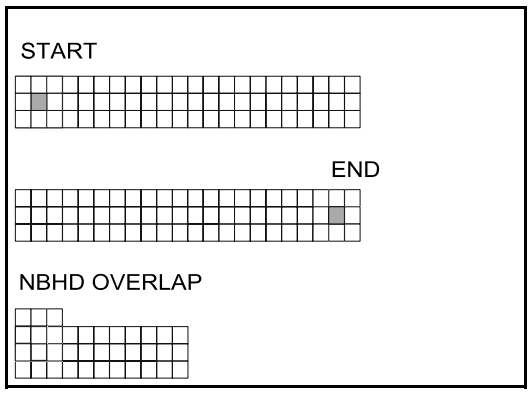
Appendix 3.A. An Algorithm for Mask Convolution

The Algorithm. A high level algorithm is described here. The mask of weights that contains p rows and q columns will be represented by {h(i,j): $0 \le i \le p-1$, $0 \le j \le q-1$}. This algorithm will process a pxq neighborhood of each pixel in the image {f(m,n): $0 \le m \le M-1$, $0 \le n \le N-1$}. It merely copies without processing those pixels whose pxq neighborhood overlaps the image, that is, is not completely in the image.

Figure 3.A.1. Processing a row



The first pixel in the upper lefthand corner to be processed will be located at the point (p/2,q/2). Recall that p and q are odd integers, so p/2 and q/2 are truncated to be the central pixel position (m,n) = (p/2,q/2) in the pxq neighborhood. The algorithm works on a strip of p rows, of which p/2 is the row to be processed (see Figure 3.A.1). Considering pxq = 3x3, the point (p/2,q/2) = (1,1) is the first pixel to be processed on the strip of rows 0, 1 and 2 with columns 0, 1 and 2 being the first 3x3 neighborhood. The last pixel in this row is at (p/2,N-1) but it can not be processed because it does not have a complete neighborhood in the image. Thus the last pixel in row p/2 to be processed is (p/2,N-1-q/2).

On a first reading of the algorithm given below, let p = 3 and q = 3. Figure 3.A.1 presents a view of a strip of rows being processed. The mask overlaps the image at the first image pixel (top left) and so we do not process the top row of pixels. Similarly, we do not process the first column nor the last, nor the last row.

The *greatest integer* for pxq = 3x3, is ⌊p/2⌋ = ⌊q/2⌋ = 1. In the first strip shown, we start the mask over a neighborhood (nbhd) in the image, and process the neighborhood to obtain a new center pixel. Then the mask moves to the right and processes the next neighborhood, and so forth, until it reaches the last row pixel for which the mask fits inside the image strip. We write the first pixel and last pixel in the strip directly to the output array without processing. Similarly, we write the first row and last row of the image to the output array. Appendix 3.B lists a C program for this. Here we present a high level process.

Algorithm 3.A.1 (ApplyMask()):

```
call GetMask();                          //Get convolution mask from user
call GetStrip();                         //Procedure gets first p rows, 0,...,p-1, from
for m = 0 to (p/2)-1 do                  //input file and places them in array f[m][n]
  for n = 0 to N-1 do                    //Read and write first p/2 rows to output array
      g[m][n] ← f[m][n];                 //g[m][n] for writing directly to output file
RowCount ← p/2;                          //First row to be processed
do                                       //Loop over all strips of p rows
  ColCount ← 0;                          //First q/2 columns can not be processed
  for n = 0 to (q/2)-1 do                //but are copied directly to output array
      g[RowCount][n] ← f[RowCount][n];   //Write unprocessed pixels in row to output array
  do                                     //Loop over all processable columns in strip
      PixelSum ← 0;                      //Initialize convolution sum to zero
      for mrow = 0 to p-1 do             //For all pixels (m,n) in current
        for ncol = Colcount to Colcount+q-1 do //neighborhood and all mask entries
            deltacol ← ncol - ColCount;  //(mask columns always go from 0 to p-1)
```

```
                PixelSum ← PixelSum + mask[mrow][deltacol]*f[mrow][ncol];   //do convolution
                PixelSum ← MaskFactor*PixelSum;       //Multiply convolution sum by mask factor
                NewPixel ← integer(PixelSum);         //Put real pixel value into integer variable
                if (NewPixel > 255) then NewPixel ← 255; //New pixel must be between 0 and 255
                if (NewPixel < 0)   then NewPixel ← 0 ;  //for grayscale processing
                g[RowCount][Colcount+q/2] ← NewPixel;//Put processed value into output array
                ColCount ← ColCount + 1;              //Increment column to be processed next
  while ColCount ≤ N-(q/2)-1;                         //& process until nbhd goes outside of image
  for n = N-(p/2) to N-1 do                           //Copy last (q/2) cols. of row to output array
        g[RowCount][n] ← f[RowCount][n];              //for current row being processed
  RowCount ← RowCount + 1;                            //Increment count of row for processing next
while GetStrip() != -1;                               //Get next strip of rows (move down 1 row)
                                                      //and place it them in array f[m][n]
for m = M-(p/2) to M-1 do                             //Read/write last p/2 rows directly to output
      for n = 0 to N-1 do                             //g[m][n] for writing to output file
            g[m][n] ← f[m][n];
```

