Lab2

BINARY IMAGE PROCESSING REPORT

Xinyi (Blanca) Bai

Cornell University Electrical and Computer Engineering

Table of Contents

1. Introduction

In this lab I gain experience with the basic VisionX utility commands and the VisionX programming tools. The first three sections of this lab provide an introduction by example of the software tools that are available. These command and results will be descripted in the last sections in this report. In the last two sections of lab2, I use these tools to develop and test two binary image processing algorithms: bound algorithm and cclable algorithm, the description and implementation is in section 2 and 3 in this report.

2. Bound Program

2.1 Algorithm Description

All pixels can be divided into three kinds, they are background, innor and border.

1) Embed the image im into the image tm with a one pixel border
2) Traverse the image tm(with boder)
3) When arrive at point[y][x], check if the pixel value in tm is 0 and the 4 border pixel of it is 0, if thery are all 0, so this is background, I set this pixel value equal to 0 in image im.
4) When arrive at point[y][x], check if the pixel value in tm is non-0, if it is, so this pixel is innor, I set this pixel value equal to 128 in muage im.
5) When arrive at point[y][x], if the above to check all failed, so this pixel is border, I set this pixel value equals 255 in image im.

2.2 Implementation

I write a program called bound.c that accepts a segmented image as input and generates an image in which the segment boundary pixels are set to 255, interior pixels are set to 128 and the background remains set to zero.

Pseudo code is:

```
  for (y = im.ylo ; y <= im.yhi ; y++) {//traverse image contains border
     for (x = im.xlo; x <= im.xhi; x++)  {//traverse image contains border
          if(tm.u[y][x] == 0 && tm.u[y + 1][x] == 0 && tm.u[y][x + 1] == 0
&& tm.u[y - 1][x] == 0 && tm.u[y][x - 1] == 0){ //background
            im.u[y][x] = 0; }
          else if(tm.u[y][x] != 0){ //innor area
            im.u[y][x] = 128;}
          else{
            im.u[y][x] = 255;}   }   } //boder
```

2.3 Results on small image

The pixel values of the input small image is:

```
[en-ec-ecelinux-10:~/lab2 [12:04am] 85$vppr testim1.vx
      0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
 15   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 14   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 13   0   0  10  10  10  10  10  10  10  10  10  10  10  10   0   0
 12   0   0  10  20  20  20  20  20  20  20  20  20  20  10   0   0
 11   0   0  10  20  30  30  30  30  30  30  30  30  20  10   0   0
 10   0   0  10  20  30  30  30  30  30  30  30  30  20  10   0   0
  9   0   0  10  20  30  30  30  30  30  30  30  30  20  10   0   0
  8   0   0  10  20  30  30  30  30  30  30  30  30  20  10   0   0
  7   0   0  10  20  30  30  30  40  30  30  30  30  20  10   0   0
  6   0   0  10  20  30  30  30  30  30  30  30  30  20  10   0   0
  5   0   0  10  20  30  30  30  30  30  30  30  30  20  10   0   0
  4   0   0  10  20  30  30  30  30  30  30  30  30  20  10   0   0
  3   0   0  10  20  20  20  20  20  20  20  20  20  20  10   0   0
  2   0   0  10  10  10  10  10  10  10  10  10  10  10  10   0   0
  1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

FIG 1. Pixel value of bound input small test image.

The object is non-zero pixel value area.

```
[en-ec-ecelinux-10:~/lab2 [12:07am] 86$vppr testim1.mx
      0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
 15   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 14   0   0 255 255 255 255 255 255 255 255 255 255 255 255   0   0
 13   0 255 128 128 128 128 128 128 128 128 128 128 128 128 255   0
 12   0 255 128 128 128 128 128 128 128 128 128 128 128 128 255   0
 11   0 255 128 128 128 128 128 128 128 128 128 128 128 128 255   0
 10   0 255 128 128 128 128 128 128 128 128 128 128 128 128 255   0
  9   0 255 128 128 128 128 128 128 128 128 128 128 128 128 255   0
  8   0 255 128 128 128 128 128 128 128 128 128 128 128 128 255   0
  7   0 255 128 128 128 128 128 128 128 128 128 128 128 128 255   0
  6   0 255 128 128 128 128 128 128 128 128 128 128 128 128 255   0
  5   0 255 128 128 128 128 128 128 128 128 128 128 128 128 255   0
  4   0 255 128 128 128 128 128 128 128 128 128 128 128 128 255   0
  3   0 255 128 128 128 128 128 128 128 128 128 128 128 128 255   0
  2   0 255 128 128 128 128 128 128 128 128 128 128 128 128 255   0
  1   0   0 255 255 255 255 255 255 255 255 255 255 255 255   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

FIG 2. Pixel value of the bounded output small test image.

The border of object is the countour which in 255 pixel value; the innor area is in 128 pixel value; and the background is in 0 pixel value.

To visualize the result, i add caption to both input image and output image, then print them with less ink, next I change the format of input image and output image form .vx to .png, with the following command:

```
vpix -neg im1.vx | vcapt c="input small image"    of=boundinputsmall.vx

vpix -neg im1.mx | vcapt c="bound processed smallnimage" of=boundoutputsmal
l.vx
```

```
vxport -png boundinputsmall.vx

vxport -png boundoutputsmall.vx
```

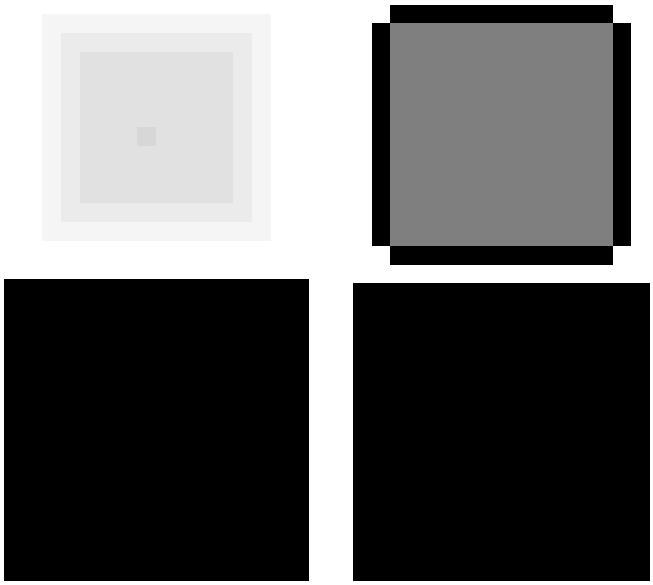


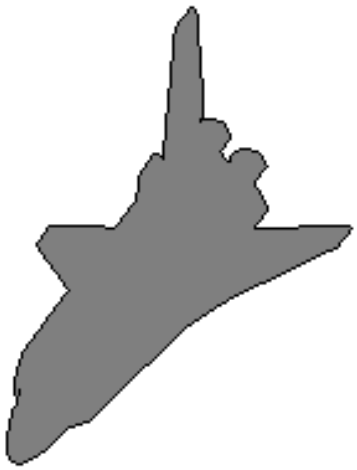
FIG 3 (left). The input small image. FIG 4 (right) the bounded output small image.
The black part in both image is caused by caption, because the image is so small, and the caption contains so many words.
From the results, my implementation of bound algorithm works.

