan (iii) berkas *bitmap* versi baru dari IBM OS/2. Yang membedakan ketiga versi berkas tersebut adalah panjang *header*-nya. *Header* adalah data yang terdapat pada bagian awal berkas citra. Data di dalam *header* berguna untuk mengetahui bagaimana citra dalam format *bitmap* dikodekan dan disimpan. Data di dalam *header* misalnya ukuran citra, kedalaman *pixel*, ofset ke data *bitmap*, dan sebagainya.

Setiap berkas *bitmap* terdiri atas *header* berkas, *header bitmap*, informasi palet, dan data *bitmap* (Gambar 3.4).

Header berkas	*Header* bitmap	Informasi palet	Data *bitmap*
← 14 *byte* →←	12 s/d 64 *byte* →←	0 s/d 1024 *byte* →←	*N byte* →

Gambar 3.4. *Format berkas bitmap*

Ukuran *header* berkas sama untuk semua versi, yaitu 14 *byte*. Tabel 3.1 memperlihatkan *field-field* penyusun *header* berkas. Ukuran *header bitmap* berbeda-beda untuk setiap versi. Untuk berkas *bitmap* versi lama, *header bitmap* berukuran 12 *byte* (Tabel 3.2), untuk berkas *bitmap* versi baru dari *Microsoft Windows*, *header bitmap* berukuran 40 (Tabel 3.3), dan untuk berkas *bitmap* versi baru dari IBM OS/2, *header bitmap* berukuran 64 *byte* (Tabel 3.4).

Tabel 3.1. *Header berkas bitmap (panjang = 14 byte)*

Byte ke-	Panjang (*byte*)	Nama	Keterangan
1 – 2	2	BmpType	Tipe berkas bitmap: BA = *bitmap array*, CI = *icon* BM = *bitmap*, CP = *color pointer* PT = *pointer*
3 – 6	4	*BmpSize*	Ukuran berkas *bitmap*
7 – 8	2	*XhotSpot*	X *hotspot* untuk kursor
9 – 10	2	*YhotSpot*	Y *hotspot* untuk kursor
11 – 14	4	*OffBits*	Ofset ke awal data *bitmap* (dalam *byte*)

Tabel 3.2. *Header bitmap versi lama dari Microsoft Windows (12 byte)*

Byte ke-	Panjang (*byte*)	Nama	Keterangan
1 – 4	4	HdrSize	Ukuran *header* dalam satuan *byte*
5 – 6	2	*Width*	Lebar *bitmap* dalam satuan *pixel*
7 – 8	2	*Height*	Tinggi *bitmap* dalam satuan *pixel*
9 – 10	2	*Planes*	Jumlah *plane* (umum- nya selalu satu)
11 – 12	2	*BitCount*	Jumlah bit per *pixel*

Tabel 3.3. *Header bitmap versi baru dari Microsoft Windows (40 byte)*

Byte ke-	Panjang (*byte*)	Nama	Keterangan
1 – 4	4	HdrSize	Ukuran *header* dalam satuan *byte*
5 – 8	4	*Width*	Lebar *bitmap* dalam satuan *pixel*
9 – 12	4	*Height*	Tinggi *bitmap* dalam satuan *pixel*
13 – 14	2	*Planes*	Jumlah *plane* (umum- nya selalu satu)
15 – 16	2	*BitCount*	Jumlah bit per *pixel*
17 – 20	4	*Compression*	0=tak dimampatkan, 1=dimampatkan
21 – 24	4	*ImgSize*	Ukuran *bitmap* dalam *byte*
25 – 28	4	*HorzRes*	Resolusi horizontal
29 – 32	4	*VertRes*	Resolusi vertikal
33 – 36	4	*ClrUsed*	Jumlah warna yang digunakan
37 – 40	4	*ClrImportant*	Jumlah warna yang penting

