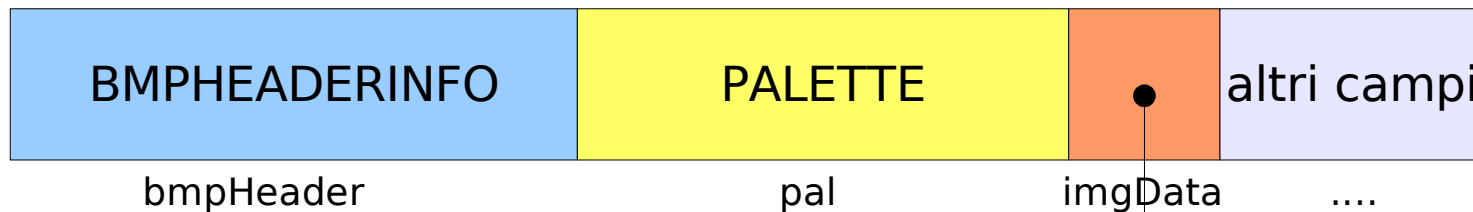


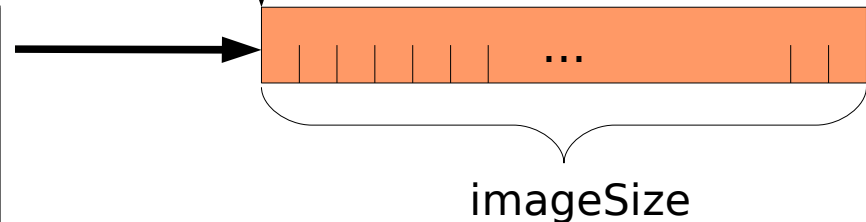
Elaborazione di immagini

- Per rappresentare i dati del file BMP in un programma possiamo raggrupparli in una struttura (FBMP)

FBMP

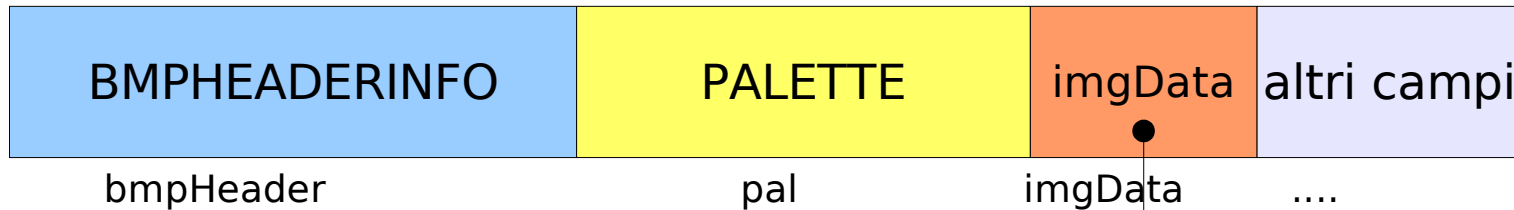


Dati RGB allocati dinamicamente in una area di memoria **da gestire** come matrice tridimensionale (per semplicità consideriamo immagini con **width % 4 == 0**)



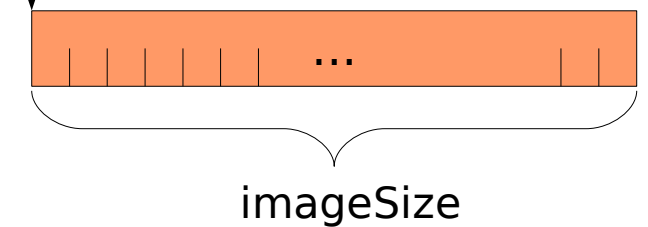
Elaborazione di immagini

FBMP



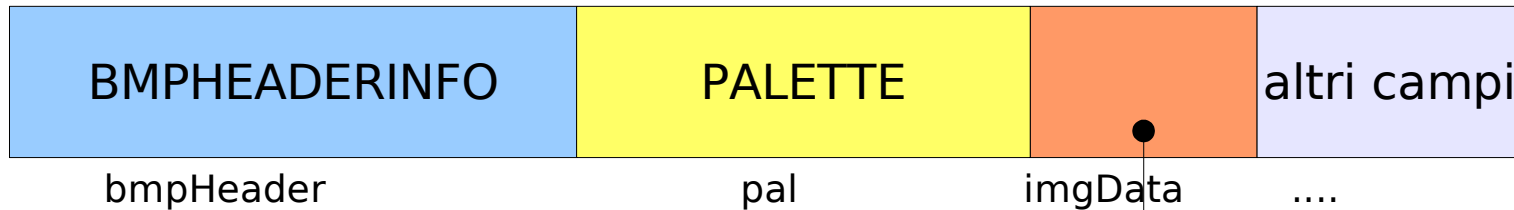
```
typedef
    unsigned char PALETTE[256][4];
```

```
typedef
    struct {
        BMPHEADERINFO bmpHeader;
        PALETTE pal;
        unsigned int palSize;
        void *imgData;
        unsigned int bpp; // bytes per pixel;
    } FBMP;
```



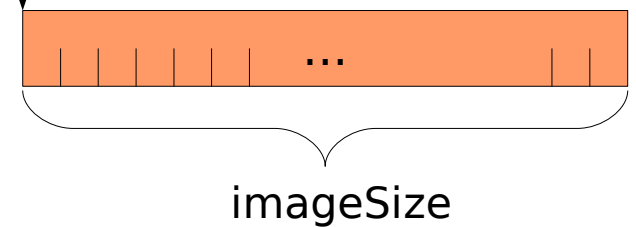
Elaborazione di immagini

FBMP



```
typedef
    unsigned char PALETTE[256][4];
```

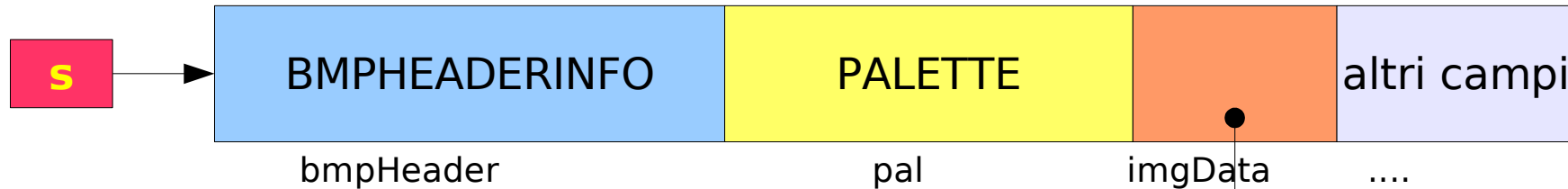
```
typedef
    struct {
        BMPHEADERINFO bmpHeader;
        PALETTE pal;
        unsigned int palSize;
        void *imgData;
        unsigned int bpp; // byt
    } FBMP;
```



Le dimensioni della matrice possono variare di volta in volta. Utilizziamo allora un puntatore generico (**void ***) ad un'area **non strutturata** che verrà poi convertito in matrice con un **type casting** implicito nel richiamo delle funzioni di elaborazione delle immagini.

