Advanced Programming Term Project

Text-based Image Operations Using Different Programming Paradigms

COLLABORATORS

	TITLE : Advanced Programming Term Project		*REFERENCE* :
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	040020365. Mehmet CAMBAZ	12.05.2006	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

List of Figures

1 Project Description

In this project, we are supposed to implement programs that can do 10 different operations on a picture with 4 different programming languages.

The programming languages are:

- C (Procedural)
- Java (Object-Oriented)
- Python (Scripting)
- Haskell (Functional)

The operations to be implemented are:

1. Flip a picture horizontally
2. Flip a picture vertically
3. Rotate a picture
4. Rotate a picture 90 degrees anticlockwise
5. Rotate a picture 90 degrees clockwise
6. Invert the color of a picture
7. Scale a picture by a factor
8. Crop a region in a picture
9. Find the difference of two pictures
10. Detect the edges in a picture

Figure 1: Sample picture

2 Environments

2.1 Development Environment

C The C program is developed with Dev-C++ 4.9.9.2 IDE and compiled with gcc compiler. It can be compiled as: "gcc 040020365_c.c -o 040020365_c" at command line.

Java The Java program is developed with Eclipse 3.1 IDE and compiled with javac compiler. Also it is controlled with netBeans 5.0 IDE, if put in a package it can be compiled properly. It can be compiled as: "javac main.java" at command line.

Python The Python program is developed with Eclipse 3.1 IDE at Pydev perspective which is a eclipse plugin. Because it is a scripting language, there is no need to compile the main code, there is only need to compile the class. The class is automatically compiled when the main.py is executed. It can be executed as: "main.py" at command line or double clicking the "main.py" file.

Haskell The Haskell program is developed with Notepad. It can be compiled as: "ghc --make Main.hs" at command line or double clicking the "Main.hs" file.

2.2 Execution Environment

C To execute the C program at Windows XP: Dev-C++ 4.9.9.2 must be installed, at Dev-C++'s "bin" folder, which have "gcc.exe" must be added to environment variables to be used as "gcc" at command line. After compiled, as expressed above, the program can be executed as entering "040020365_c" at command line.

Java To execute the Java program at Windows XP: The Java Development Kit 1.5.0.06 (jdk-1_5_0_06-windows-i586-p.exe), The Java Runtime Environment 1.5.0.06 (jre-1_5_0_06-windows-i586-p.exe) -which can be get from http://sun.java.com- must have been installed. Also to be able to use "javac" and "java" at command line, Java Development Kit's "bin" folder must be added to environment variables. After compiling the program as expressed above, the Java program can be executed as "java main" at command line. If wanted, the folder can be added to a Eclipse java project and can be run as a java application which requires Eclipse 3.1 to be installed.

Python To execute the Python program at Windows XP: Python 2.4.2 Software Development Kit for Windows must have been installed. To execute the Python program entering "main.py" at command line will be enough. Also if the "main.py" is double clicked the program will be automatically executed.

Haskell To execute the Haskell program at Windows XP: Glasgow Haskell Compiler 6.4.1 must have been installed. To be able to use "ghc" at command line, GHC's "bin" folder must be added to environment variables. After compiling the program as expressed above, the Haskell program can be executed as "main" at command line. Also it can be executed with double-clicking the "Main.hs" file.

3 Problem Discussions

3.1 Data Structures

3.1.1 Data Structure in C

A picture is a struct which consists a two-dimensional character array named "pixels".

```
typedef struct pic_s
{
    char pixels[100][100];
} pic_t;
```