Tabel 3.4. *Header bitmap versi baru dari IBM OS/2 (64 byte)*

Byte ke-	Panjang (*byte*)	Nama	Keterangan
1 – 4	4	HdrSize	Ukuran *header* dalam satuan *byte*
5 – 8	4	*Width*	Lebar *bitmap* dalam satuan *pixel*
9 – 12	4	*Height*	Tinggi *bitmap* dalam satuan *pixel*
13 – 14	2	*Planes*	Jumlah *plane* (umumnya selalu satu)
15 – 16	2	*BitCount*	Jumlah bit per *pixel*
17 – 20	4	*Compression*	0 = tak dimampatkan, 1 = dimampatkan
21 – 24	4	*ImgSize*	Ukuran *bitmap* dalam *byte*
25 – 28	4	*HorzRes*	Resolusi horizontal
29 – 32	4	*VertRes*	Resolusi vertikal
33 – 36	4	*ClrUsed*	Jumlah warna yang digunakan
37 – 40	4	*ClrImportant*	Jumlah warna yang penting
41 – 42	2	*Units*	Satuan pengukuran yang dipakai
43 – 44	2	*Reserved*	*Field* Cadangan
45 – 46	2	*Recording*	Algoritma perekaman
47 – 48	2	*Rendering*	Algoritma *halftoning*
49 – 52	4	*Size1*	Nilai ukuran 1
53 – 56	4	*Size2*	Nilai ukuran 2
57 – 60	4	*ClrEncoding*	Pengkodean warna
61 – 64	4	Identifier	Kode yang digunakan aplikasi

Informasi palet warna terletak sesudah *header bitmap*. Informasi palet warna dinyatakan dalam suatu tabel *RGB*. Setiap *entry* pada tabel terdiri atas tiga buah *field*, yaitu *R* (*red*), *G* (*green*), dan *B* (*blue*).

Data *bitmap* diletakkan sesudah informasi palet. Penyimpanan data *bitmap* di dalam berkas disusun terbalik dari bawah ke atas dalam bentuk matriks yang berukuran *Height* × *Width*. Baris ke-0 pada matriks data *bitmap* menyatakan data *pixel* di citra baris terbawah, sedangkan baris terakhir pada matriks menyatakan data *pixel* di citra baris teratas.

Contoh format citra 8-bit kira-kira seperti Gambar 3.5. Format citra 4-bit (16 warna) serupa dengan format citra 8-bit. Pada citra 4-bit dan citra 8-bit, warna suatu *pixel* diacu dari tabel informasi palet pada *entry* ke-*k* (*k* merupakan nilai dengan rentang 0 – 15 untuk citra 16 warna dan 0 – 255 untuk citra 256 warna). Sebagai contoh pada Gambar 3.5, *pixel* pertama bernilai 2; warna *pixel* pertama ini ditentukan oleh komponen RGB pada tabel palet warna *entry* ke-2, yaitu *R* = 14, *G* = 13, dan *B* = 16. *Pixel* kedua serupa dengan *pixel* pertama. *Pixel* ketiga bernilai 1, warnanya ditentukan oleh komponen RBG pada tabel warna *entry* ke-1, yaitu *R* = 20, *G* = 45, dan *B* = 24. Demikian seterusnya untuk *pixel-pixel* lainnya. Khusus untuk citra hitam-putih (8 bit), komponen *R*, *G*, dan *B* suatu *pixel*

bernilai sama dengan data *bitmap pixel* tersebut. Jadi, *pixel* dengan nilai data *bitmap* 129, memiliki nilai $R = 129$, $G = 129$, dan $B = 129$.