Elaborazione di immagini

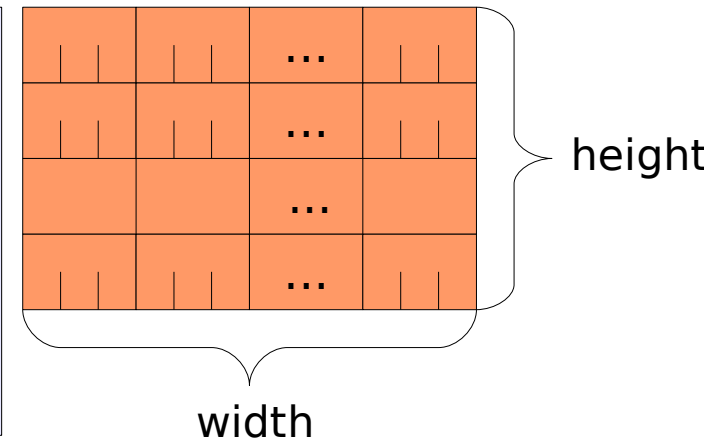
FBMP



```
...  
elab( s->bmpHeader.height,  
      s->bmpHeader.width,  
      s->bpp,  
      s->imgData );  
...
```

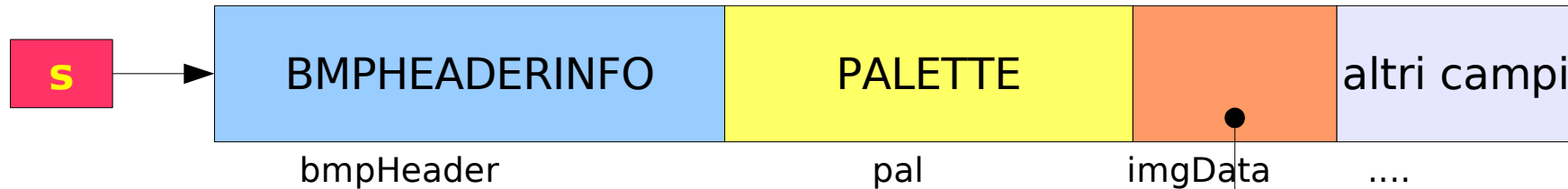
Nell'ambiente chiamante **imgData** punta a void (indirizzo generico non dereferenzabile); la zona di memoria **non ha** una struttura

```
void elab( int h, int w, int bpp,  
          unsigned char img[h][w][bpp] ){  
    for (i=0; i<h; ++i)  
        for (j=0; j<w; ++j)  
            for (k=0; k<bpp; ++k)  
                ... img[i][j][k] ...  
}
```



Elaborazione di immagini

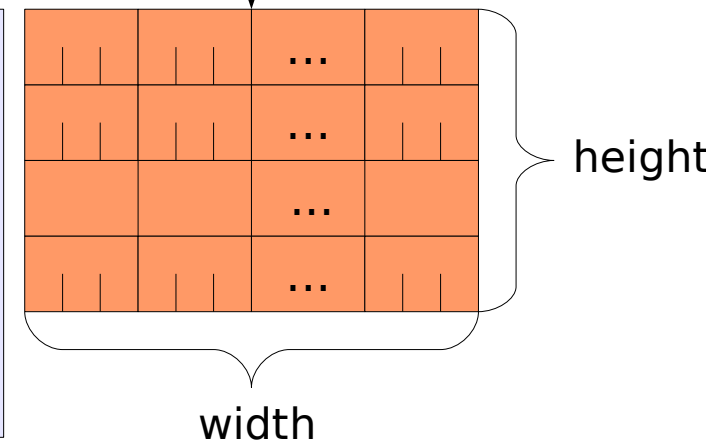
FBMP



```
...  
elab( s->bmpHeader.height,  
      s->bmpHeader.width,  
      s->bpp,  
      s->imgData );  
...
```

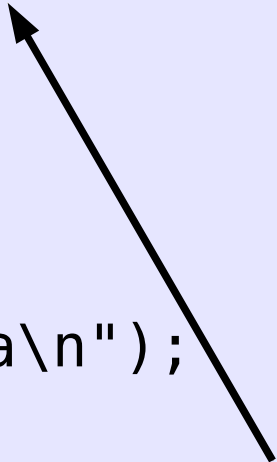
Nell'ambiente chiamato **img** è un array tridimensionale: la zona di memoria **viene vista con** una struttura

```
void elab( int h, int w, int bpp,  
          unsigned char img[h][w][bpp] ) {  
    for (i=0; i<h; ++i)  
        for (j=0; j<w; ++j)  
            for (k=0; k<bpp; ++k)  
                ... img[i][j][k] ...  
}
```



Lettura file BMP

```
void readBmpFile(char *fBmpName, FBMP *fBmpStr){  
    FILE *fBmp;  
    void *imm;  
  
    fBmp=fopen(fBmpName, "rb+");  
    if (fBmp==NULL){  
        printf("Errore in apertura\n");  
        fBmpStr->imgData=NULL;  
    }  
    else{  
        ...  
    }  
}
```



La funzione riceve il nome del file e restituisce la struttura FBMP riempita nei suoi campi. Se si verificano errori il campo dei dati RGB punterà a NULL

Lettura file BMP


...

```
fread(&fBmpStr->bmpHeader, sizeof(BMPHEADERINFO), 1, fBmp);  
imm=malloc(fBmpStr->bmpHeader.imageSize);  
fBmpStr->palSize=fBmpStr->bmpHeader.headerSize -  
                fBmpStr->bmpHeader.infoSize - 14;  
if (fBmpStr->palSize > 0)  
    fread(fBmpStr->pal, 1, fBmpStr->palSize, fBmp);  
fseek(fBmp, fBmpStr->bmpHeader.headerSize, SEEK_SET);  
fread(imm, fBmpStr->bmpHeader.imageSize, 1, fBmp);  
fBmpStr->imgData=imm;  
fclose(fBmp);  
switch (fBmpStr->bmpHeader.color)  
    case 8: fBmpStr->bpp=1; break;  
    case 24: fBmpStr->bpp=3; break;  
}  
}  
}
```

Per prima cosa si legge
l'header del file nel campo
apposito.