2.4 Results on full-sized image

```
bound shuttle.vx of= shuttle.mx
```

To visualize the result on full-sized image, I add caption to both input image and output shuttle image, then print them with less ink, next I change the format of input image and output image form .vx, .mx to .png, with the commands similar to the commands for small tests. Results are shown below:

FIG 5. The input full-size image



bound processed shuttle image

FIG 6. The bounded output full-size image

When I convert the image fromat to png, the boder become dark, the innor part become gray, the background become white, which is different from the preview of .vx format in vview.

The preview screenshot in vview is:

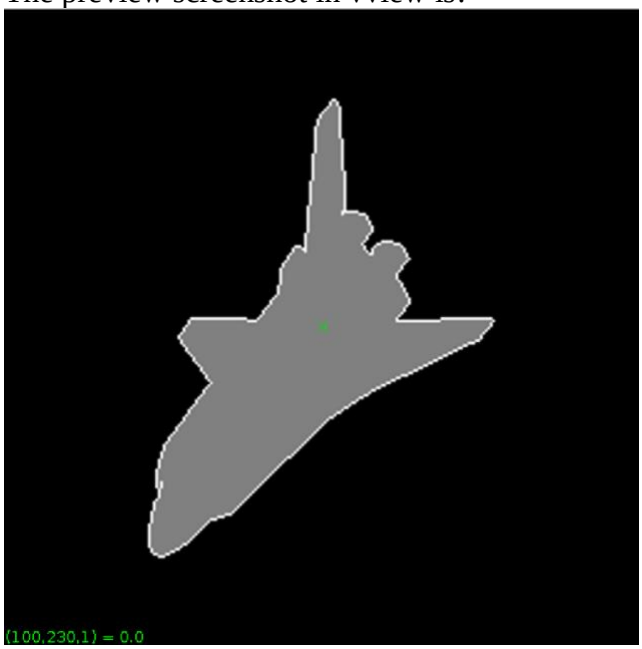


FIG 7. Screenshot of the preview of the bounded full-size image in vview

The color of boder and background is converted, because of some characteristic of PNG image, the PNG file is an image file stored in the Portable Network Graphic (PNG) format. It contains a bitmap of indexed colors and uses lossless compression, the background of PNG image is transparent.

All tests are passed, which show my bound implementation works well.

3. Cclabel Program

3.1 Algorithm Description

1) Embed the image im into the image tm with a one pixel border.

   NOTE: Declare the im, vm and tm as global variable. In this way you do not need to pass them as arguments to your subprocedure setlabel

2) Clear the output (label) image, im set all pixel value in im equals 0, after clear. Then copy the image im as vm to avoid "Segmentation default" error, then use the image vm for the label (output) image

3) Set the current label to 20.

4) Search the input image, tm, until an object pixel is found.

5) If the object pixel is not labeled (i.e. the output image, im, is zero for this pixel), then call the recursive function setlabel(i, j, n)
   where (i, j) is the object pixel location and n is the next label value.

6) Increment the label value by 30.

7) Repeat the steps 4, 5 and 6 for all unlabeled object pixels.

For recursive function setlabel(y, x, h) is

1) Set the output image pixel at (y, x) to the label h, (vm(y, x) = h)

2) If the pixel above the given pixel is an object pixel and is not labeled (vm(y, x+1)==0) then call setlabel(y, x+1, h)

3) If the pixel below the given pixel is an object pixel and is not labeled (vm(y, x-1)==0) then call setlabel(y, x-1, h)

4) If the pixel left of the given pixel is an object pixel and is not labeled (vm(y-1, x)==0) then call setlabel(y-1, x, h)

5) If the pixel right of the given pixel is an object pixel and is not labeled (vm(y+1, x)==0) then call setlabel(y+1, x, h)

3.2 Implementation

I write a program called cclabel.c that accepts a segmented image as input and generates a labeled output image in which all pixels of a given connected component are assigned a unique number. The label numbers is in sequence, start from 20, and increment 30 every time. Where n is the number of connected components in the input image. The program cclabel use a recursive subprocedure called setlabel.

Pesudo codes are:
First I preset all pixels in im to 0, which implies unlabled

```
  for (y = im.ylo ; y <= im.yhi ; y++){
     for (x = im.xlo ; x <= im.xhi ; x++){
       vm.u[y][x] = 0; }}
```

when meet object and check isunlabled in im, then use lable function to lable it
in detail, it is when we find the pixel in vm is unlabeled, which is value = 0, as well we this pixel in tm, the image we traversed, the value is non zero, so we label it,

```
   int i = 20;
   for (y = im.ylo ; y <= im.yhi ; y++){
     for(x = im.ylo ; x <= im.xhi ; x++){
       if(tm.u[y][x] != 0 && vm.u[y][x] == 0){
         setlabel(y, x, i);
         i=i+30; }}}
```

copy image im as vm, and lable on image vm to avoid “Segmentation default” error

```
 for (y = im.ylo ; y <= im.yhi ; y++){

     for(x = im.ylo ; x <= im.xhi ; x++){

        im.u[y][x] = vm.u[y][x];}}
```

implement setlabel function recursively,

```
void setlabel(int y, int x, int h){
  vm.u[y][x] = h;
  if(vm.u[y][x + 1] == 0 && tm.u[y][x + 1] != 0){
    setlabel(y, x + 1, h);}
  if(vm.u[y][x - 1] == 0 && tm.u[y][x - 1] != 0){
    setlabel(y, x - 1, h);}
  if(vm.u[y - 1][x] == 0 && tm.u[y - 1][x] != 0){
    setlabel(y - 1, x, h);}
  if(vm.u[y + 1][x] == 0 && tm.u[y + 1][x] != 0){
    setlabel(y + 1, x, h);}}
```