<*header berkas*>
<*header bitmap*>

<*palet warna RGB*>

	R	*G*	*B*
1	20	45	24
2	14	13	16
3	12	17	15
…			
256	46	78	25

<*data bitmap*>
2 2 1 1 1 3 5 …

Gambar 3.5. *Format citra 8-bit (256 warna)*

Berkas citra 24-bit (16,7 juta warna) tidak emmpunyai palet *RGB*, karena nilai *RGB* langsung diuraikan dalam data *bitmap*. Setiap elemen data *bitmap* panjangnya 3 *byte*, masing-masing *byte* menyatakan komponen *R*, *G*, dan *B*. Contoh format citra 24-bit (16 juta warna) kira-kira seperti pada Gambar 3.6. Pada contoh format citra 24-bit tersebut *pixel* pertama mempunyai $R = 20$, $G = 19$, $B = 21$, *pixel* kedua mempunyai $R = 24$, $G = 24$, $B = 23$. Demikian seterusnya

<*header* berkas>
<*header bitmap*>

<data *bitmap*>
20 19 21 24 24 23 24 …

Gambar 3.6. *Format citra 24-bit (16,7 juta warna)*

Tabel 3.5 memperlihatkan panjang informasi palet untuk setiap versi *bitmap*, masing-masing untuk citra 16 warna, 256 warna, dan 16,7 juta warna.

Tabel 3.5. *Panjang informasi palet untuk setiap versi berkas bitmap*

Citra *m* warna	Versi lama (*Windows*)	Versi baru (*Windows*)	Versi baru (OS/2)
Citra 16 warna	48 *byte*	64 *byte*	64 *byte*
Citra 256 warna	768 *byte*	1024 *byte*	1024 *byte*
Citra 16,7 juta warna	0 *byte*	0 *byte*	0 *byte*

3.6 Primitif Citra *Bitmap*

Meskipun saat ini kakas pemrograman visual (*visual programming*) seperti *Visual C++*, *Delphi*, *Borland C++ Builder*, dan lain-lain, sudah banyak memberikan kemudahan untuk memprogram citra *bitmap* (dengan menyediakan komponen, struktur data, dan metode-metode untuk mengkases citra *bitmap*), namun ada baiknya kita membuat sendiri primitif-primitif citra *bitmap*. Primitif tersebut adalah berupa fungsi dan prosedur untuk membaca citra dari arsip, menyimpan citra ke dalam arsip, dan menampilkan citra ke layar.

Berikut ini diberikan beberapa primitif citra *bitmap*. Citra *bitmap* yang ditangani hanyalah citra dengan kedalaman 8-bit (256 warna) dan citra 24-bit (16,7 juta warna), baik berupa citra skala-abu maupun citra berwarna. Citra biner (1-bit) tidak ditangani di sini karena citra biner dimampatkan dengan metode *RLE* (*Run Length Encoding*). Hingga bab 3 ini kita belum membicarkan metode pemampatan citra. Pembahasan mengenai metode RLE dapat anda baca di dalam Bab 10.