3.1.2 Data Structure in Java

A picture is a public class has private members rows (number of rows in the picture), cols (number of columns in the picture) and public two-dimensional character array pixels whose size is set at creation of the picture. The picture class has two constructors, has a copy constructor, methods for setting and getting private members rows and cols and a print method that prints the picture's pixels to user.

```
/**
* picture class
* represents a picture
* contains number of rows (height)
* number of columns (width)
* picture content (pixels)
* */
public class picture
{
  public char pixels[][];   //picture content
  private int rows;         //number of rows (height)
  private int cols;         //number of columns (width)

  //constructor
  picture()
  {
    //means nothing assigned
    cols = 0;
    rows = 0;
    pixels = new char[100][100];  //default picture size is 100*100
  }

  //given height and width constructor
  picture(int row, int col)
  {
    cols = col;
    rows = row;
    pixels = new char[row][col];
  }

  //copy constructor
  picture(picture right)
  {
    cols = right.cols;
    rows = right.rows;

    for(int i=0; i<right.getRows(); i++)
      pixels[i] = right.pixels[i];
  }

  public int getCols()
  {
    return cols;
  }

  public int getRows()
  {
    return rows;
  }

  public void setCols(int col)
  {
    cols = col;
  }
```

```
  public void setRows(int row)
  {
    rows = row;
  }

  /**
  * prints the picture content to user
  */
  public void print()
  {
    for(int i=0; i<getRows(); i++)
    {
      for(int j=0; j<getCols(); j++)
        System.out.print(pixels[i][j]);

      System.out.print("\n");
    }
  }
}
```

3.1.3 Data Structure in Python

A picture is a class like in Java but every member is public because of Python's class characteristics. Also the picture content is list of list of characters.

```
################################
# picture class                #
# used to represent pictures   #
################################
class picture:
    rows = 0        #number of rows in the picture (height)
    cols = 0        #number of columns in the picture (width)
    pixels = []     #picture pixel content

    #constructor
    def __init__(self, row, col, filename=""):
        self.rows = row
        self.cols = col
        self.pixels = []

        if filename != "":
            f = open(filename, 'r')

            for i in range(row):
                column = []
                l = f.readline()
                for j in range(col):
                    column.append(l[j])
                self.pixels.append(column)

            f.close()

    #prints picture content
    def printPixels(self):
        for i in range(self.rows):
            for j in range(self.cols):
                print self.pixels[i][j],
            print '\n',
```

3.1.4 Data Structure in Haskell

A picture is list of list of characters.

```
type Picture = [[Char]]
```

3.2 Problems and solutions

3.2.1 Flipping the picture horizontally

The given picture will be flipped horizontally. It means that the picture will be flipped like there is a mirror at the bottom of it. This operation can be made by changing the bottom line pixels of the picture with upper line pixels.

3.2.1.1 Solution in C

```
void flipHorizontal(pic_t picInMemory, int lastRowIndex, int lastColIndex)
{
  pic_t flippedPic;
  int j, i;

  //flip horizontally
  for(j=lastColIndex; j>=0; j--)
  {
    for(i=0; i<lastRowIndex+1; i++)
      flippedPic.pixels[lastRowIndex-1-i][j] = picInMemory.pixels[i][j];    //critical line
  }
  //printing out
}
```

3.2.1.2 Solution in Java

```
private static void flipHorizontal(picture picInMemory)
{
  picture flippedPic = new picture(picInMemory.getRows(), picInMemory.getCols());
  int j, i;
  int lastColIndex = picInMemory.getCols()-1;
  int lastRowIndex = picInMemory.getRows()-1;

  //flip horizontally
  for(j=lastColIndex; j>=0; j--)
  {
    for(i=0; i<lastRowIndex+1; i++)
      flippedPic.pixels[lastRowIndex-i][j] = picInMemory.pixels[i][j];    //critical line
  }

  flippedPic.print();
}
```

3.2.1.3 Solution in Python

```
def flipHorizontal(pic):

  flippedPic = picture(pic.rows, pic.cols)

  for i in range(pic.rows):
```

```
    column = []
    for j in range(pic.cols):
      column.append(pic.pixels[pic.rows-1-i][j])        #critical line
    flippedPic.pixels.append(column)
```

3.2.1.4 Solution in Haskell

```
flipHorizontal :: Picture -> Picture
flipHorizontal = reverse
```

3.2.1.5 Output picture

```
..#...#...
..#...#...
..#.#.#...
..##.##...
..#...#...
```

3.2.2 Flipping the picture vertically

The given picture will be flipped vertically. It means that the picture will be flipped like there is a mirror at the right of it. This operation can be made by changing the right vertical line pixels of the picture with left vertical line pixels.