3.3 Results on small image

The pixel values of the input small image is:

```
[en-ec-ecelinux-10:~/lab2 [1:41pm] 119$vppr testim2.vx
      0   1   2   3   4   5   6   7   8   9  10  11
  9   0   0   0   0   0   0   0   0   0   0   0   0
  8   0 255 255 255 255   0   0   0   0 255   0   0
  7   0 255 255   0 255   0   0   0 255 255 255   0
  6   0 255 255 255 255 255   0   0 255 255 255   0
  5   0 255 255 255 255 255   0   0 255 255 255   0
  4   0 255 255 255   0   0   0   0 255 255 255   0
  3   0 255 255   0   0 255 255 255 255 255 255   0
  2   0   0   0   0 255   0   0   0   0   0   0   0
  1   0   0   0 255   0   0 255 255 255   0   0   0
  0   0   0   0   0   0   0 255 255 255   0   0   0
```

FIG 8. Pixel value of cclable input small test image.

This small image contains 5 seperated objects, it is easy to tell from our naked eyes.

```
[en-ec-ecelinux-10:~/lab2 [1:36pm] 118$vppr test.vx
      0   1   2   3   4   5   6   7   8   9  10  11
  9   0   0   0   0   0   0   0   0   0   0   0   0
  8   0 110 110 110 110   0   0   0   0 140   0   0
  7   0 110 110   0 110   0   0   0 140 140 140   0
  6   0 110 110 110 110 110   0   0 140 140 140   0
  5   0 110 110 110 110 110   0   0 140 140 140   0
  4   0 110 110 110   0   0   0   0 140 140 140   0
  3   0 110 110   0   0 140 140 140 140 140 140   0
  2   0   0   0   0  80   0   0   0   0   0   0   0
  1   0   0   0  50   0   0  20  20  20   0   0   0
  0   0   0   0   0   0   0  20  20  20   0   0   0
```

FIG 9. Pixel value of cclable output small test image.

The first object is in 20 pixel value, the second object is in 50 pixel value, the third object is in 80 pixel value, the fourth object is in 110 pixel value, and the last object is in 140 pixel value. The results are in accordance with the pixel value I set in my program, which is start from 20, and increment 30 each time.

To visualize the result on full-sized image, I add caption to both input image and output image, then print them with less ink, next I change the format of input image and output image form .vx to .png, command are as below:

```
en-ec-ecelinux-10:~/lab2 [1:41pm] 120$vpix -neg testim2.vx | vcapt c="input
 image testim2.vx"    of=cclabeltestim2.vx
en-ec-ecelinux-10:~/lab2 [1:48pm] 123$vxport -png cclabeltestim2.vx
```

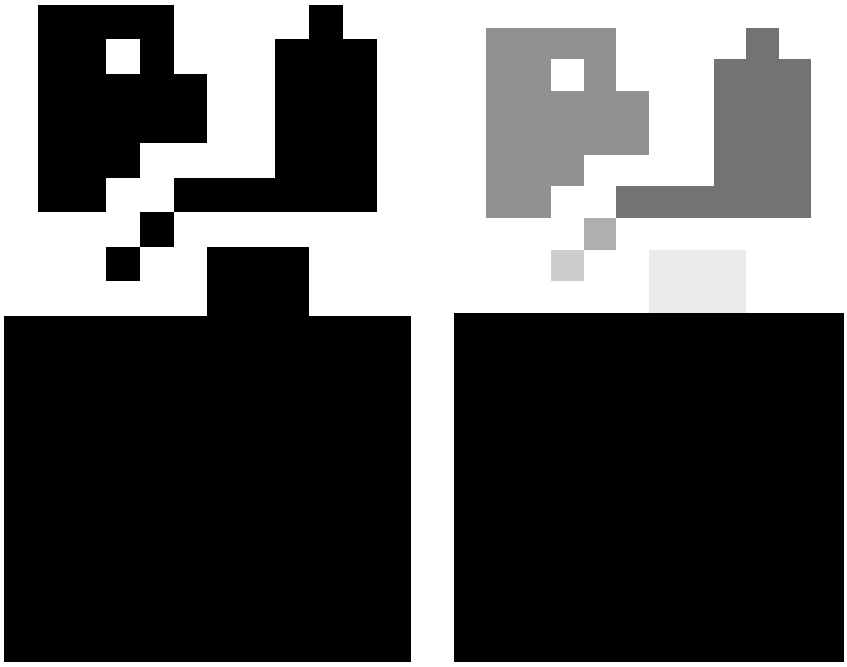


FIG 10 (left). The input small image. FIG 11 (right) the cclabeled output small image. The black part in both image is caused by caption, because the image is so small, and the caption contains so many words.
From the results, my implementation of cclabel algorithm works.

3.4 Results on full-sized image

To visualize the result on full-sized image, I add caption to both input image and output shuttle image, then print them with less ink, next I change the format of input image and output image form .vx to .png, with the commands below:

```
en-ec-ecelinux-10:~/lab2 [1:55pm] 128$cclabel if=im1.mx of=cclabeledletter
s.vx
en-ec-ecelinux-10:~/lab2 [1:59pm] 132$vpix -neg im1.vx | vcapt c="input ccl
abeled im1.vx" of=cclabeledim1input.vx
en-ec-ecelinux-10:~/lab2 [DING!] 133$vxport -png cclabeledim1input.vx
```

FIG 11. The input full-size image

```
en-ec-ecelinux-10:~/lab2 [1:56pm] 129$vpix -neg cclabeledletters.vx | vcapt
 c="cclabeled im1.vx" of=cclabeledim1.vx
en-ec-ecelinux-10:~/lab2 [1:57pm] 131$vxport -png cclabeledim1.vx
```

FIG 12. The cclabeled putput full-size image

In the Fig 12, we can tell there are 3 objects in this full-sized image, in light-grey color(pixel value = 20), in middle-gray color(pixel value = 50), in gray color(pixel value = 80).

In order to make the comparison more obvious, I set the incremental pixel value to 30 instead of 1, which is a smart advice form our teaching assistant.