1. Berkas *header*
   Nama arsip: `bitmap.h`
   Penjelasan: Berkas *header* berisi struktur data untuk citra *bitmap*

```
#ifndef bitmap_H
#define bitmap_H

#include <stdio.h>
#include <io.h>
#include <stdlib.h>
#include <alloc.h>


/* maksimum derajat keabuan */
#define  DERAJAT_KEABUAN   255

/* tipe data citra bitmap 8-bit */
#define TIPE_CITRA   unsigned char
```

```
/* tipe data citra bitmap 24-bit */
typedef struct
{
  unsigned char R;
  unsigned char G;
  unsigned char B;
} TIPE_CITRA24BIT;

TIPE_CITRA      **Image;        /* Matriks bitmap citra 8-bit */
TIPE_CITRA24BIT **Image24bit;   /* Matriks bitmap citra 24-bit */

/* HEADER BERKAS CITRA BMP, ada dua macam, yaitu header arsip
   dan header data bitmap (BMP)  */

/* 1. Header arsip */
unsigned char      TipeBMP1;
unsigned char      TipeBMP2;
unsigned long      UkuranBMP;
unsigned short int XhotSpot;
unsigned short int YhotSpot;
unsigned int       OffBits;

/* 2.Header data bitmap
     a. Ukuran header data BMP: 12 byte (versi lama), 40 byte (versi baru
        Windows), dan 64 byte (versi OS 2)
*/
unsigned long UkuranHeader;


/*   b.1. Header data BMP versi lama */
unsigned short int LebarBMPvLama;
unsigned short int TinggiBMPvLama;
unsigned short int BidangBMPvLama;
unsigned short int JumlahBitBMPvLama;

/*   b.2. Header data BMP versi Windows */
unsigned int       LebarBMPvBaru;
unsigned int       TinggiBMPvBaru;
unsigned short int BidangBMPvBaru;
unsigned short int JumlahBitBMPvBaru;
unsigned int       MampatBMPvBaru;
unsigned int       UkuranCitraBMPvBaru;
unsigned int       ResolusiHorizontalBMPvBaru;
unsigned int       ResolusiVertikalBMPvBaru;
unsigned int       WarnaTerpakaiBMPvBaru;
unsigned int       WarnaPentingBMPvBaru;

/*   b.3. Header data BMP versi OS 2  */
unsigned int       LebarBMPvOS2;
unsigned int       TinggiBMPvOS2;
unsigned short int BidangBMPvOS2;
unsigned short int JumlahBitBMPvOS2;
unsigned int       MampatBMPvOS2;
unsigned int       UkuranCitraBMPvOS2;
unsigned int       ResolusiHorizontalBMPvOS2;
unsigned int       ResolusiVertikalBMPvOS2;
unsigned int       WarnaTerpakaiBMPvOS2;
unsigned int       WarnaPentingBMPvOS2;
unsigned short int UnitBMPvOS2;
unsigned short int CadanganBMPvOS2;
unsigned short int PerekamanBMPvOS2;
unsigned short int RenderingBMPvOS2;
```

```
unsigned int       Ukuran1BMPvOS2;
unsigned int       Ukuran2BMPvOS2;
unsigned int       PengkodeanWarnaBMPvOS2;
unsigned int       PenciriBMPvOS2;

/* 3. Informasi palet */
typedef struct
{
  unsigned char R;
  unsigned char G;
  unsigned char B;
} RGBvLama;

typedef struct
{
  unsigned char R;
  unsigned char G;
  unsigned char B;
  unsigned char cadangan;
} RGBvBaru;

/* peubah-peubah bantu */
RGBvLama *paletVlama;
RGBvBaru *paletVbaru;
int lebar, tinggi, jumlahpalet, jumlahbit;
char* NamaArsip;

/* Purwarupa fungsi yang digunakan */
void **alokasi(int N, int M, int UkuranElemen);
void *xalloc(unsigned ukuran);
void BacaBerkasCitra(char *NamaArsip);
void AlokasiMemoriCitra(int N, int M);
void AlokasiMemoriCitra24Bit(int N, int M);
void BacaHeader(FILE *fp);
void BacaDataBitmap(FILE *masukan, N, int M);
void TampilkanCitra(int N, int M);
void DealokasiMemoriCitra(int N, int jumlahbit);

#endif
```

Algoritma 3.8. *Berkas* `bitmap.h`.

2. Membaca citra dari arsip.
   Nama fungsi: `BacaBerkasCitra(char *NamaArsip)`
   Include: `bitmap.h`
   Penjelasan: Fungsi untuk membaca citra dari arsip.

```
void BacaBerkasCitra(char *NamaArsip)
/* Membaca citra bitmap dari arsip */
{
   FILE *fp;  /* berkas masukan  */
   int  i, j;

   fp=fopen(NamaArsip,"rb");

   /* Baca header berkas citra masukan */
```

```
   BacaHeader(fp);

   /* Alokasi memori untuk citra */
   if (jumlahbit == 8)    /* citra 8 bit */
     AlokasiMemoriCitra(lebar, tinggi);
   else
     if (jumlahbit == 24)  /* citra 24 bit */
        AlokasiMemoriCitra24Bit(lebar, tinggi);
     else
        {
         printf("Tidak menangani citra selain 8-bit dan 24-bit");
         exit(0);
        }

   /* baca seluruh pixel citra */
   BacaDataBitmap(fp, lebar, tinggi);
   fclose(fp);

   /* Tampilkan citra yang sudah dibaca dari arsip */
   TampilkanCitra(lebar, tinggi);

}
```