Lettura file BMP

```
...  
fread(&fBmpStr->bmpHeader, sizeof(BMPHEADERINFO), 1, fBmp);  
imm=malloc(fBmpStr->bmpHeader.imageSize);  
fBmpStr->palSize=fBmpStr->bmpHeader.headerSize -  
                fBmpStr->bmpHeader.infoSize - 14;  
if (fBmpStr->palSize > 0)  
    fread(fBmpStr->pal, 1, fBmpStr->palSize, fBmp);  
fseek(fBmp, fBmpStr->bmpHeader.headerSize, SEEK_SET);  
fread(imm, fBmpStr->bmpHeader.imageSize, 1, fBmp);  
fBmpStr->imgData=imm;  
fclose(fBmp);  
switch (fBmpStr->bmpHeader.colorType)  
    case 8: fBmpStr->bpp=1; break;  
    case 24: fBmpStr->bpp=3; break;  
}  
}  
}
```



Quindi si alloca lo spazio per i dati RGB. **imm** è un puntatore ausiliario locale.


Lettura file BMP

```
...  
fread(&fBmpStr->bmpHeader, sizeof(BMPHEADERINFO), 1, fBmp);  
imm=malloc(fBmpStr->bmpHeader.imageSize);  
fBmpStr->palSize=fBmpStr->bmpHeader.headerSize -  
                fBmpStr->bmpHeader.infoSize - 14;  
if (fBmpStr->palSize > 0)  
    fread(fBmpStr->pal, 1, fBmpStr->palSize, fBmp);  
fseek(fBmp, fBmpStr->bmpHeader.headerSize, SEEK_SET);  
fread(imm, fBmpStr->bmpHeader.imageSize, 1, fBmp);  
fBmpStr->imgData=imm;  
fclose(fBmp);  
switch (fBmpStr->bmpHeader.colorDepth){  
    case 1: break;  
    case 3: break;  
}  
}
```

Se esiste una palette la si legge nel campo apposito.

Lettura file BMP

```
...  
fread(&fBmpStr->bmpHeader, sizeof(BMPHEADERINFO), 1, fBmp);  
imm=malloc(fBmpStr->bmpHeader.imageSize);  
fBmpStr->palSize=fBmpStr->bmpHeader.headerSize -  
                fBmpStr->bmpHeader.infoSize - 14;  
if (fBmpStr->palSize > 0)  
    fread(fBmpStr->pal, 1, fBmpStr->palSize, fBmp);  
fseek(fBmp, fBmpStr->bmpHeader.headerSize, SEEK_SET);  
fread(imm, fBmpStr->bmpHeader.imageSize, 1, fBmp);  
fBmpStr->imgData=imm;  
fclose(fBmp);  
switch (fBmpStr->bmpHeader.colorDepth){  
    case 1; break;  
    case 3; break;  
}  
}
```



Ci si sposta direttamente all'inizio dei dati RGB e si leggono nella zona di memoria allocata dinamicamente.

Lettura file BMP

```
...  
fread(&fBmpStr->bmpHeader, sizeof(BMPHEADERINFO), 1, fBmp);  
imm=malloc(fBmpStr->bmpHeader.imageSize);  
fBmpStr->palSize=fBmpStr->bmpHeader.headerSize -  
                fBmpStr->bmpHeader.infoSize - 14;  
if (fBmpStr->palSize > 0)  
    fread(fBmpStr->pal, 1, fBmpStr->palSize, fBmp);  
fseek(fBmp, fBmpStr->bmpHeader.headerSize, SEEK_SET);  
fread(imm, fBmpStr->bmpHeader.imageSize, 1, fBmp);  
fBmpStr->imgData=imm;  
fclose(fBmp);  
switch (fBmpStr->bmpHeader.colorDepth){  
    case 8: fBmpStr->bpp=1; break;  
    case 24: fBmpStr->bpp=3; break;  
}
```

Si assegna il puntatore alla zona
allocata al campo della struttura

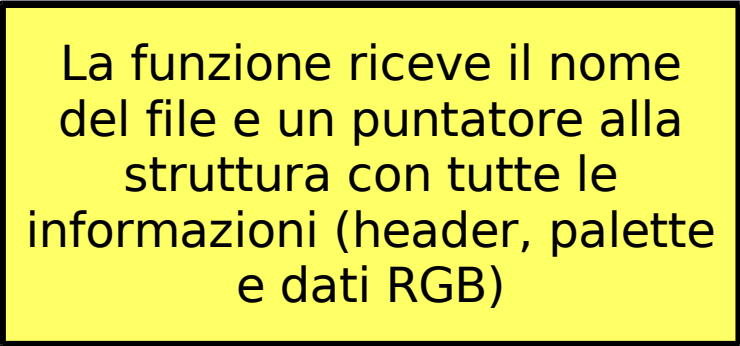
Lettura file BMP

```
...  
fread(&fBmpStr->bmpHeader, sizeof(BMPHEADERINFO), 1, fBmp);  
imm=malloc(fBmpStr->bmpHeader.imageSize);  
fBmpStr->palSize=fBmpStr->bmpHeader.headerSize -  
                fBmpStr->bmpHeader.infoSize - 14;  
if (fBmpStr->palSize > 0)  
    fread(fBmpStr->pal, 1, fBmpStr->palSize, fBmp);  
fseek(fBmp, fBmpStr->bmpHeader.offsetToPixels, SEEK_SET);  
fread(imm, fBmpStr->bmpHeader.imageSize, 1, fBmp);  
fBmpStr->imgData=imm;  
fclose(fBmp);  
switch (fBmpStr->bmpHeader.colorDepth){  
    case 8: fBmpStr->bpp=1; break;  
    case 24: fBmpStr->bpp=3; break;  
}  
}  
}
```