On the first call the procedure **GetStrip**() reads the first p rows (indexed as 0,...,p-1) into the array variable f[m][n] as the first strip. The first p/2 rows are not processed because no complete neighborhoods exist for them and so they are written directly to the output array. The row count is set at p/2, which is the first row to be processed. The first (q/2) center pixels in that row are not processed (no complete neighborhood exists) and so are copied directly into the output array. Then the column count is set at zero and the first q columns are used as the first neighborhood. Mask convolution processes it to obtain the new output pixel g[p/2][q/2]. After that, the column count is incremented by one to move the mask to the right one pixel and the mask convolution is done again, and so forth.

When the processing of the current row (designated by RowCount) is completed (all pixels in the row have been processed except the first 0,...,(q/2)-1 and the last N-(q/2) columns), then these last (q/2) pixels are copied directly to the output array (the first q/2 pixels in this row wre previously copied). Then a new strip of p rows that consists of rows Rowcount,...,Rowcount+p-1 is placed in f[m][n] for m = 0,...,p-1 by **GetStrip**() and is processed the same way. When the procedure **GetStrip**() gets the last p rows M-p,...,M-(p/2)-1, the returned value is still 1 as in all previous cases. The next time, however, it returns -1 to indicate that there is no complete strip to process (no complete neighborhoods exist for the pixels in the remaining rows) and the process terminates. The last p/2 rows are then copied into the output array g[m][n] directly. Another procedure can be used to write the output row to the output image file during processing, or the entire array {g[m][n]) can be written to the file at the end. Appendix 3.B presents the C source code for mask convolution.

Linux/Solaris/UNIX System Calls to XV. It is convenient to make a *system call* to XV from the image processing program to display an image on the screen. A system call interacts with the operating system (of UNIX type) just as if the user were typing from the command line. The commands must be written into a buffer and given to the system call instruction. As before, *X-windows* must be running.

We prefer to first display the original image with a system call to XV and then display the processed image with another system call. We use the *poll* parameter so that we may process the image again, write the processed image to the output image file and display the processed image in the same processed image display window as before.

A separate subroutine (procedure, or function) can be written to make the system calls for display. We call this function **Display**() and call it from the main routine in the program at the end of a loop that

processes the image. Then the user can exit from the loop or select parameters to process the image again. When the repeat processing is done and written to the output file, then that image is displayed by XV. Our high level main procedure contains the functions in the order given below.

```
DisplayHeading();                    /Displays program heading/
OpenFiles();                         /Opens input and output image files/
do                                   /Start of do-while loop for repeat processing/
    GetMask();                       /Accepts mask input from user/
    ApplyMask();                     /Calls GetStrip() for convolution, writes output
    Display();                       /Calls XV to display images/
    key ← GetUserInfo();             /User inputs key to stop or process again/
while key != 'x';                    /End of do-while loop, 'x' exits loop/
CloseFiles();                        /Write data to output files, close files/
```

The C code for the **Display**() procedure uses the string print function *sprintf*() to write the command to be given to the system call into a buffer. The first buffer is for the command that is to copy the output file (*outfile*) to a new output file (*display_file*) to be used in the display. It is called *buffer1*. This is accomplished via

sprintf(buffer1, " %s %s %s ", " cp ", outfile, display_file);

This writes three string variables into buffer1. The first is the string "cp" to give the Linux/Solaris/UNIX command to copy a file. The second is the name of the processed (source) output file *outfile* and the third is the name *display_file* of the destination file to use for displaying the image.