Appendix 1 Source Code of two program :

```
/* vtemp      Compute local max operation on a single byte image   */
/*****************************************************************/

#include "VisXV4.h"           /* VisionX structure include file    */
#include "Vutil.h"            /* VisionX utility header files      */

VXparam_t par[] =             /* command line structure            */
{ /* prefix, value,   description                        */
{    "if=",    0,    " input file  vtemp: local max filter "},
{    "of=",    0,    " output file "},
{     0,       0,    0}  /* list termination */
};
#define  IVAL   par[0].val
#define  OVAL   par[1].val

main(argc, argv)
int argc;
char *argv[];
{
Vfstruct (im);                    /* i/o image structure          */
Vfstruct (tm);                    /* temp image structure         */
```

```
int        y,x;                    /* index counters               */
  VXparse(&argc, &argv, par);       /* parse the command line       */

  Vfread(&im, IVAL);                /* read image file              */
  Vfembed(&tm, &im, 1,1,1,1);       /* image structure with border  */
  if ( im.type != VX_PBYTE ) {      /* check image format           */
     fprintf(stderr, "vtemp: no byte image data in input file\n");
     exit(-1);
  }
  for (y = im.ylo ; y <= im.yhi ; y++) {  /* compute the function */
     for (x = im.xlo; x <= im.xhi; x++)  {     /***********************/
          if(tm.u[y][x] == 0 && tm.u[y + 1][x] == 0 && tm.u[y][x + 1] == 0 &&
tm.u[y - 1][x] == 0 && tm.u[y][x - 1] == 0){
            im.u[y][x] = 0;
          }

          else if(tm.u[y][x] != 0){
            im.u[y][x] = 128;
          }

          else{
            im.u[y][x] = 255;
          }
     }
   }

   Vfwrite(&im, OVAL);             /* write image file                */
   exit(0);
}


/*****************************************************************/

#include "VisXV4.h"           /* VisionX structure include file    */
#include "Vutil.h"            /* VisionX utility header files      */

VXparam_t par[] =             /* command line structure            */
{ /* prefix, value,   description                          */
{    "if=",    0,    " input file  vtemp: local max filter "},
{    "of=",    0,    " output file "},
{     0,       0,    0}  /* list termination */
};
#define  IVAL    par[0].val
#define  OVAL    par[1].val
/*void lmax(int, int); */
/* Blanca add the declaration of setlable function */
```

```
void setlabel(int, int, int);

Vfstruct (im);                      /* i/o image structure          */
Vfstruct (tm);                      /* temp image structure         */
Vfstruct (vm);
main(argc, argv)
int argc;
char *argv[];
{
int        y,x;                     /* index counters               */
  VXparse(&argc, &argv, par);       /* parse the command line       */

  Vfread(&im, IVAL);                /* read image file              */
  Vfembed(&tm, &im, 1,1,1,1);       /* image structure with border  */
  Vfembed(&vm, &im, 1,1,1,1);
  if ( im.type != VX_PBYTE ) {      /* check image format           */
     fprintf(stderr, "vtemp: no byte image data in input file\n");
     exit(-1);
  }
   for (y = im.ylo ; y <= im.yhi ; y++){
     for (x = im.xlo ; x <= im.xhi ; x++){
       vm.u[y][x] = 0;
     }
   }

   /* when meet object and check isunlabled in im, then use lable function to
lable it    */
   int i = 20;
   for (y = im.ylo ; y <= im.yhi ; y++){
     for(x = im.ylo ; x <= im.xhi ; x++){
       if(tm.u[y][x] != 0 && vm.u[y][x] == 0){
         setlabel(y, x, i);
         i=i+30;

     }
}
}

 for (y = im.ylo ; y <= im.yhi ; y++){
     for(x = im.ylo ; x <= im.xhi ; x++){
       im.u[y][x] = vm.u[y][x];
       }
}
   //Blanca's code -->

   Vfwrite(&im, OVAL);              /* write image file               */
```

```
   exit(0);
}


//<-- Blanca's code
/* setlabel(y, x, h) function here   */
void setlabel(int y, int x, int h){
  vm.u[y][x] = h;
  if(vm.u[y][x + 1] == 0 && tm.u[y][x + 1] != 0){
    setlabel(y, x + 1, h);
  }
  if(vm.u[y][x - 1] == 0 && tm.u[y][x - 1] != 0){
    setlabel(y, x - 1, h);
  }
  if(vm.u[y - 1][x] == 0 && tm.u[y - 1][x] != 0){
    setlabel(y - 1, x, h);
  }
  if(vm.u[y + 1][x] == 0 && tm.u[y + 1][x] != 0){
    setlabel(y + 1, x, h);
  }
}

//Blanca's code -->
```

Appendix 2: all implementations and experiments in this lab

Some comman used VisionX command:

```
unsigned char A[N][M];
        short B[N][M];
        int i,j;
        for ( i=0, i< N, i++ ) {
             for ( j=0, j< M, j++ ) {
                 A[i,j] = B[i,j];
             }
        }
```

```
1)       Vfstruct (Aim); /* declare the image structures */
2)       Vfstruct (Bim);
3)       int y,x;
4)       /* ... read in image Bim and declare the structure for Aim.. */
5)       /* in this process the size, type, and channels of the       */
6)       /* images are dynamically specified.                         */
7)       for ( y=Bim.ylo, y<=Bim.yhi, y++ ) {
8)           for ( x=Bim.xlo, x <=Bim.xhi, x++ ) {
9)                  Aim.u[y][x] = Bim.s[y][x];
10)          }
11)      }
```

```
Vfread(&im, IVAL);          /* read file and initialize input structure */
Vfembed(&tm, &im,1,1,1,1); /* image structure with border              */
```

The first statement reads an input image "im" from the input file. The second statement declares the structure "tm" and copies the image data from "im". Vfembed also adds a border of elements (set to 0) around the edge of the original image (as specified by the last four arguments of "1"). That is, "tm" has two addition rows and two additional columns with respect to "im".

The main program will compute a function from the padded image "tm" and store it in "im" and then write out the original image data structure "im" (with modified contents).