Algoritma 3.9. *Membaca citra dari arsip.*

3. Membaca *header* berkas *bitmap*.
   Nama fungsi: `BacaHeader(FILE *fp)`
   Include: `bitmap.h`
   Penjelasan: Fungsi untuk membaca *header* berkas *bitmap.*

```
void BacaHeader(FILE *fp)
/* Membaca header berkas citra BMP. fp adalah berkas citra yang
dimampatkan. */
{
  /* Baca header arsip */
  fread(&TipeBMP1,sizeof(unsigned char),1,fp);
  fread(&TipeBMP2,sizeof(unsigned char),1,fp);
  fread(&UkuranBMP,sizeof(unsigned long),1,fp);
  fread(&XhotSpot,sizeof(unsigned short int),1,fp);
  fread(&YhotSpot,sizeof(unsigned short int),1,fp);
  fread(&OffBits,sizeof(unsigned int),1,fp);

  if (TipeBMP1 == 'B' && TipeBMP2 == 'M')
  {
    fread(&UkuranHeader,sizeof(unsigned long),1,fp);  /*baca ukuran header
                                                        BMP */
    if (UkuranHeader==12)     /* berkas BMP versi lama */
    {
      /* baca header BMP versi lama */
      fread(&LebarBMPvLama,sizeof(unsigned short int),1,fp);
      fread(&TinggiBMPvLama,sizeof(unsigned short int),1,fp);
      fread(&BidangBMPvLama,sizeof(unsigned short int),1,fp);
      fread(&JumlahBitBMPvLama,sizeof(unsigned short int),1,fp);

      tinggi = (int) TinggiBMPvLama;        /* tinggi citra */
      lebar  = (int) LebarBMPvLama;         /* lebar citra */
```

```
    jumlahbit = (int) JumlahBitBMPvLama;  /* kedalaman warna */
  }
  else
   {
    if (UkuranHeader==40)    /* berkas BMP versi Windows */
    {
      /* baca hader BMP versi Windows */
      fread(&LebarBMPvBaru,sizeof(unsigned int),1,fp);
      fread(&TinggiBMPvBaru,sizeof(unsigned int),1,fp);
      fread(&BidangBMPvBaru,sizeof(unsigned short int),1,fp);
      fread(&JumlahBitBMPvBaru,sizeof(unsigned short int),1,fp);
      fread(&MampatBMPvBaru,sizeof(unsigned int),1,fp);
      fread(&UkuranCitraBMPvBaru,sizeof(unsigned int),1,fp);
      fread(&ResolusiHorizontalBMPvBaru,sizeof(unsigned int),1,fp);
      fread(&ResolusiVertikalBMPvBaru,sizeof(unsigned int),1,fp);
      fread(&WarnaTerpakaiBMPvBaru,sizeof(unsigned int),1,fp);
      fread(&WarnaPentingBMPvBaru,sizeof(unsigned int),1,fp);

      tinggi = (int) TinggiBMPvBaru;        /* tinggi citra */
      lebar = (int) LebarBMPvBaru;          /* lebar citra */
      jumlahbit = (int) JumlahBitBMPvBaru;  /* kedalaman pixel */
    }
    else    /* UkuranHeader = 64,  berkas BMP versi OS2 */
      {
       /* baca header BMP versi OS2 */
       fread(&JumlahBitBMPvOS2,sizeof(unsigned short int),1,fp);
       fread(&MampatBMPvOS2,sizeof(unsigned int),1,fp);
       fread(&UkuranCitraBMPvOS2,sizeof(unsigned int),1,fp);
       fread(&ResolusiHorizontalBMPvOS2,sizeof(unsigned int),1,fp);
       fread(&ResolusiVertikalBMPvOS2,sizeof(unsigned int),1,fp);
       fread(&WarnaTerpakaiBMPvOS2,sizeof(unsigned int),1,fp);
       fread(&WarnaPentingBMPvOS2,sizeof(unsigned int),1,fp);
       fread(&UnitBMPvOS2,sizeof(unsigned short int),1,fp);
       fread(&CadanganBMPvOS2,sizeof(unsigned short int),1,fp);
       fread(&PerekamanBMPvOS2,sizeof(unsigned short int),1,fp);
       fread(&RenderingBMPvOS2,sizeof(unsigned short int),1,fp);
       fread(&Ukuran1BMPvOS2,sizeof(unsigned int),1,fp);
       fread(&Ukuran2BMPvOS2,sizeof(unsigned int),1,fp);
       fread(&PengkodeanWarnaBMPvOS2,sizeof(unsigned int),1,fp);
       fread(&PenciriBMPvOS2,sizeof(unsigned int),1,fp);

       tinggi = (int) TinggiBMPvOS2;       /* tinggi citra */
       lebar  = (int) LebarBMPvOS2;        /* lebar citra */
       jumlahbit = (int) JumlahBitBMPvOS2; /* kedalaman warna */
      }
  }

  /* baca palet */
  if (UkuranHeader==12)  /* citra BMP versi lama */
  {
    jumlahpalet = 1 << JumlahBitBMPvLama;      /* 2^JumlahBitBMPvLama */
    paletVlama = (RGBvLama*) malloc (jumlahpalet*sizeof(RGBvLama));
    fread(paletVlama,sizeof(RGBvLama),jumlahpalet,fp);
  }
  else  /* citra BMP versi baru */
    if (UkuranHeader==40)
    {
      if (JumlahBitBMPvBaru != 24)
      {
       if (WarnaTerpakaiBMPvBaru == 0)
          jumlahpalet = 1 << JumlahBitBMPvBaru;
       else
```

```
            jumlahpalet = WarnaTerpakaiBMPvBaru;

         paletVbaru = (RGBvBaru*) malloc (jumlahpalet*sizeof(RGBvBaru));
         fread(paletVbaru,sizeof(RGBvBaru),jumlahpalet,fp);
        }
      }
      else
         if (UkuranHeader==64)
         {
          if (JumlahBitBMPvOS2 != 24)
          {
            if (WarnaTerpakaiBMPvOS2 == 0)
              jumlahpalet = 1 << JumlahBitBMPvOS2;
            else
              jumlahpalet = WarnaTerpakaiBMPvOS2;

            paletVbaru = (RGBvBaru*)malloc(jumlahpalet*sizeof(RGBvBaru));
            fread(paletVbaru,sizeof(RGBvBaru),jumlahpalet,fp);
          }
        }
  }
  else
   {
    printf("Bukan berkas citra bitmap");
    exit(0);
   }
}
```