The buffer that holds the command to call XV to display the image is *buffer2*. Thus we next use

sprintf(buffer2, " %s %s %s ", " xv -poll -geometry +10-50 ",display_file);

This writes four string variables into a buffer (*buffer2*). The first is the command *xv* that causes the operating system to run the XV program. The second is the parameter that tells XV to poll to see if the image file has changed and to update the display if it has. The third gives the location on the screen of the image, which in this case is at 10 pixels from the left and 50 from the bottom of the screen. The fourth is the name of the file to display, which here is *display_file*.

To actually execute the commands stored in *buffer1* and *buffer2* we use the *system*() function per

```
system(buffer1);                     /Execute file copy command in buffer1/
system("sleep 5");                   /Wait 5 msecs. for file copy to be completed/
system(buffer2);                     /Buffer2 comand calls XV to display image/
```

These system commands can be put in loops or branches and so forth, so the programmer can be quite creative in the use of XV to display the images. We use *bufferA* and *bufferB* to hold similar commands to display the original image, except that we do not use the "-poll" parameter (the original stays the same throughout and only the reprocessed output image changes). The C source code for **Display**() follows.

A Display Function in C for Use with XView

```
void Display()
{
 /***Part 1. Copy original image "infile" to file "Display_File1" and display if first time***/
 if (first_time == 1)
 { //-----Command to copy original image file "infile" to file "Display_File1"-----
        sprintf(bufferA, " %s %s %s " , " cp " , infile, Display_File1);
  //-----Give system command in "bufferA" to system to execute copy----------
        system(bufferA);
  //-----Wait for image file to be completely copied-----------------------------------
        system("sleep 2");
  //-----Command to display image in "Display_File1" with XV at (2,2)------------
        sprintf(bufferB, " %s %s %s " , " xv -geometry +2-2" , Display_File1,"&");
  //-----Display original image in "Display_File1" with XV--------------------------
        system(bufferB);
 }
//=============================================
 /***Part 2. Copy processed image "outfile" to file "Display_File2" and display on any call***/
 //-----Command to copy processed file "outfile" to file "Display_File2"------------
        sprintf(buffer1, " %s %s %s " , " cp ", outfile, Display_File2);
 //-----Give system command in "buffer2" to system to execute copy----------------
        system(buffer1);
 //-----Wait for image file to be completely copied--------------------------------------
        system("sleep 2");
 if (first_time == 1)
 { //-----Command to display image in "Display_File2" with XV at (200,200)------
        sprintf(buffer2, " %s %s %s ", " xv -poll -geometry +200-200 " , Display_File2,"&");
  //-----Display processed image in "Display_File2" with XV------------------------
        system(buffer2);
  //-----Turn off first_time so as to not call XV further-------------------------------
        first_time = 0;
  //-----The polling feature updates changed processed files automatically----------
 }
 printf(" Move images to desired location for viewing and comparison!\n");
 return;
} // End of Display() function
//============================================================
```

Notes.

i) *infile* and *outfile* are names of files given by the user to read the input image file and write the output image file, respectively. They are declared in *main* via

```
char    infile[40];
char    static outfile[40];
```

ii) *Display_File1* and *Display_File2* are names of the files to which the original image file *infile* and *outfile* are copied, respectively. They are declared in *main* via

```
char    Display_File1 = "Image_Before";
char    Display_File2 = "Image_After";
```

iii) the buffers are strings that are declared in *main* via

char bufferA[100], bufferB[100], buffer1[100], buffer2[100];

Obviously, a single buffer could be used to store each command in a sequential fashion before making a system call, but we have taken the pedagogical approach.

iii) *first_time* is an integer variable declared in *main* and set to 1. After an image has been processed, the function *Display()* is called to display the original and the processed images in two separate windows. Then *first_time* is turned off, that is, set to 0 so that no further processing of the original image will be displayed by a system call. However, any reprocessing of the image and call of *Display()* causes the new reprocessed image to be copied to *Display_File2*. The *poll* parameter causes the new reprocessed image in *Display_File2* to be written to the display in the window over the previous processed image.