Si chiude quindi il file e si
determina il numero di byte per
pixel

Scrittura file BMP

```
void writeBmpFile(char *fBmpName, const FBMP *fBmpStr){  
    FILE *fBmp;  
  
    fBmp=fopen(fBmpName, "wb+");  
    if (fBmp==NULL){  
        printf("Errore in creazione\n");  
    }  
    else{  
        fwrite(&fBmpStr->bmpHeader,  
               sizeof(BMPHEADERINFO), 1, fBmp);  
        if (fBmpStr->palSize>0)  
            fwrite(fBmpStr->pal, 1, fBmpStr->palSize, fBmp);  
        fseek(fBmp, fBmpStr->bmpHeader.headerSize, SEEK_SET);  
        fwrite(fBmpStr->imgData,  
               fBmpStr->bmpHeader.imageSize, 1, fBmp);  
        fclose(fBmp);  
    }  
}
```

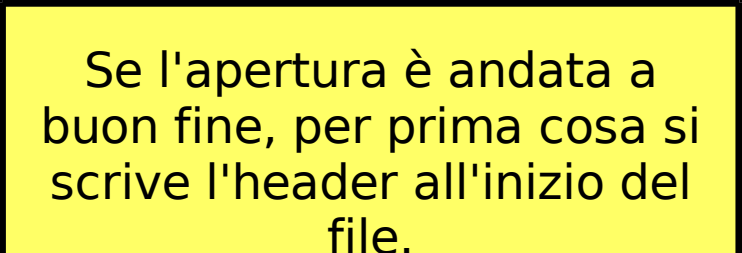


La funzione riceve il nome del file e un puntatore alla struttura con tutte le informazioni (header, palette e dati RGB)

Scrittura file BMP

```
void writeBmpFile(char *fBmpName, const FBMP *fBmpStr){
    FILE *fBmp;

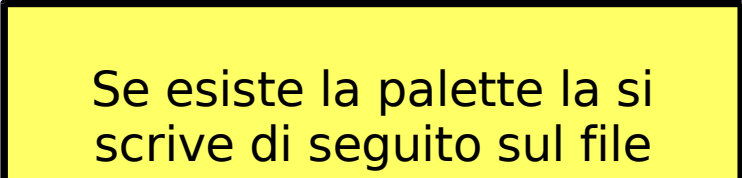
    fBmp=fopen(fBmpName, "wb+");
    if (fBmp==NULL){
        printf("Errore in creazione\n");
    }
    else{
        fwrite(&fBmpStr->bmpHeader,
            sizeof(BMPHEADERINFO), 1, fBmp);
        if (fBmpStr->palSize>0)
            fwrite(fBmpStr->pal, 1, fBmpStr->palSize, fBmp);
        fseek(fBmp, fBmpStr->bmpHeader.headerSize, SEEK_SET);
        fwrite(fBmpStr->imgData,
            fBmpStr->bmpHeader.i
        fclose(fBmp);
    }
}
```



Se l'apertura è andata a buon fine, per prima cosa si scrive l'header all'inizio del file.

Scrittura file BMP

```
void writeBmpFile(char *fBmpName, const FBMP *fBmpStr){  
    FILE *fBmp;  
  
    fBmp=fopen(fBmpName,"wb+");  
    if (fBmp==NULL){  
        printf("Errore in creazione\n");  
    }  
    else{  
        fwrite(&fBmpStr->bmpHeader,  
                sizeof(BMPHEADERINFO),1,fBmp);  
        if (fBmpStr->palSize>0)  
            fwrite(fBmpStr->pal,1,fBmpStr->palSize,fBmp);  
        fseek(fBmp,fBmpStr->bmpHeader.headerSize,SEEK_SET);  
        fwrite(fBmpStr->imgData,  
                fBmpStr->bmpHeader.imageSize,1,fBmp);  
        fclose(fBmp);  
    }  
}
```



Se esiste la palette la si scrive di seguito sul file

Scrittura file BMP

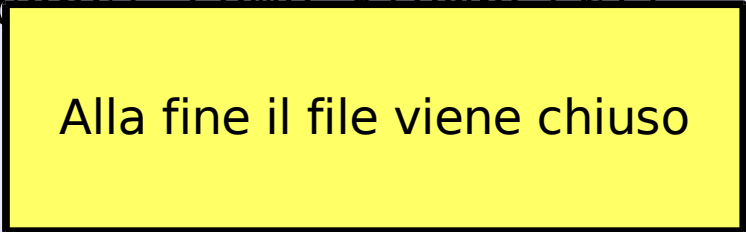
```
void writeBmpFile(char *fBmpName, const FBMP *fBmpStr) {
    FILE *fBmp;

    fBmp=fopen(fBmpName, "wb+");
    if (fBmp==NULL){
        printf("Errore in creazione\n");
    }
    else{
        fwrite(&fBmpStr->bmpHeader,
               sizeof(BMPHEADERINFO), 1, fBmp);
        if (fBmpStr->palSize>0)
            fwrite(fBmpStr->pal, 1, fBmpStr->palSize, fBmp);
        fseek(fBmp, fBmpStr->bmpHeader.headerSize, SEEK_SET);
        fwrite(fBmpStr->imgData,
              fBmpStr->bmpHeader.imageSize, 1, fBmp);
        fclose(fBmp);
    }
}
```

Ci si posiziona dopo l'header e la palette e si scrivono i dati RGB. (La seek potrebbe non servire)

Scrittura file BMP

```
void writeBmpFile(char *fBmpName, const FBMP *fBmpStr) {  
    FILE *fBmp;  
  
    fBmp=fopen(fBmpName, "wb+");  
    if (fBmp==NULL){  
        printf("Errore in creazione\n");  
    }  
    else{  
        fwrite(&fBmpStr->bmpHeader,  
               sizeof(BMPHEADERINFO), 1, fBmp);  
        if (fBmpStr->palSize>0)  
            fwrite(fBmpStr->pal, 1, fBmpStr->palSize, fBmp);  
        fseek(fBmp, fBmpStr->bmpHeader.headerSize, SEEK_SET);  
        fwrite(fBmpStr->imgData,  
               fBmpStr->bmpHeader.imageSize, 1, fBmp);  
        fclose(fBmp);  
    }  
}
```