Algoritma 3.10. *Pembacaan header berkas bitmap.*

4. Membaca data *bitmap*.
   Nama fungsi: `BacaDataBitmap(FILE *fp, int N, int M)`
   Include: *bitmap.h*
   Penjelasan: Fungsi untuk membaca data *bitmap.*

```
void BacaDataBitmap(FILE *fp, int N, int M)
/* Membaca data pixel dari berkas masukan fp untuk citra yang berukuran N
x M */
{
  int i, j, k;

  switch (jumlahbit)
  {
    case 8:
    {
      for (i=N-1; i>=0; i--)
         /* Baca pixel per baris */
        if (fread(Image[i],sizeof(TIPE_CITRA),M,fp)!= (unsigned int)M)
        {
          printf("Data bitmap tidak lengkap");
          exit(0);
        }
      break;
    }
    case 24:
    {
```

```
      for (i=N-1; i>=0; i--)
      /* Baca pixel per baris */
      if (fread(Image24bit[i],sizeof(TIPE_CITRA24BIT),M,fp)!= (unsigned
         int)M)
      {
         printf("Data bitmap tidak lengkap");
         exit(0);
      }
      break;
    }
    default:
    {
      printf("Bukan format citra 8 atau 24 bit");
      exit(0);
    }
  }
}
```