Appendix 3.B - A Mask Convolution Program in C

```
//++++++++++++++++++++++<M D I P>++++++++++++++++++++++++++++++
//
//    >>>MDIP 3.1 -  MASK DIGITAL IMAGE PROCESSING Program<<<
//             Linux Version -- Gnu C++ Compiler
//           (also UNIX Version with C++ Compiler)
//
//       This program processes an input image with a user
//   supplied (user input) convolution mask, writes out result.
//
//-------------------------------------------------------------------------------------------------
//      It makes calls to XV (Xview) to display original and
//   processed images on screen. Reprocessed images are updated
//   on the screen.
//=================================================================
//                    Version updated: Jul. 2002
//-------------------------------------------------------------------------------------------------
//   This program processes a *.pgm image of raw data of size up
//   to 1280 (pixels wide) by M (pixels high) where M can be any
//   reasonable value, say 768, 1024, 1280, 4096, etc. The user
//   provides the input image file name and an image output file
//   where the results will be written. The output file is written
//   as a stream of characters where each is a gray scale value
//   of from 0 to 255. The output file is a *.pgm file (raw data).
//        ---------------------------------------------------------------------------------------
//   The mask height and width are selected so that each is one
//   of {3,5,7,9,11,13,15,17,19,21} independently of the other.
//   The user must input the mask. elements.
//
//=================================================================
//-----------------------P R O G R A M-----------------------------------------------------------
//=================================================================
//
//  main();
//     heading();                    display program heading
//        instruct();                displays instructions if selected
//     openfiles();                  opens input & output image files
//     getmask();                    gets convolution mask, multiplier from user
//     applymask();                  does convolution with mask on input image
//        readhdr();                 reads input image file header
//        getstrip();                reads strip of rows from input image file
//        convolve();                performs convolution of mask and row strip
//        writefile();               writes processed row to output image file
//        lastrows();                writes last rows (unprocessed) to output file
//        closefiles();              closes input and processed output image files
//     display();                    displays original and processed image via XV
//-------------------------------------------------------------------------------------------------
# include <stdio.h>
# include <math.h>
```

```
# include <stdlib.h>
//-------------------------------------------------------------------------------------------------------
void     heading(void);
void     openfiles(void);
void     getmask(void);
void     applymask(void);
void     instruct(void);
void     display(void);
//-------------------------------------------------------------------------------------------------------
int      fin[21][1280];
int      gout[21][1280];
float    mask[21][21];                  //values for convolution mask
int      MRows, NCols;                    //indices for rows and cols
int      p, q;                          //height/width of pxq mask
int      rowcount;                       //count of row being processed
FILE     *infptr, *outfptr;              //input/output file pointers
char     infile[40];
char     static  outfile[40];       //names of input/output image files
char     key;                            //key to select instructions
char     display_File1[] = "Original_Image";
char     display_File2[] = "Processed_Image";
int      first_time;
float    factor;                      //multiplier of convolution mask
//-------------------------------------------------------------------------------------------------------
//-----------------------------MAIN--------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------
main()
{ void closefiles();
  char   changekey, stopkey;      //key to process another image or stop
//------------------(Put Heading on Screen)-----------------------------------------------------