The Template Program vtemp.c

```
/*****************************************************************/
/* vtemp      Compute local max operation on a single byte image   */
/*****************************************************************/

#include "VisXV4.h"           /* VisionX structure include file    */
#include "Vutil.h"            /* VisionX utility header files      */

VXparam_t par[] =             /* command line structure            */
{ /* prefix, value,   description                          */
{    "if=",    0,   " input file  vtemp: local max filter "},
{    "of=",    0,   " output file "},
{     0,       0,   0}  /* list termination */
};
#define  IVAL   par[0].val
#define  OVAL   par[1].val

main(argc, argv)
int argc;
char *argv[];
{
Vfstruct (im);                      /* i/o image structure          */
Vfstruct (tm);                      /* temp image structure         */
int        y,x;                     /* index counters               */
  VXparse(&argc, &argv, par);       /* parse the command line       */

  Vfread(&im, IVAL);                /* read image file              */
  Vfembed(&tm, &im, 1,1,1,1);       /* image structure with border  */
  if ( im.type != VX_PBYTE ) {      /* check image format           */
     fprintf(stderr, "vtemp: no byte image data in input file\n");
     exit(-1);
   }
   for (y = im.ylo ; y <= im.yhi ; y++) {  /* compute the function */
      for (x = im.xlo; x <= im.xhi; x++)  {     /*********************/
           im.u[y][x] = MAX(tm.u[y][x],         /* You only need to     */
                        MAX(tm.u[y][x+1],     /* change this section  */
                          MAX(tm.u[y+1][x],   /* for your program     */
                            MAX(tm.u[y][x-1], /*********************/
                                 tm.u[y-1][x])));
      }
   }

   Vfwrite(&im, OVAL);              /* write image file               */
   exit(0);
}
```

```
[ECE2400: ~/lab2 % . ~/e4                                                    ]
[en-ec-ecelinux-08:~/lab2 [3:48pm] 11$ls                                     ]
 im1.vx@     ptest*      testim1*    tst.a@      vtemp.c@
 im2.vx@     ptestpy*    testim2*    vtemp2.c@   vtempm*
 im3.vx@     shuttle.vx@ testim3*    vtemp2y*    vtempy*
[en-ec-ecelinux-08:~/lab2 [3:48pm] 12$vl                                     ]
   shuttle.vx: V4.pu: vcp shuttle.v3 -o shuttle.vx  (256x256)
       im3.vx: V4.pu: vcp im3.v3 -o im3.vx  (256x256)
       im2.vx: V4.pu: vcp im2.v3 -o im2.vx  (256x50)
       im1.vx: V4.pu: vcp im1.v3 -o im1.vx  (256x56)
[en-ec-ecelinux-08:~/lab2 [3:49pm] 13$vview &                                ]
 [1] 49977
 en-ec-ecelinux-08:~/lab2 [3:49pm] 14$Fontconfig warning: ignoring UTF-8: not a v
 alid region tag
```

```
[vls -l im1.vx
       im1.vx: V4.pu: vcp im1.v3 -o im1.vx  (256x56)
 [ im1.vx]
           created by : [21216]
           created on : Fri Jul  8 15:51:00 2005
              machine : linux
              history :
vtrim facsimile yl=51 yh=106
USC: facsimile
vpix th=200
vpix -neg of=im1.vx
```

After this command, visionX windows dropped out

vdview im1.vx

In vview double click on the file names to view the images. To see the image properties use "Tools→ Image Statistics"

Command	Action
vls [-l] [files]	provide information on VisionX files
vl [-l]	list VisionX files in the current directory (in the order that they were created
vdview	display a VisionX image file

To access the user manual information for any visionx command go to "Help→ vman VX commands", scroll or search for the command name and double click on it.

Compiling and Testing a (VisionX) program

Compile the template program vtemp

```
vcc vtemp.c -o vtemp
```
Test it with image data
```
vtemp im1.vx of=im1.mx
```

```
vview
```
then click on "im1.vx" to see the original image, as below:

Click on "im1.mx" to see our test result, as below:

Explore the use of scripts for program development. First review the script testim1
```
less testim1
```

```
#!/bin/sh
#
# script to generate the test image testim1.vx

pid=$$
vgenim s=16 -im of=tmp1.$pid hi=10
vgenim s=16  r=4,4 hi=10 | vop -add tmp1.$pid  of=tmp2.$pid
vgenim s=16  r=5,5 hi=10 | vop -add tmp2.$pid  of=tmp1.$pid
vgenim s=16  r=6,6 hi=10 | vop -add tmp1.$pid  of=testim1.vx

~
~
~
~
~
~
~
~
~
~
~
~
~
testim1 lines 1-10/10 (END)
```

Then execute it to create the image testim1.vx

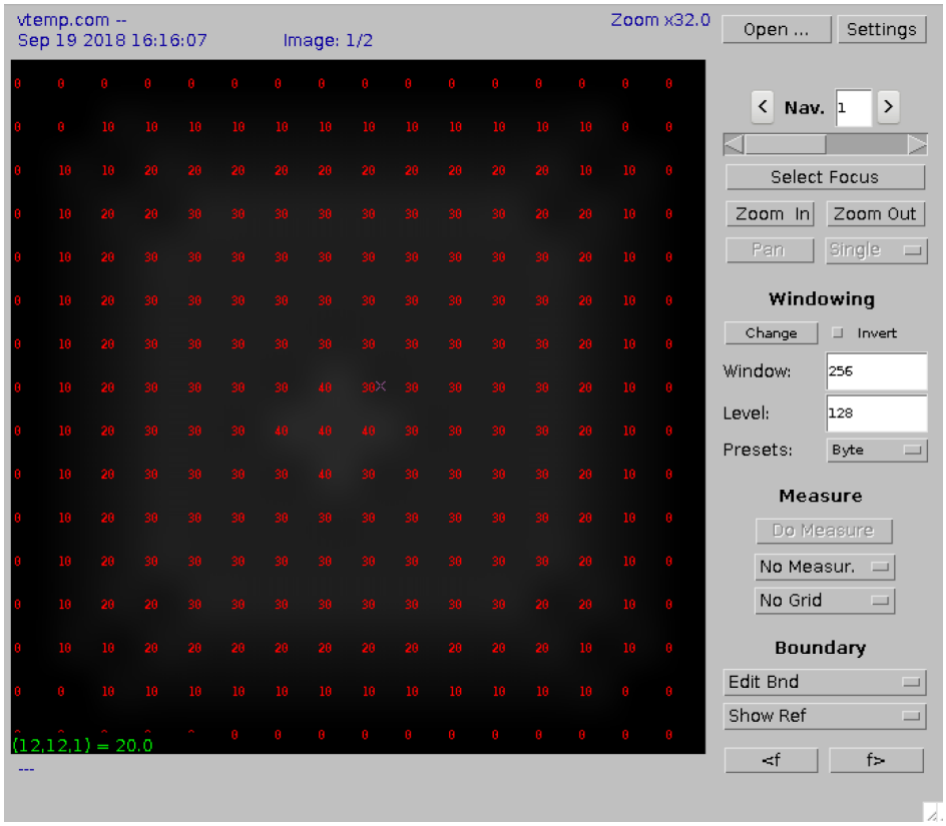
```
testim1
```

Review the script "ptest" which is designed to: (a) compile a program, (b) execute that program on a specified image, (c) create a composite image that is suitable for visual review of output vs input and (d) display that image for your review.

```
less ptest
```