3.2.2.1 Solution in C

Critical line will be:

```
flippedPic.pixels[i][lastColIndex-j] = picInMemory.pixels[i][j];
```

3.2.2.2 Solution in Java

Critical line will be:

```
flippedPic.pixels[i][lastColIndex-j] = picInMemory.pixels[i][j];
```

3.2.2.3 Solution in Python

Critical line will be:

```
column.append(picInMemory.pixels[i][pic.cols-1-j])
```

3.2.2.4 Solution in Haskell

```
flipVertical :: Picture -> Picture
flipVertical = map reverse
```

3.2.2.5 Output picture

```
...#...#..
...##.##..
...#.#.#..
...#...#..
...#...#..
```

3.2.3 Rotating the picture

The given picture will be flipped both vertically and horizontally.

3.2.3.1 Solution in C

```
pic_t rotatedPic, tempPic;
int j, i;

//flip horizontally
for(j=lastColIndex; j>=0; j--)
{
   for(i=0; i<lastRowIndex+1; i++)
       tempPic.pixels[lastRowIndex-1-i][j] = picInMemory.pixels[i][j];
}

//flip vertically
for(j=lastColIndex; j>=0; j--)
{
   for(i=0; i<lastRowIndex+1; i++)
        rotatedPic.pixels[i][lastColIndex-j] = tempPic.pixels[i][j];
}
```

3.2.3.2 Solution in Java

```
picture rotatedPic = new picture(picInMemory.getRows(), picInMemory.getCols());
picture tempPic = new picture(picInMemory.getRows(), picInMemory.getCols());

int j, i;
int lastColIndex = picInMemory.getCols()-1;
int lastRowIndex = picInMemory.getRows()-1;

//flip horizontally
for(j=lastColIndex; j>=0; j--)
{
  for(i=0; i<lastRowIndex+1; i++)
    tempPic.pixels[lastRowIndex-i][j] = picInMemory.pixels[i][j];
}

//flip vertically
for(j=lastColIndex; j>=0; j--)
{
  for(i=0; i<lastRowIndex+1; i++)
    rotatedPic.pixels[i][lastColIndex-j] = tempPic.pixels[i][j];
}
```

3.2.3.3 Solution in Python

```
rotatedPic = picture(pic.rows, pic.cols)
temppixels = []

for i in range(pic.rows):
    column1 = []
    for j in range(pic.cols):
        column1.append(pic.pixels[pic.rows-1-i][j])
    temppixels.append(column1)

for i in range(pic.rows):
    column2 = []
    for j in range(pic.cols):
        column2.append(temppixels[i][pic.cols-1-j])
    rotatedPic.pixels.append(column2)
```

3.2.3.4 Solution in Haskell

```
rotate :: Picture -> Picture
rotate = flipHorizontal . flipVertical
```

3.2.3.5 Output picture

```
...#...#..
...#...#..
...#.#.#..
...##.##..
...#...#..
```

3.2.4 Rotating the picture 90 degrees anti-clockwise

The given picture will be rotated 90 degrees anti-clockwise. It means that the picture will be will be rotated to left by 90 degrees. It can be done by taking the leftmost vertical line pixels to bottom horizontal line pixels.

3.2.4.1 Solution in C

Critical line will be:

```
rotatedPic.pixels[lastColIndex-j][i] = picInMemory.pixels[i][j];
```

3.2.4.2 Solution in Java

Critical line will be:

```
rotatedPic.pixels[lastColIndex-j][i] = picInMemory.pixels[i][j];
```

3.2.4.3 Solution in Python

Critical line will be:

```
column.append(pic.pixels[j][pic.cols-1-i])
```

3.2.4.4 Solution in Haskell

```
rotate90ac :: Picture -> Picture
rotate90ac =  transpose . reverse
```

3.2.4.5 Output picture

```
.....
.....
.....
#####
.#...
..#..
.#...
#####
.....
.....
```

3.2.5 Rotating the picture 90 degrees clockwise

The given picture will be rotated 90 degrees clockwise. It means that the picture will be will be rotated to right by 90 degrees. It can be done by taking the leftmost vertical line pixels to upper horizontal line pixels.