  do
  { heading();
    if (key == 'i') instruct();
    first_time = 1;
    do
    { //---------------(Open Input & Output Image Files)---------------------------------------
      openfiles();
      //---------------(Get Mask Entries & Multiplier)-----------------------------------------
      getmask();
      //---------------(Do Convolution on Image with Mask)-------------------------------------
      applymask();
      //---------------(Display Image with XV Program)-----------------------------------------
      display();
      //---------------(Select to Change this Image or Not)------------------------------------
      do
      { printf("\n Process this image again with new parameters (y/n): ?");
        scanf("%1s",&changekey);
      } while ((changekey!='y') && (changekey!='Y') &&
               (changekey!='n') && (changekey!='y'));
    } while ((changekey=='y') || (changekey=='Y'));
```

```
   //------------------(Close Any Open Image Files)-----------------------------------------
   closefiles();
   //----------------(Select Stop or Process Another Image)---------------------------------
   do
   { printf("\n Enter <s> to stop or <i> to process another image ");
     scanf("%1s",&stopkey);
   } while ((stopkey != 's') && (stopkey != 'S') &&
          (stopkey != 'i') && (stopkey != 'I'));
   //-------------------------------------------------------------------------------------------
  } while ((stopkey != 's') && (stopkey != 'S'));
  printf("\n Bye! Bye!\n");
  return;
} //end main()
//-------------------------------------------------------------------------------------------
//---------------------------HEADING-----------------------------------------------------
//-------------------------------------------------------------------------------------------
void heading()
{ int      i;
  for (i=0;i<16;i++) printf("                                    +\n");
  printf("        MDIP Ver. 3.1 - Convolution Mask Digital Image Processing\n");
  printf("                          by Prof. Carl G. Looney\n");
  printf("                       Computer Science Department/171\n");
  printf("                          UNIVERSITY OF NEVADA\n");
  printf("                             Reno, NV 89557\n");
  printf("                           looney@cs.unr.edu\n");
  printf("                            Updated: Jul. 2002\n");
  for (i=0;i<4;i++)  printf("                                    +\n");
  do
  {  printf("\n Enter <i> for instructions or <c> to continue: ");
     scanf("%1s",&key);
  } while ((key != 'i') && (key != 'c') && (key != 'I') && (key != 'C'));
  return;
}//end heading()
//-------------------------------------------------------------------------------------------
//---------------------------INSTRUCT----------------------------------------------------
//-------------------------------------------------------------------------------------------
void instruct(void)
{ printf("\n\n"); printf("\n\n"); printf("\n\n");
  printf("          >> I N S T R U C T I O N S <<"); printf("\n\n");
  printf(" This program processes RAW (packed) DATA *.PGM image files\n");
  printf(" (from XView) for 0 - 255 GRAY LEVELS. The image processing is\n");
  printf(" done via convolution with a pxq MASK that is entered by the\n");
  printf(" user, where p,q are each in {3, 5, 7, 9, 11, 13}. The image\n");
  printf(" file header must have 4 lines, and the third line must give\n");
  printf(" N = number of columns and M = number of rows in the image.\n");
  printf("-----------------------------------------------------------\n\n");
  printf(" P5      (Line 1: P5 => raw packed bytes, P2 => ASCII codes)\n");
  printf(" # ....   (Line 2: a comment)\n");
  printf(" 640 480 (Line 3: no. columns and no. rows)\n");
  printf(" 255       (Line 4: max. no. of gray levels - this may be on line 3 above)\n");
```

```
  printf("-------------------------------------------------------------\n\n");
  printf(" The input .PGM file is processed and the results are written\n");
  printf(" to the .PGM output file. The mask MULTIPLIER is composed of\n");
  printf(" integer inputs for NUMERATOR and DENOMINATOR.\n\n");
  printf(" Please write the mask and multiplier parts on paper and enter.\n");
  printf("-------------------------------------------------------------\n\n");
  printf(" Use a text editor if you need to fix the 4 line header on the\n");