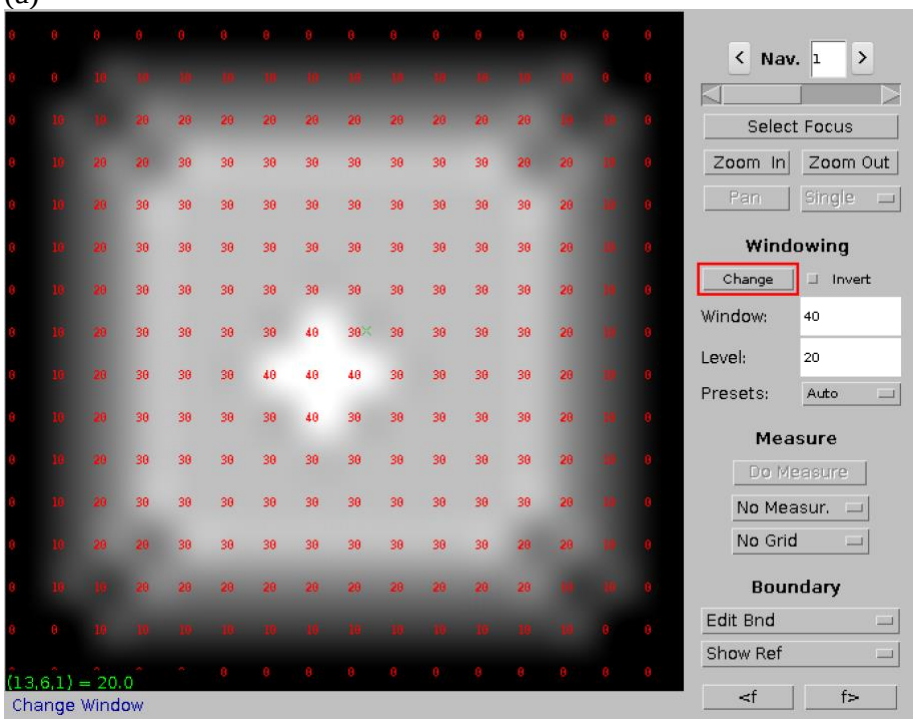
Run the script on your program with the test image.

```
ptest testim1.vx  vtemp
```

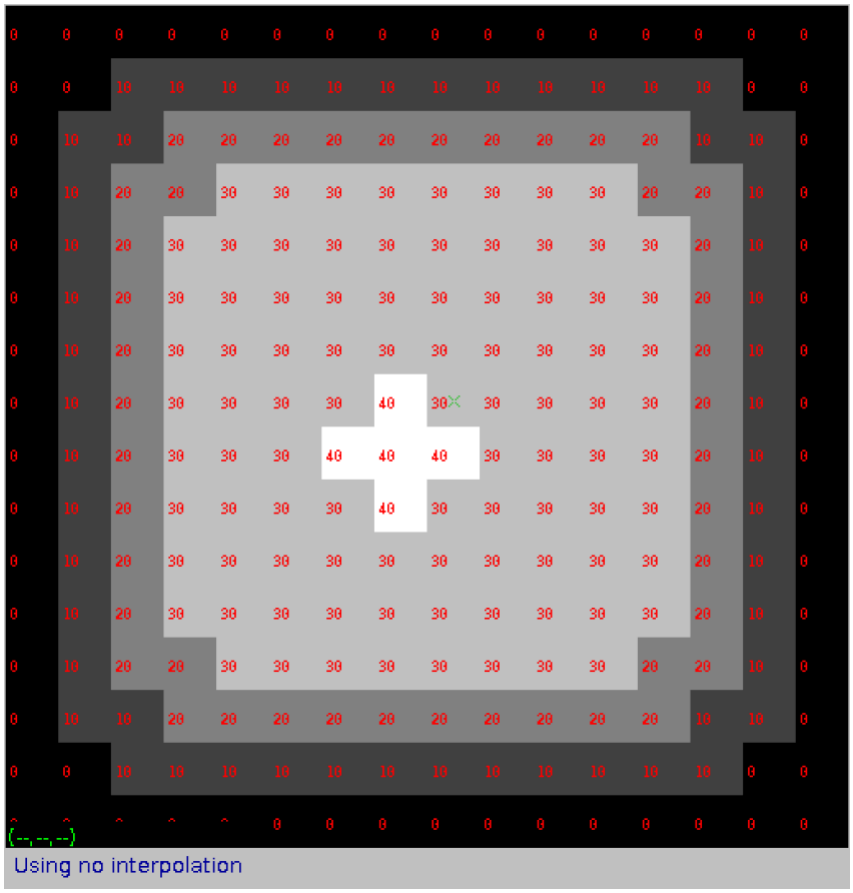


In the image that is displayed (a) set the window "Presets:" to "Auto",(b) in "settings" select "No Interpolation" and (c) compare the input and output images by rotating the mouse wheel.

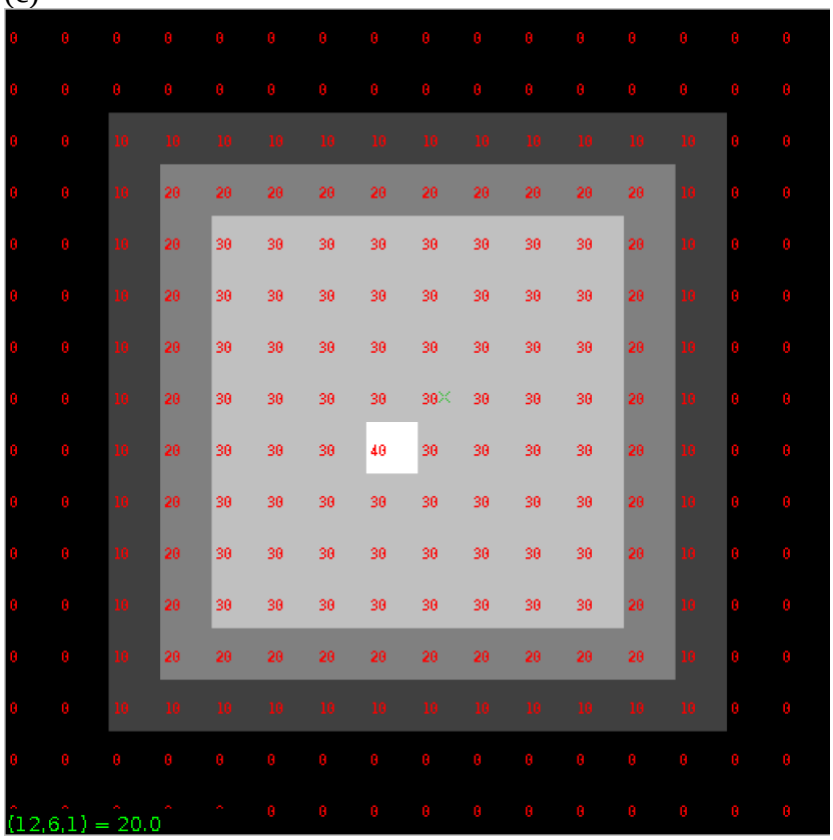
(a)