3.2.5.1 Solution in C

Critical line will be:

```
rotatedPic.pixels[j][lastRowIndex-i] = picInMemory.pixels[i][j];
```

3.2.5.2 Solution in Java

Critical line will be:

```
rotatedPic.pixels[j][lastRowIndex-i] = picInMemory.pixels[i][j];
```

3.2.5.3 Solution in Python

Critical line will be:

```
column.append(pic.pixels[pic.rows-1-j][i])
```

3.2.5.4 Solution in Haskell

```
rotate90 :: Picture -> Picture
rotate90 = rotate90ac . rotate
```

3.2.5.5 Output picture

```
.....
.....
#####
...#.
..#..
...#.
#####
.....
.....
.....
```

3.2.6 Inverting the picture's color

Inverting the picture's color means that if the picture's pixel is '.' then the pixel will be '#', if the picture's pixels is '#' then the pixel will be '.'.

3.2.6.1 Solution in C

```
pic_t invertedPic;
int j, i;

for(j=0; j<lastColIndex+1; j++)
{
  for(i=0; i<lastRowIndex+1; i++)
    invertedPic.pixels[i][j] = picInMemory.pixels[i][j] == '#' ? '.' : '#';
}
```

3.2.6.2 Solution in Java

```
for(j=0; j<lastColIndex+1; j++)
{
  for(i=0; i<lastRowIndex+1; i++)
    invertedPic.pixels[i][j] = picInMemory.pixels[i][j] == '#' ? '.' : '#';
}
```

3.2.6.3 Solution in Python

```
inverted = picture(pic.rows, pic.cols)

for i in range(pic.rows):
  column = []
  for j in range(pic.cols):
    if pic.pixels[i][j] == '#':
      column.append('.')
    else:
      column.append('#')

  inverted.pixels.append(column)
```

3.2.6.4 Solution in Haskell

```
-- Function that inverts the color of the given picture
invertColor :: Picture -> Picture
invertColor = map (map invert)

-- Function that inverts the color of a pixel
invert :: Char -> Char
invert ch = if ch == '.' then '#' else '.'
```

3.2.6.5 Output picture

```
##.###.###
##..#..###
##.#.#.###
##.###.###
##.###.###
```

3.2.7 Scaling the picture with given factor

The given picture will be scaled by the given factor. It means that all the pixels will be regenerated in a scale*scale area.

3.2.7.1 Solution in C

```
for(i=0; i<(lastRowIndex+1)*factor; i+=factor)
{
  for(j=0; j<(lastColIndex+1)*factor; j+=factor)
  {
    for(k=0; k<factor; k++)
    {
      for(m=0; m<factor; m++)
      {
        scaledPic.pixels[i+k][j+m] = picInMemory.pixels[i/factor][j/factor];
      }
    }
  }
}
```

3.2.7.2 Solution in Java

```
picture scaledPic = new picture(picInMemory.getRows()*factor, picInMemory.getCols()*factor) ↩
    ;

int lastColIndex = picInMemory.getCols()-1;
int lastRowIndex = picInMemory.getRows()-1;
int j, i, k, m;

for(i=0; i<(lastRowIndex+1)*factor; i+=factor)
{
  for(j=0; j<(lastColIndex+1)*factor; j+=factor)
  {
    for(k=0; k<factor; k++)
    {
      for(m=0; m<factor; m++)
      {
        scaledPic.pixels[i+k][j+m] = picInMemory.pixels[i/factor][j/factor];
```

```
      }
    }
  }
}
```

3.2.7.3 Solution in Python

```
def scaleByFactor(pic, factor):

    row = pic.rows*factor
    col = pic.cols*factor
    scaledPic = picture(row, col)

    for i in range(pic.rows):
        temppixels = []
        for j in range(pic.cols):
            for k in range(factor):
                temppixels.append(pic.pixels[i][j])
        for m in range(factor):
            scaledPic.pixels.append(temppixels)
```

3.2.7.4 Output picture (scaled by factor=2)

```
....##......##......
....##......##......
....####..####......
....####..####......
....##..##..##......
....##..##..##......
....##......##......
....##......##......
....##......##......
....##......##......
```

3.2.8 Cropping a region from the picture

With given values: cropping x-start, y-start point and crop height and width; the wanted region will be cropped. This means the content pixels of x-start to x-start+height and y-start to y-start+width area is the cropping region.