  printf(" image file (e.g., XV may put in an extra comment line!).\n\n");
  return;
}//end instruct()
//-----------------------------------------------------------------------------------------------
//-------------------------OPENFILES---------------------------------------------------------
//-----------------------------------------------------------------------------------------------
void openfiles(void)
{ void  readhdr(void);
  if (first_time == 1)
  { printf("\n               OPEN an image file\n");
    printf("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n");
    printf(" Enter name of *.pgm INPUT image file: ? ");
    scanf("%s",&infile);
    printf(" Enter name of *.pgm OUTPUT image file: ? ");
    scanf("%s",&outfile);
  }
  if ((infptr = fopen(infile, "r")) == NULL)
  { printf(" Can NOT open input image file: <%s>\n",infile);
    printf(" Exiting program..... "); exit(1);
  }
  else  printf(" Input file <%s> opened sucessfully\n\n",infile);
  if ((outfptr = fopen(outfile,"w")) == NULL)
  { printf(" Can NOT open output image file <%s>\n\n",outfile);
    printf(" Exiting program....."); exit(1);
  }
  else printf(" Output file <%s> is opened sucessfully\n\n",outfile);
  readhdr();
  return;
}//end openfiles()
//-----------------------------------------------------------------------------------------------
//-------------------------GETMASK-----------------------------------------------------------
//-----------------------------------------------------------------------------------------------
void getmask()
{ char     redo;                    //key to redo the input mask
  int      i, j;
  float    numer, denom;
  printf(".......+.........+.............+...........+.............+.......\n");
  printf(" Image is to be processed by:\n");
  printf("        -- convolutions MASK with mask MULTIPLIER\n");
  printf("        -- it requires INTEGER entries\n\n");
  do
  { do
    { printf("\n Enter height of convolution mask (3,5,7,9,11,13,15,17,19,21): ?");
```

```
    scanf("%d",&p);
   } while ((p!=3) && (p!=5) && (p!=7) && (p!= 9) && (p!=11) && (p!=13)
              && (p!=15) && (p!=17) && (p!=19) && (p!=21));
   do
   { printf(" Enter width  of convolution mask (3,5,7,9,11,13,15,17,19,21): ?");
    scanf("%d",&q); printf("\n");
   } while ((q!=3) && (q!=5) && (q!=7) && (q!=9) && (q!=11) && (p!=13)
              && (q!=15) && (q!=17) && (q!=19) && (q!=21));
   printf(" Enter mask multiplier NUMERATOR (integer): ?");
   scanf("%f",&numer); printf("\n");
   printf(" Enter mask multiplier DENOMINATOR (integer): ?");
   scanf("%f",&denom); printf("\n\n");
   factor = numer/denom;
   for (i=0;i<p;i++)
   { printf(" Row (INTEGERS) %d::\n",i); printf(" _____\n");
    for (j=0;j<q;j++)
    { printf(" Mask(%d,%d): ? ",i,j);
        scanf("%f",&mask[i][j]);
    }
    printf("\n");
   }
   printf(" User's Convolution Mask (multiplier = %f):\n",factor);
   for (i=0;i<p;i++)
   { for (j=0;j<q;j++) printf("    %f ",mask[i][j]);
    printf("\n");
   }
   printf("+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++\n\n");
   printf(" Hit <a> to accept given mask or <r> to redo it: ?");
   scanf("%1s",&redo); printf("\n\n");
  } while (redo == 'r');
}//end getmask()
//-------------------------------------------------------------------------------------------
//---------------------------APPLYMASK-------------------------------------------------------
//-------------------------------------------------------------------------------------------
void applymask()
{
  void    getstrip(void);
  void    convolve(void);
  void    writefile(void);
  void    lastrows(void);
  void    closefiles(void);

  int     Mminusphalf, Nminusqhalf, Mend;
  int     pm1, phalf, phalfp1;

  pm1 = p - 1; phalf = p/2; phalfp1 = phalf + 1;
  rowcount = 0;
  Mend = MRows - phalf;
  getchar();
  //--------------------[Read Input Image Header]-----------------------------------------------
```

```
  do
  { //------------[Read Strip of p Image Rows to Process]-------------------------------------
    getstrip();
    //-----------------[Convolve Row Strip with Mask]-----------------------------------------
    convolve();
    //--------------[Write Processed Row to Output File]-----------------------------------
    writefile();
    rowcount++;
  } while (rowcount < Mend);
  lastrows();
  closefiles();
  return;
}//end applymask()