(b)



(c)



Using Scripts: Note, the scripts testim1, testim2, testim3, and ptest are editable text files. You may make copies of them and modify them to match your needs. If you copy a script file you may need to set the execution permision in the file desctiption so that the system will recognize it as a file that can be executed. The command to do this is "chmod +x". For example

```
cp testim1 testim1cp
cp testim2 testim2cp
cp testim3 testim3cp

chmod  +x testim1
chmod  +x testim2
chmod  +x testim3
```

set the execute permission for the three files

Prepare images for printing or for inclusion in your lab report

```
vpix -neg im1.vx | vcapt c="input image im1.vx"
of=p1.vx

vpix -neg im1.mx | vcapt c="vtemp processed image"
of=p2.vx
```

This command inverts the image so that you would use less "ink" if you wanted to print it.

View my results, as below:

Convert VisionX images into png, jpeg or gif files (for importing into LibreOffice Write and Draw programs for your report)

these commands will create two new image files "p1.gif" and "p2.gif" If you click on the Refresh button these files will become visible in vview. They may be displayed with either the vdview or "Tab view" viewers.

```
.. (Up Directory)
im1.mx        VXB   256x56x1    20180919
im1.vx        VXB   256x56x0    20050711
im2.vx        VXB   256x50x0    20050711
im3.vx        VXB   256x256x0   20050711
p1.gif        GIF   256x80x0    20180919
p1.vx         VXB   256x80x0    20180919
p2.vx         VXB   256x80x0    20180919
ptest         TEXT  NA          20180906
ptestpy       TEXT  NA          20180906
shuttle.vx    VXB   256x256x0   20050711
testim1       TEXT  NA          20180906
testim1.vx    VXB   16x16x0     20180919
testim1cp     TEXT  NA          20180919
testim2       TEXT  NA          20180906
testim2cp     TEXT  NA          20180919
testim3       TEXT  NA          20180906
testim3cp     TEXT  NA          20180919
tmp1.52110    VXB   16x16x0     20180919
tmp2.52110    VXB   16x16x0     20180919
tst.a         TEXT  NA          20100909
```

1. n general you should use the gif format for gray images and png or jpeg for color images. To generate png or jpeg images replace gif with png or jpeg in the above commands.

Command	Action
vcc	compile a c program with VisionX include files and library
vpix	perform point operations on an image
vcapt	add a caption to the bottom of an image
vxport	change image format from VisionX format

Small Image Manipulation

you develop is first tested on very small images that you can check the result. the following VisionX commands are used to create and display small test images:

Examine the ASCII "image" tst.a

```
less tst.a
```

```
0 0 0 0 0 0 0
0 1 1 0 0 0 0
0 0 1 0 0 0 0
0 0 0 0 3 3 0
0 0 0 3 3 3 0
0 0 0 0 0 0 8
```

The file "test.a" is also shown in vview as a TEXT file. By double clicking on this name you can view its contents. You can also edit the file in vview and save the changes by clicking on the "Save" button.

Generate a VisionX image

```
vrawtovx -t tst.a of=tst.vx
```

Display the VisionX image

```
vppr tst.vx
```

	0	1	2	3	4	5	6
5	0	0	0	0	0	0	0
4	0	1	1	0	0	0	0
3	0	0	1	0	0	0	0
2	0	0	0	0	3	3	0
1	0	0	0	3	3	3	0
0	0	0	0	0	0	0	8

Test the program vtemp

```
vtemp tst.vx | vppr
```

	0	1	2	3	4	5	6
5	0	1	1	0	0	0	0
4	1	1	1	1	0	0	0
3	0	1	1	1	3	3	0
2	0	0	1	3	3	3	3
1	0	0	3	3	3	3	8
0	0	0	0	3	3	8	8

Use scripts as an alternative approach. Create a testimge by executing script testim2 and then use ptest to test vtemp with the image imtest2.vx.