Alla fine il file viene chiuso

Stampa header file BMP

```
void printBMPHEADERINFO(const BMPHEADERINFO *bmpHeader){  
    printf("%10s: %d\n", "fileSize", bmpHeader->fileSize);  
    printf("%10s: %d\n", "reserved", bmpHeader->reserved[0]);  
    printf("%10s: %d\n", "reserved", bmpHeader->reserved[1]);  
    printf("%10s: %d\n", "headerSize", bmpHeader->headerSize);  
    printf("%10s: %d\n", "HeaderSize", bmpHeader->headerSize);  
    printf("%10s: %d\n", "width", bmpHeader->width);  
    printf("%10s: %d\n", "height", bmpHeader->height);  
    printf("%10s: %d\n", "biPlanes", bmpHeader->biPlanes);  
    printf("%10s: %d\n", "colorDepth", bmpHeader->colorDepth);  
    printf("%10s: %d\n", "compression", bmpHeader->compression);  
    printf("%10s: %d\n", "imageSize", bmpHeader->imageSize);  
    printf("%10s: %d\n", "horRes", bmpHeader->horRes);  
    printf("%10s: %d\n", "verRes", bmpHeader->verRes);  
    printf("%10s: %d\n", "colorsNumber",  
           bmpHeader->colorsNumber);  
    printf("%10s: %d\n", "importantColor",  
           bmpHeader->importantColor);  
}
```

Come per tutte le strutture, va fatta campo per campo

Elaborazione

```
void doOp(FBMP *fBmpStr, unsigned int op){
    void *outData, *tmp;

    if (fBmpStr->imgData == NULL)
        return;
    outData=malloc(fBmpStr->bmpHeader.imageSize);
    switch(op){
        case 2: flipHorizontal(fBmpStr->bmpHeader.height,
                               fBmpStr->bmpHeader.width,
                               fBmpStr->bpp,
                               fBmpStr->imgData,outData);
                ;break;
        case 3: flipVertical(fBmpStr->bmpHeader.height,
                              fBmpStr->bmpHeader.width,
                              fBmpStr->bpp,
                              fBmpStr->imgData,outData);
                ;break;
        case 4: negative(fBmpStr->bmpHeader.height,
                          fBmpStr->bmpHeader.width,
                          fBmpStr->bpp,
                          fBmpStr->imgData,outData);
                ;break;
        case 5: color2gray(fBmpStr->bmpHeader.height,
                            fBmpStr->bmpHeader.width,
                            fBmpStr->bpp,
                            fBmpStr->imgData,outData);
                ;break;
        default:;
    }
    tmp=fBmpStr->imgData;
    fBmpStr->imgData=outData;
    free(tmp);
    scrivi(fBmpStr);
}
```

La funzione riceve il puntatore alla struttura dei dati e un intero che rappresenta l'operazione da applicare (proveniente dal main)

Elaborazione

```
void doOp(FBMP *fBmpStr, unsigned int op){
    void *outData, *tmp;

    if (fBmpStr->imgData == NULL)
        return;
    outData=malloc(fBmpStr->bmpHeader.imageSize);
    switch(op){
        case 2: flipHorizontal(fBmpStr->bmpHeader.height,
                               fBmpStr->bmpHeader.width,
                               fBmpStr->bpp,
                               fBmpStr->imgData,outData);
                ;break;
        case 3: flipVertical(fBmpStr->bmpHeader.height,
                              fBmpStr->bmpHeader.width,
                              fBmpStr->bpp,
                              fBmpStr->imgData,outData);
                ;break;
        case 4: negative(fBmpStr->bmpHeader.height,
                          fBmpStr->bmpHeader.width,
                          fBmpStr->bpp,
                          fBmpStr->imgData,outData);
                ;break;
        case 5: color2gray(fBmpStr->bmpHeader.height,
                            fBmpStr->bmpHeader.width,
                            fBmpStr->bpp,
                            fBmpStr->imgData,outData);
                ;break;
        default:;
    }
    tmp=fBmpStr->imgData;
    fBmpStr->imgData=outData;
    free(tmp);
    scrivi(fBmpStr);
}
```

Se i dati RGB non esistono (es. lettura del file non corretta) esci immediatamente dalla funzione