//-------------------------------------------------------------------------------------------------
//-----------------------------READHDR-------------------------------------------------------------
//-------------------------------------------------------------------------------------------------
void readhdr()
{
  int     i, k, Maxgrays;
  char    c, c1, buffer[128];
  //-----------------------[Read PGM File Header]-----------------------------------------------
  printf("\n\n File <%s> Header Bytes:\n",infile);
  printf("-------------------------------------------------------------\n");
  k = 0;
  do
  { i = 0;
    do
    { c = fgetc(infptr);
      buffer[i] = c; i++;
    } while (c != '\n');
    if (k == 0)
    { c1 = buffer[1];
      if (c1 == '5')
      {
        printf("\n File is:  <P%c>\n",c1);
      }
      else
      { printf(" Image in WRONG format!! Quitting.....\n\n");
        exit(0);
      }
    }
    buffer[i] = '\0'; k++;
    fprintf(outfptr,"%s",buffer);
    printf("%s",buffer);
  } while (k < 2);
  fscanf(infptr,"%d %d %d",&NCols, &MRows, &Maxgrays); c = fgetc(infptr);
  fprintf(outfptr,"%d %d", NCols, MRows);
  fprintf(outfptr,"%c %d %c",'\n', Maxgrays, '\n');
  printf(" %d ",NCols);
  printf(" %d                <-----(Width & Height)\n", MRows);
```

```
  printf(" %d                    <-----(Max. Gray Level)\n\n",Maxgrays);
}//end readhdr()
//-------------------------------------------------------------------------------------------------------
//-----------------------------GETSTRIP--------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------
void getstrip()
{ int      row, col, rowp1, pm1, qm1;
  unsigned char    item;                    //read in pixel as char
  pm1 = p - 1; qm1 = q-1;
  if (rowcount == 0)                    //in case of first base row,
  { for (row=0;row<p;row++)             //read in p rows from input file
    { for (col=0;col<NCols;col++)          //cols. from 0 to end of row
      {   item = fgetc(infptr);              //read in binary byte
          fin[row][col]  = (int) item;
          gout[row][col] = (int) item;
      }
    }
  }
  if (rowcount != 0)                //in case of second or greater row
  { for (row=0;row<pm1;row++)     //shift rows up 1 line from bottom row
    { rowp1 = row + 1;
      for (col=0;col<NCols;col++)
      { fin[row][col] =  fin[rowp1][col];
      }
    }
    for (col=0;col<NCols;col++)             //now read in new bottom row
    { item = fgetc(infptr); fin[pm1][col] = (int) item;
    }
    printf(" . ");
  }
}//end getstrip()
//-------------------------------------------------------------------------------------------------------
//-----------------------------CONVOLVE--------------------------------------------------------
//-------------------------------------------------------------------------------------------------------
void convolve()
{  float    fpixel;                      //float pixel output value
   int      row, col;
   int      Ncount, colcount;
   int      ncol, pixelcol;
   int      pixelsum, pixnum;              //pixel sum for convolution
   int      phalf, qhalf, Nend;
   int      Nminusqhalf;
   phalf = p/2; qhalf = q/2;
   Nend  = NCols - qhalf - 1;
   Nminusqhalf = NCols - qhalf;
   colcount = 0;                      //base column of current strip
   do
   { fpixel = 0;  Ncount = colcount + q;
     for (row=0;row<p;row++)             //for p rows: pxq pixel block
     { for (col=colcount;col<Ncount;col++)      //do convolution process
```

```
      { ncol = col - colcount;                  //pixels of base block
        fpixel = fpixel + mask[row][ncol]*fin[row][col];
      }
    }
    fpixel = factor*fpixel + 0.5001;           //get processed pixel
    pixnum = (int) fpixel;
    if (pixnum < 0) pixnum = 0;
    if (pixnum > 255) pixnum = 255;
    pixelcol = colcount + qhalf;              //get center col. of block
    gout[phalf][pixelcol] = pixnum;        //write to output image file
    if ((rowcount < 1) && (colcount < 1))
    { printf("\n First Processed Block:\n");
      for (row=0;row<p;row++)
      { for (col=0;col<q;col++)
        { printf("  %d  ",fin[row][col]);
            if (col == q-1) printf("\n");
          }
      }
      printf("\n");
      printf(" Old Pixel = %d    ", fin[phalf][pixelcol]);
      printf(" New Pixel = %d\n\n", gout[phalf][pixelcol]);
      printf(" Hit <ENTER> to continue!\n");
      getchar();
    }
    colcount++;