review the script testim2

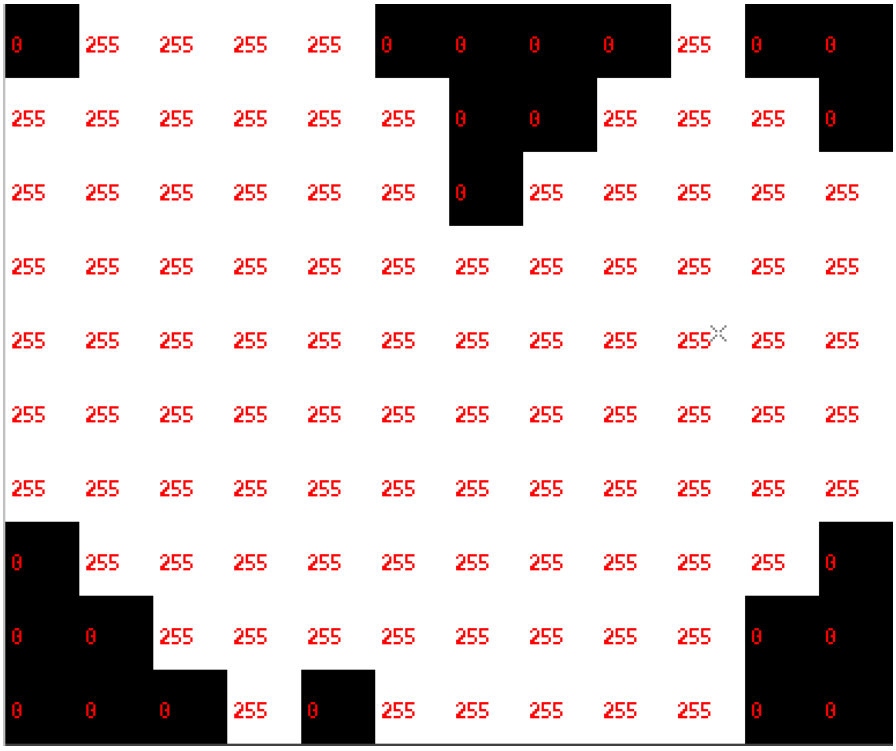
```
less testim2
```

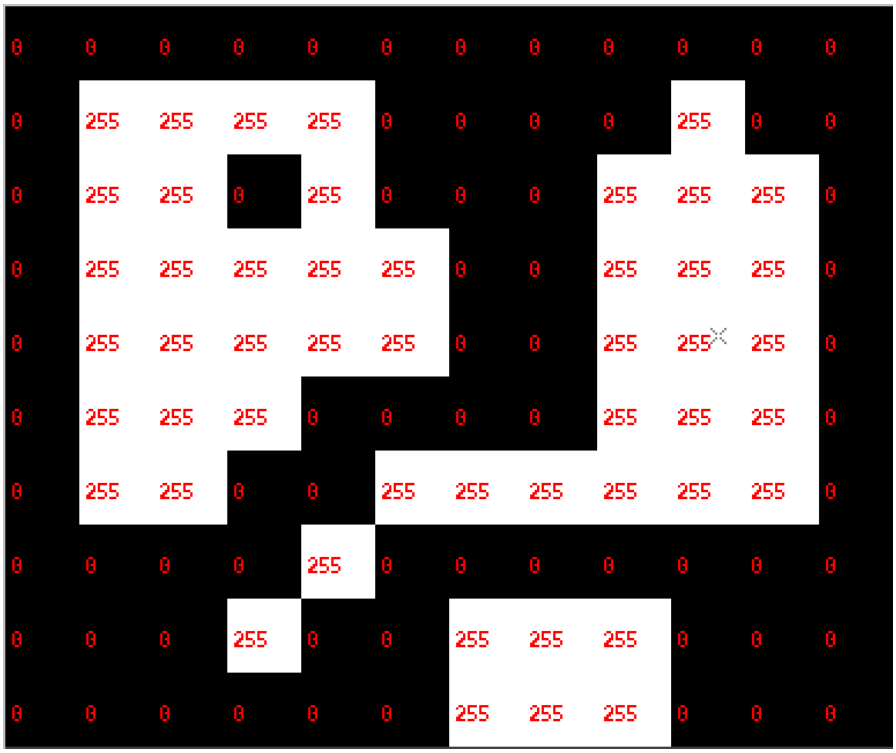
```
#!/bin/sh
#
# Example sh script to generate a binary test image
# The test image size and contents may be edited
# then execute this script to create testim2.vx
#
# script to generate the test image testim1.vx
vrawtovx -t << EOF | vfix -a of=testim2.vx
0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 0 0 0 0 1 0 0
0 1 1 0 1 0 0 0 1 1 1 0
0 1 1 1 1 1 0 0 1 1 1 0
0 1 1 1 1 1 0 0 1 1 1 0
0 1 1 1 0 0 0 0 1 1 1 0
0 1 1 0 0 1 1 1 1 1 1 0
0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 1 0 0 1 1 1 0 0 0
0 0 0 0 0 0 1 1 1 0 0 0
EOF
```

execute it to create the image testim2.vx

```
testim2
so we got
testim2.vx

ptest testim2.vx  vtemp
```





vvpr testim2.vx

	0	1	2	3	4	5	6	7	8	9	10	11
9	0	0	0	0	0	0	0	0	0	0	0	0
8	0	255	255	255	255	0	0	0	0	255	0	0
7	0	255	255	0	255	0	0	0	255	255	255	0
6	0	255	255	255	255	255	0	0	255	255	255	0
5	0	255	255	255	255	255	0	0	255	255	255	0
4	0	255	255	255	0	0	0	0	255	255	255	0
3	0	255	255	0	0	255	255	255	255	255	255	0
2	0	0	0	0	255	0	0	0	0	0	0	0
1	0	0	0	255	0	0	255	255	255	0	0	0
0	0	0	0	0	0	0	255	255	255	0	0	0

```
vtemp testim2.vx | vppr
```

	0	1	2	3	4	5	6	7	8	9	10	11
9	0	255	255	255	255	0	0	0	0	255	0	0
8	255	255	255	255	255	255	0	0	255	255	255	0
7	255	255	255	255	255	255	0	255	255	255	255	255
6	255	255	255	255	255	255	255	255	255	255	255	255
5	255	255	255	255	255	255	255	255	255	255	255	255
4	255	255	255	255	255	255	255	255	255	255	255	255
3	255	255	255	255	255	255	255	255	255	255	255	255
2	0	255	255	255	255	255	255	255	255	255	255	0
1	0	0	255	255	255	255	255	255	255	255	0	0
0	0	0	0	255	0	255	255	255	255	255	0	0

Make a new test image of your own design to test vtemp. You can do this with either the direct method or by using scripts. While the direct method may be simpler, the script method is superior for efficient program development but requires some understanding of the script language. It is suggested that you attempt both.

I Modify my own copy of the ASCII image using the vim text editor

```
cp tst.a mine.a
vim mine.a

and I revise my mine.a
as below:
```

0	0	0	0	0	0	0
0	1	1	0	0	0	0
0	0	1	0	0	0	0
0	0	0	0	3	3	0
0	0	0	3	3	3	0
0	0	0	0	0	0	8

0	0	0	0	1	0	0
0	1	0	0	0	0	0
0	4	0	0	5	0	0
0	0	0	1	3	3	0
0	0	1	2	3	2	0
0	0	2	0	1	0	4

Test the program vtemp with the new image

```
vrawtovx -t mine.a of=mine.vx
```

```
vppr mine.vx
```

	0	1	2	3	4	5	6
5	0	0	0	0	1	0	0
4	0	1	0	0	0	0	0
3	0	4	0	0	5	0	0
2	0	0	0	1	3	3	0
1	0	0	1	2	3	2	0
0	0	0	2	0	1	0	4

```
vtemp mine.vx | vppr
```

	0	1	2	3	4	5	6
5	0	1	0	1	1	1	0
4	1	4	1	0	5	0	0
3	4	4	4	5	5	5	0
2	0	4	1	3	5	3	3
1	0	1	2	3	3	3	4
0	0	2	2	2	3	4	4