Elaborazione

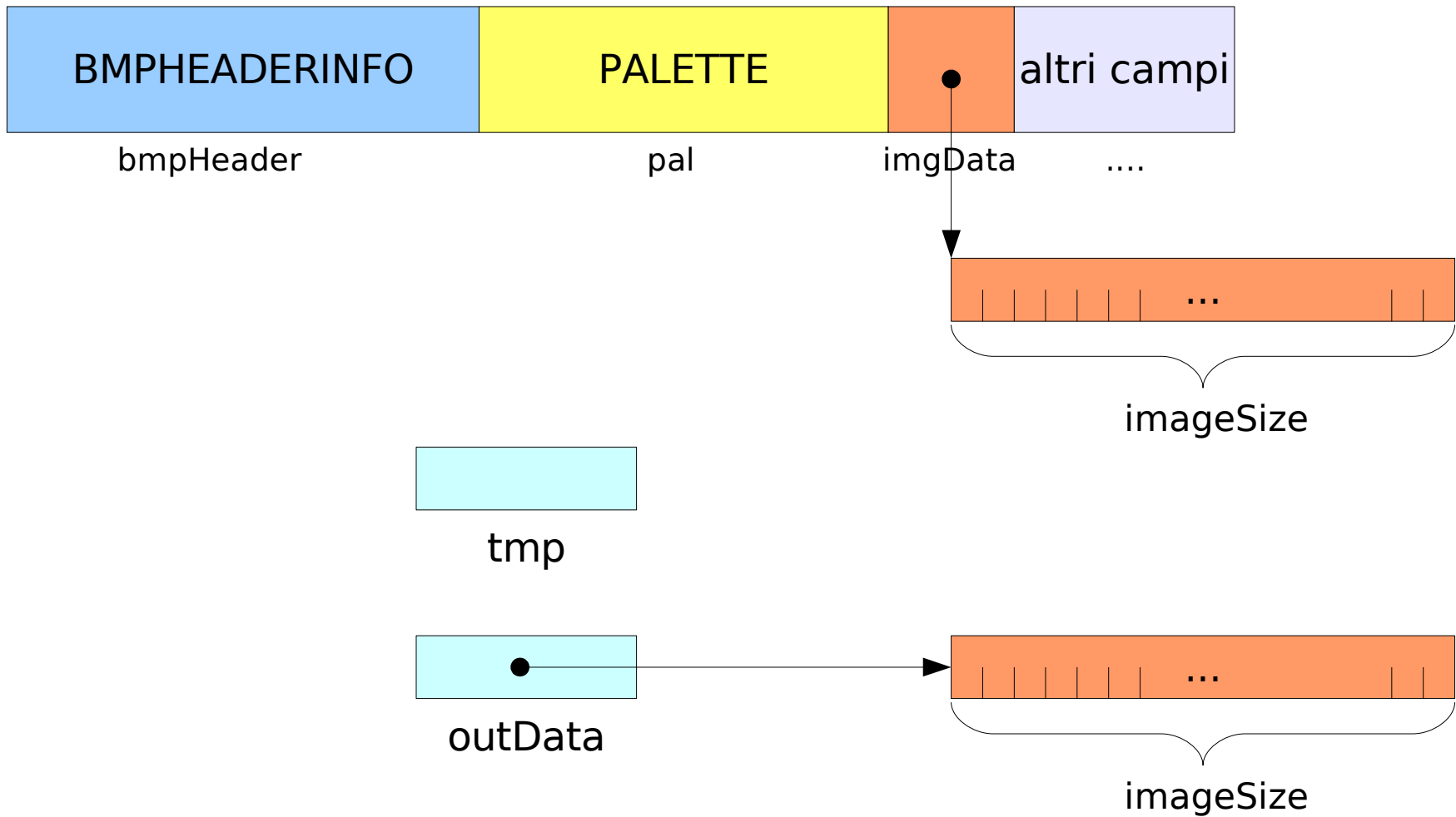
```
void doOp(FBMP *fBmpStr, unsigned int op){
    void *outData, *tmp;

    if (fBmpStr->imgData == NULL)
        return;
    outData=malloc(fBmpStr->bmpHeader.imageSize);
    switch(op){
        case 2: flipHorizontal(fBmpStr->bmpHeader.height,
                               fBmpStr->bmpHeader.width,
                               fBmpStr->bpp,
                               fBmpStr->imgData,outData);
                ;break;
        case 3: flipVertical(fBmpStr->bmpHeader.height,
                              fBmpStr->bmpHeader.width,
                              fBmpStr->bpp,
                              fBmpStr->imgData,outData);
                ;break;
        case 4: negative(fBmpStr->bmpHeader.height,
                          fBmpStr->bmpHeader.width,
                          fBmpStr->bpp,
                          fBmpStr->imgData,outData);
                ;break;
        case 5: color2gray(fBmpStr->bmpHeader.height,
                            fBmpStr->bmpHeader.width,
                            fBmpStr->bpp,
                            fBmpStr->imgData,outData);
                ;break;
        default:;
    }
    tmp=fBmpStr->imgData;
    fBmpStr->imgData=outData;
    free(tmp);
    scrivi(fBmpStr);
}
```

Alloca un'altra zona di dimensione pari ai dati RGB destinata a contenere il risultato dell'elaborazione. **outData** è un puntatore locale

Elaborazione

FBMP



Elaborazione

```
void doOp(FBMP *fBmpStr, unsigned int op){
    void *outData, *tmp;

    if (fBmpStr->imgData == NULL)
        return;
    outData=malloc(fBmpStr->bmpHeader.imageSize);
    switch(op){
        case 2: flipHorizontal(fBmpStr->bmpHeader.height,
                               fBmpStr->bmpHeader.width,
                               fBmpStr->bpp,
                               fBmpStr->imgData,outData);

                               ;break;
        case 3: flipVertical(fBmpStr->bmpHeader.height,
                              fBmpStr->bmpHeader.width,
                              fBmpStr->bpp,
                              fBmpStr->imgData,outData);

                              ;break;
        case 4: negative(fBmpStr->bmpHeader.height,
                          fBmpStr->bmpHeader.width,
                          fBmpStr->bpp,
                          fBmpStr->imgData,outData);

                          ;break;
        case 5: color2gray(fBmpStr->bmpHeader.height,
                            fBmpStr->bmpHeader.width,
                            fBmpStr->bpp,
                            fBmpStr->imgData,outData);

                            ;break;
        default:;
    }
    tmp=fBmpStr->imgData;
    fBmpStr->imgData=outData;
    free(tmp);
    scrivi(fBmpStr);
}
```

La struttura switch attiva la procedura indicata da **op**. Ciascuna funzione riceve le **dimensioni** dell'immagine, la **profondità** di colore e il **puntatore** ai dati RGB di partenza e il **puntatore** ai dati RGB risultato.

Elaborazione

```
void doOp(FBMP *fBmpStr, unsigned int op){
    void *outData, *tmp;

    if (fBmpStr->imgData == NULL)
        return;
    outData=malloc(fBmpStr->bmpHeader.imageSize);
    switch(op){
        case 2: flipHorizontal(fBmpStr->bmpHeader.height,
                               fBmpStr->bmpHeader.width,
                               fBmpStr->bpp,
                               fBmpStr->imgData,outData);
                ;break;
        case 3: flipVertical(fBmpStr->bmpHeader.height,
                              fBmpStr->bmpHeader.width,
                              fBmpStr->bpp,
                              fBmpStr->imgData,outData);
                ;break;
        case 4: negative(fBmpStr->bmpHeader.height,
                          fBmpStr->bmpHeader.width,
                          fBmpStr->bpp,
                          fBmpStr->imgData,outData);
                ;break;
        case 5: color2gray(fBmpStr->bmpHeader.height,
                            fBmpStr->bmpHeader.width,
                            fBmpStr->bpp,
                            fBmpStr->imgData,outData);
                ;break;
        default:;
    }
    tmp=fBmpStr->imgData;
    fBmpStr->imgData=outData;
    free(tmp);
    scrivi(fBmpStr);
}
```