  } while (colcount < Nend);
  //---------[complete the unprocessed ends of processed row]----------
  for (col=0;col<qhalf;col++)
  { gout[phalf][col] = fin[phalf][col];  //first pixels to block center
  }
  for (col=Nminusqhalf;col<NCols;col++)
  { gout[phalf][col] = fin[phalf][col];     //last pixels to end of row
  }
}//end convolve()
//-------------------------------------------------------------------------------------------------
//----------------------------WRITEFILE--------------------------------------------------
//-------------------------------------------------------------------------------------------------
void writefile()
{ int       row, col;
  int       phalf, qhalf, NColsmqhalf;
  int       pixchar;
  phalf = p/2; qhalf = q/2; NColsmqhalf = NCols - qhalf;
  //-------------[write first p/2 - 1 rows to output file]--------------
  if (rowcount == 0)              //on first pass, write 1st half strip
  { for (row=0;row<=phalf;row++)           //to output file at beginning
    { for (col=0;col<NCols;col++)
      { pixchar = gout[row][col];
        fprintf(outfptr,"%c", (char) pixchar);
      }
    }
```

```
  }
  //---------------[write processed row to output file]-----------------------------------------
  if (rowcount > 0)
  { for (col=0;col<NCols;col++) //always write processed pixels to outfile
    { pixchar = gout[phalf][col];
      fprintf(outfptr,"%c", (char) pixchar);
    }
  }
} //end writefile()
//-------------------------------------------------------------------------------------------------
//-----------------------------LASTROWS--------------------------------------------------------
//-------------------------------------------------------------------------------------------------
void lastrows()
{ int    i, j;
  int    Mm1, phalf, phalfp1;
  phalf = p/2; Mm1 = MRows - 1;
  phalfp1 = phalf + 1; printf("\n\n");
  //-----------------[read/write last few rows of image]----------------
  printf(" Rows 0 - %d are processed/written to output file\n\n", Mm1);
  printf(" Closing file: %s\n",outfile);
  for (i=phalfp1;i<p;i++)
  { for (j=0;j<NCols;j++)
    { fprintf(outfptr,"%c", (char) fin[i][j]);
    }
  }
  return;
}//end lastrows()
//-------------------------------------------------------------------------------------------------
//----------------------------CLOSEFILES-----------------------------------------------------------
//-------------------------------------------------------------------------------------------------
void closefiles()
{ //-------------------------(Close Files)-----------------------------------------------------
  fclose(infptr);
  fclose(outfptr);
  return;
} //end closefiles()
//-------------------------------------------------------------------------------------------------
//-------------------DISPLAY—(comment out this for MS Visual or Borland C++-----------
//-------------------------------------------------------------------------------------------------
void display()
{ char    buffer1[100], buffer2[100];
//---------------Part 1: Copy and Display Original Image-------------------------------------
 //copy original image to display_File1 for displaying on first time
  if (first_time == 1)
  { sprintf(buffer1," %s %s %s "," cp ", infile, display_File1);
    system(buffer1);
    system("sleep 5");
    //----(Display Original Image at (y,x)-----------------------------------------------------
    sprintf(buffer2," %s %s %s "," xv -geometry +2-2 ", display_File1,"&");
    system(buffer2);
```

```
  }
  //-------------------------------------------------------------------------------------------------------
  //------------Part 2: Copy and Display Processed Image-----------------------------------------
  //-------------------------------------------------------------------------------------------------------
  sprintf(buffer1," %s %s %s "," cp ", outfile, display_File2);
  system(buffer1);
  system("sleep 2");
  //display processed image on first time this function is called
  if (first_time == 1)
  { sprintf(buffer2," %s %s %s "," xv -poll -geometry +200-200 ", display_File2," & ");
    system(buffer2);
    //------Turn first_time off------
    first_time = 0;
  }
  printf("\n Move images to desired position on screen with mouse!\n");
  return;
} //end display()
//-------------------------------------------------------------------------------------------------------
```