Algoritma 3.11. *Pembacaan header berkas bitmap.*

5. Menampilkan citra ke layar.
   Nama fungsi: `TampilkanCitra(int N, int M)`
   Include: *bitmap.h*
   Penjelasan: Fungsi untuk menampilkan citra ke layar.

```
void TampilkanCitra(int N, int M)
/* Menampilkan citra yang berukura N x M ke layar */
{
   COLORREF TabelWarna[256];       /* Tabel warna (palet) citra BMP */
   HDC      MemDC;                 /* Handle ke memory device context */
   HBITMAP  mbitmap;               /* Handle ke citra */
   HWND     hwnd;                  /* Handle ke window */

   int  palet,  /* indeks palet pixel */
        i, j;

   hwnd = GetActiveWindow();
   MemDC = CreateCompatibleDC(GetDC(hwnd));
   mbitmap = CreateCompatibleBitmap(GetDC(hwnd), M, N);
   SelectObject(MemDC, mbitmap);

   switch (jumlahbit)
   {
     case 8:
     {
       /* Definisikan palet */
       for (i=0; i<jumlahpalet; i++)
       {
         if (UkuranHeader==40 ||UkuranHeader==64)  /* bitmap versi baru */
             TabelWarna[i]=GetNearestColor(MemDC, RGB(paletVbaru[i].B,
                                           paletVbaru[i].G,
                                           paletVbaru[i].R));
         else /*UkuranHeader=12, bitmap versi lama */
            TabelWarna[i]=GetNearestColor(MemDC, RGB(paletVlama[i].B,
                                           paletVlama[i].G,
                                           paletVlama[i].R));
```

```
        }
        /* Isikan pixel ke memori device (layar) */
        for (i=0; i<N; i++)
          for (j=0; j<M; j++)
           { palet = Image[i][j];
             SetPixelV(MemDC, j, i, TabelWarna[palet]);
           }

       /* Tembakkan citra ke layar */
       BitBlt(GetDC(hwnd), 0, 0, M, N, MemDC, 0, 0, SRCCOPY);
       break;
      }
      case 24:
      {
        for (i=0; i<N; i++)
          for (j=0; j<M; j++)
           { palet = GetNearestColor(MemDC, RGB(Image24bit[i][j].B,
                                     Image24bit[i][j].G,
                                     Image24bit[i][j].R));
             SetPixelV(MemDC, j, i, palet);
           }

       /* Tembakkan citra ke layar */
       BitBlt(GetDC(hwnd), 0, 0, M, N, MemDC, 0, 0, SRCCOPY);
      }
    }
}
```

Algoritma 3.12. *Menampilkan citra ke layar.*

6. Alokasi/dealoaksi memori matriks (citra).
   Nama fungsi: a. `AlokasiMemoriCitra(int N, int M)`
   b. `AlokasiMemoriCitra24Bit(int N, int M)`
   c. `**alokasi(int N, int M, int UkuranElemen)`
   d. `*xalloc(unsigned ukuran)`
   e. `DealokasiMemoriCitra(int N, int jumlahbit)`
   Include: `bitmap.h`
   Penjelasan: Fungsi-fungsi untuk alokasi/dealokasi memori matriks.