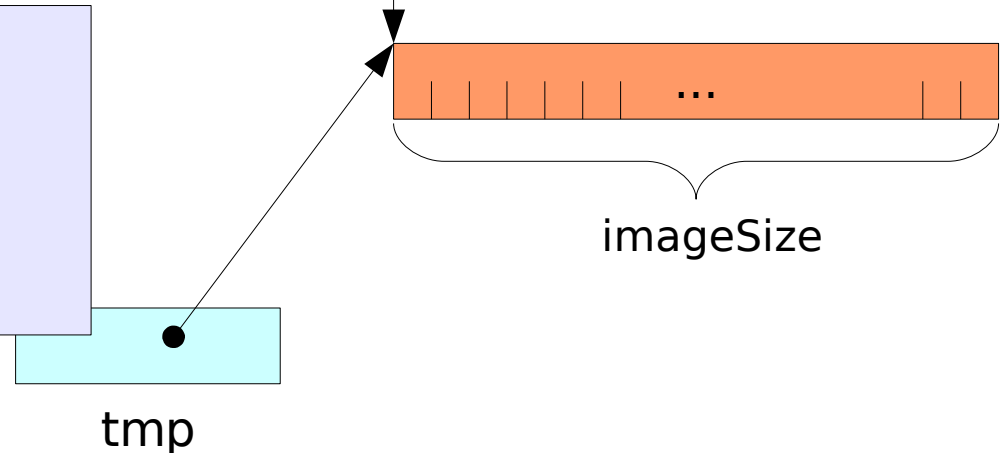
Al termine fai puntare il campo **imgData** della struttura con i dati BMP alla zona con i risultati dell'elaborazione puntati da **outData**.

Elaborazione

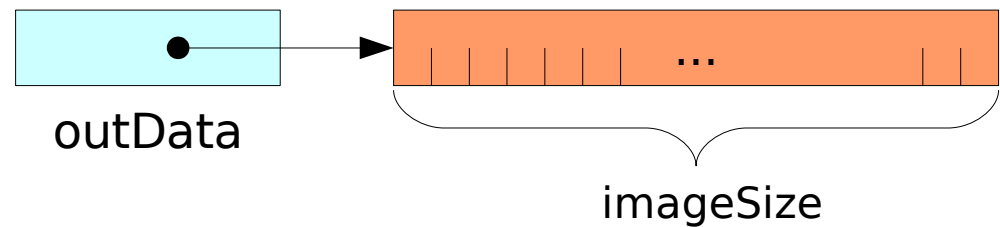
FBMP



```
...  
tmp=fBmpStr->imgData;  
fBmpStr->imgData=outData;  
free(tmp);  
...
```

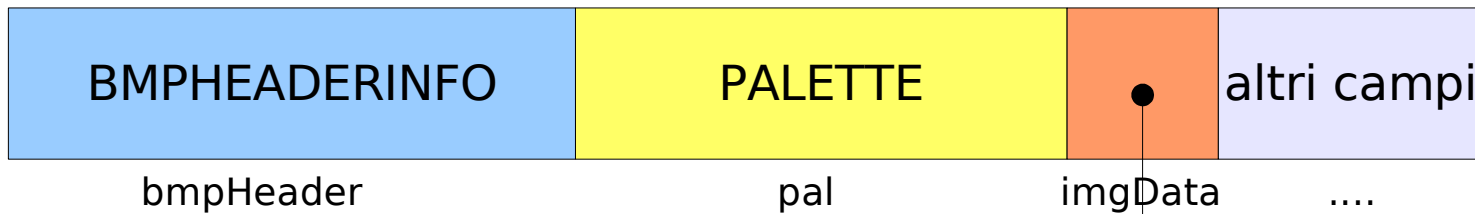


Salva il puntatore ai vecchi dati RGB in un puntatore di appoggio **tmp** (per evitare di perdere il link)

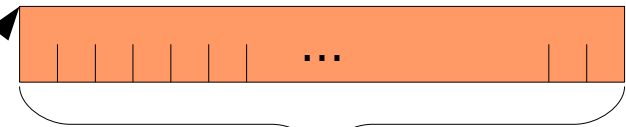


Elaborazione

FBMP



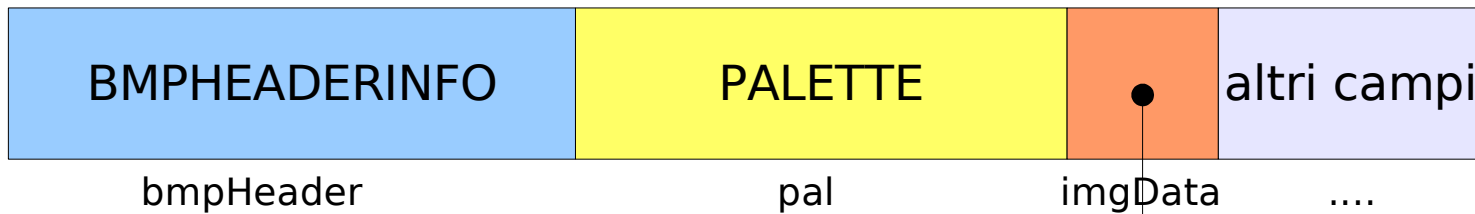
```
...  
tmp=fBmpStr->imgData;  
fBmpStr->imgData=outData;  
free(tmp);  
...
```



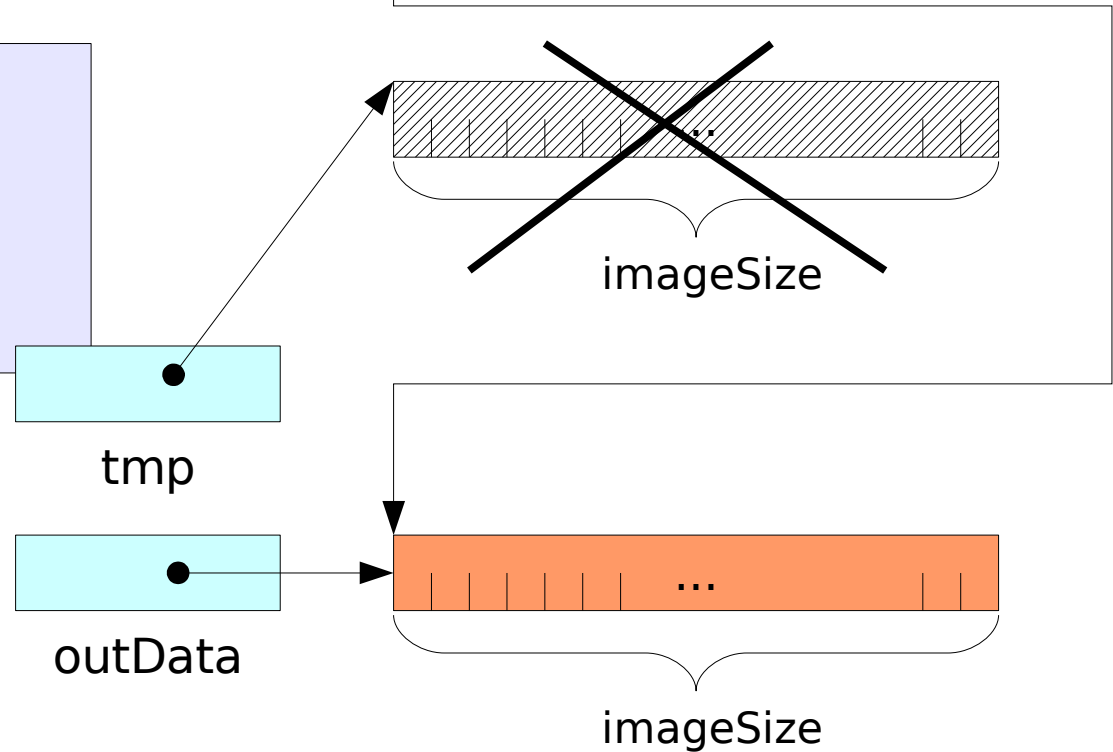
Fai puntare **imgData** ai
nuovi dati RGB