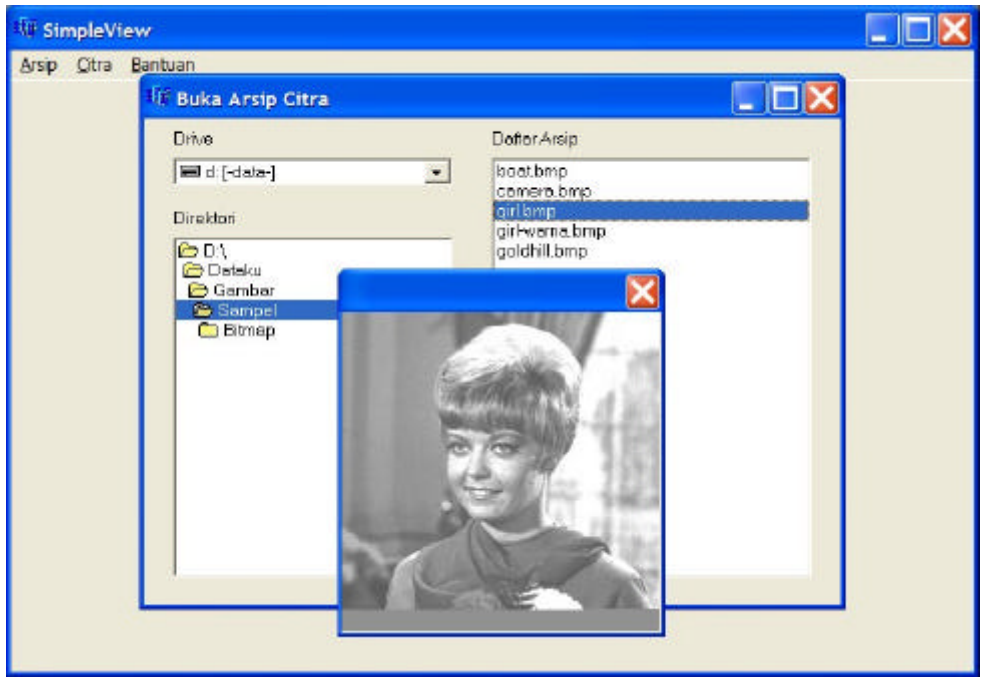
```
void AlokasiMemoriCitra(int N, int M, int jumlahbit)
/* Mengalokasikan memori untuk citra 8-bit yang berukuran N x M */
{
  Image = (TIPE_CITRA**) alokasi (N, M, sizeof(TIPE_CITRA));
}

void AlokasiMemoriCitra24Bit(int N, int M, int jumlahbit)
/* Mengalokasikan memori untuk citra 24-bit yang berukuran N x M */
{
  Image24bit = (TIPE_CITRA24BIT**) alokasi (N, M, sizeof(TIPE_CITRA24BIT));

}


void *xalloc(unsigned ukuran)
```

```
/* Mengalokasikan memori dan memeriksa apakah alokasi memori berhasil */
{
  void *p = malloc(ukuran);
  if (p==NULL)
  {
    printf("Memori tidak cukup untuk alokasi matriks");
    exit(0);
  }
  return p;
}


void **alokasi(int N, int M, int UkuranElemen)
/* Mengalokasi memori untuk matriks yang berukuran N x M. Setiap elemen
martiks
   berukuran UkuranElemen byte.

*/
{
  int i;
  void **larik = (void**)xalloc(N * sizeof(void *));

  for (i=0; i<N; i++)
    larik[i] = (void*)xalloc(M * UkuranElemen);
  return larik;
}


void DealokasiMemoriCitra(int N, int jumlahbit)
/* Mengembalikan memori yang digunakan oleh matriks */
{
  int i;

  if (jumlahbit!=24)
  {
    for (i=0; i<N; i++)
    {
     free(citra[i]);
    }
    free(citra);
  }
  else
  {
    for (i=0; i<N; i++)

    {
     free(citra24bit[i]);
    }
    free(citra24bit);
  }
}
```

Algoritma 3.13. *Alokasi/dealokasi memori matriks.*

Gambar 3.7 adalah contoh antarmuka program yang membaca dan menampilkan citra *bitmap* ke layar, dengan menggunakan primitif-primitif *bitmap* yang sudah

disebutkan di atas. Program dibuat dengan *Borland C++ Bulder 6* dan diberi nama *SimpleView*. Pengguna membuka citra bitmap dengan memilih nama berkas citra dari daftar arsip. Citra *bitmap* yang dibuka kemudian ditampilkan ke layar. Contoh citra yang ditampilkan adalah citra *girl*.



Gambar 3.7 *Program SimpleView untuk membuka dan menampilkan citra bitmap (dibuat dengan Borland C++ Builder)*