Elaborazione

FBMP



```
...  
tmp=fBmpStr->imgData;  
fBmpStr->imgData=outData;  
free(tmp);  
...
```



Ora puoi liberare i **vecchi** dati RGB (senza **tmp** non avremmo potuto farlo).

Elaborazione


```
void doOp(FBMP *fBmpStr, unsigned int op){
    void *outData, *tmp;

    if (fBmpStr->imgData == NULL)
        return;
    outData=malloc(fBmpStr->bmpHeader.imageSize);
    switch(op){
        case 2: flipHorizontal(fBmpStr->bmpHeader.height,
                               fBmpStr->bmpHeader.width,
                               fBmpStr->bpp,
                               fBmpStr->imgData,outData);
                ;break;
        case 3: flipVertical(fBmpStr->bmpHeader.height,
                              fBmpStr->bmpHeader.width,
                              fBmpStr->bpp,
                              fBmpStr->imgData,outData);
                ;break;
        case 4: negative(fBmpStr->bmpHeader.height,
                          fBmpStr->bmpHeader.width,
                          fBmpStr->bpp,
                          fBmpStr->imgData,outData);
                ;break;
        case 5: color2gray(fBmpStr->bmpHeader.height,
                            fBmpStr->bmpHeader.width,
                            fBmpStr->bpp,
                            fBmpStr->imgData,outData);
                ;break;
        default;;
    }
    tmp=fBmpStr->imgData;
    fBmpStr->imgData=outData;
    free(tmp);
    scrivi(fBmpStr);
}
```

Liberata quindi la memoria allocata, scrivi l'immagine sul file di output e visualizzalo (la funzione scrivi raccoglie queste due operazioni)

Funzione di elaborazione


```
void negative( int h, int w, int bpp,  
              unsigned char imm[h][w][bpp],  
              unsigned char out[h][w][bpp]){  
    int i,j,k;  
    for (i=0; i<h; i++)  
        for (j=0; j<w; j++)  
            for (k=0; k<bpp; k++)  
                out[i][j][k]=~imm[i][j][k];  
}
```



La funzione di elaborazione riceve le dimensioni dell'immagine e i **puntatori a due zone di memoria allocata dinamicamente, che vengono viste dalla funzione come array a 3 dimensioni a dimensione variabile** (passate come parametro).

Funzione di elaborazione

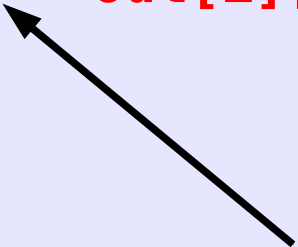
```
void negative( int h, int w, int bpp,  
              unsigned char imm[h][w][bpp],  
              unsigned char out[h][w][bpp]){  
    int i,j,k;  
    for (i=0; i<h; i++)  
        for (j=0; j<w; j++)  
            for (k=0; k<bpp; k++)  
                out[i][j][k]=~imm[i][j][k];  
}
```



In questo modo le zone di memoria **non strutturata esternamente alla funzione**, puntate da **imgData** e **outData** (che sono puntatori a **void**) **acquistano una struttura internamente alla funzione**.

Funzione di elaborazione

```
void negative( int h, int w, int bpp,  
              unsigned char imm[h][w][bpp],  
              unsigned char out[h][w][bpp]){  
    int i,j,k;  
  
    for (i=0; i<h; i++)  
        for (j=0; j<w; j++)  
            for (k=0; k<bpp; k++)  
                out[i][j][k]=~imm[i][j][k];  
}
```



L'elaborazione diventa quindi un semplice algoritmo sulle matrici; scansione della matrice e inversione (in questo caso) dei colori.

Funzione di elaborazione

```
void color2gray( int h, int w, int bpp,  
                unsigned char imm[h][w][bpp],  
                unsigned char out[h][w][bpp]){  
  
    int i,j,k;  
  
    for (i=0; i<h; i++)  
        for (j=0; j<w; j++){  
            unsigned long c=0;  
            for (k=0; k<bpp; k++)  
                c+=imm[i][j][k];  
            c=c/bpp;  
            for (k=0; k<bpp; k++)  
                out[i][j][k]=(unsigned char)c;  
        }  
  
}
```


Funzione di elaborazione

```
void flipHorizontal( int h, int w, int bpp,  
                    unsigned char imm[h][w][bpp],  
                    unsigned char out[h][w][bpp]){  
    int i,j,j1,k;  
    for (i=0; i<h; i++)  
        for (j=0,j1=w-1; j<w; j++,j1--)  
            for (k=0; k<bpp; k++)  
                out[i][j][k]=imm[i][j1][k];  
}
```

Funzione di elaborazione

```
void flipVertical(    int h, int w, int bpp,
                    unsigned char imm[h][w][bpp],
                    unsigned char out[h][w][bpp]){

    int i,j,k;

    for (i=0,j=h-1; i<h; i++,j--)
        copyArr(w,bpp,out[i],imm[j]);
}

void copyArr(    int n, int bpp,
                unsigned char dest[n][bpp],
                unsigned char src[n][bpp]){

    int i,j;
    for (i=0; i<n; i++)
        for (j=0; j<bpp; j++)
            dest[i][j]=src[i][j];
}
```