3.2.8.1 Solution in C

x is x-start point, y is y-start point, w is width, h is height for cropping:

```
for(i=x; i<x+h; i++)
{
  for(j=y; j<y+w; j++)
    croppedPic.pixels[i-x][j-y] = picInMemory.pixels[i][j];
}
```

3.2.8.2 Solution in Java

x is x-start point, y is y-start point for cropping:

```
for(i=x; i<x+height; i++)
{
  for(j=y; j<y+width; j++)
    croppedPic.pixels[i-x][j-y] = picInMemory.pixels[i][j];
}
```

3.2.8.3 Solution in Python

x is x-start point, y is y-start point, w is width, h is height for cropping:

```
croppedPic = picture(h, w)

for i in range(h):
  tempcols = []
  for j in range(w):
    tempcols.append(pic.pixels[x+i][y+j])
  croppedPic.pixels.append(tempcols)
```

3.2.8.4 Output picture (x-start=0 y-start=0 w=5 h=5)

```
..#..
..##.
..#.#
..#..
..#..
```

3.2.9 Finding the differences of two pictures

If the pictures' pixel is same then the difference picture's that pixel will be '.' otherwise it will be '#'.

3.2.9.1 Solution in C

picInMemory1 is the first picture, picInMemory2 is the second picture. lastRowIndex1, lastColIndex1 are first picture's size indexes the others are second picture's. MAX is defined as "#define MAX(x, y) x>=y ? x : y"

```
maxRow = MAX(lastRowIndex1, lastRowIndex2);
maxCol = MAX(lastColIndex1, lastColIndex2);

for(i=0; i<maxRow; i++)
{
  for(j=0; j<maxCol+1; j++)
    differencePic.pixels[i][j] = picInMemory1.pixels[i][j]==picInMemory2.pixels[i][j] ? '.' ↩
        : '#';
}
```

3.2.9.2 Solution in Java

```
/**
* returns the bigger or equal to of given two numbers
*/
private static int max(int n1, int n2)
{
  if(n1 >= n2)
    return n1;
```

```
  else
    return n2;
}

private static void findDifferences(picture picInMemory, picture picInMemory2)
{
  int maxRow = max(picInMemory.getRows()-1, picInMemory2.getRows()-1);
  int maxCol = max(picInMemory.getCols()-1, picInMemory2.getCols()-1);

  picture differencePic = new picture(maxRow+1, maxCol+1);
  int i, j;

  for(i=0; i<maxRow+1; i++)
  {
    for(j=0; j<maxCol+1; j++)
    differencePic.pixels[i][j] = picInMemory.pixels[i][j]==picInMemory2.pixels[i][j] ? '.'
        : '#';
  }
}
```

3.2.9.3 Solution in Python

```
#returns the smaller or equal to of given two numbers
def min(n1, n2):

    if n1 <= n2:
        return n1
    else:
        return n2

def findDifferences(pic, pic2):

  #if pictures are not the same size, the minimum size will be the size of difference
     picture
  minRow = min(pic.rows, pic2.rows)
  minCol = min(pic.cols, pic2.cols)

  differencePic = picture(minRow, minCol)

  for i in range(minRow):
    column = []
    for j in range(minCol):
      if pic.pixels[i][j] == pic2.pixels[i][j]:
        column.append('.')
      else:
        column.append('#')
    differencePic.pixels.append(column)

  differencePic.printPixels()
```

3.2.9.4 Output picture (picture 1 is 1.txt, picture 2 is 2.txt)

```
..#.##.####
..#..##...#
..###.#...#
..##..#...#
..#.##.#..#
###########
```

3.2.10 Detecting the edges of a picture

This operation finds the edges of the picture. An edge pixel must be black ('#') and must not be surrounded with black pixels. If a black pixel is all surrounded with black pixels, it is not on the edge. If a black pixel is on a border line of the picture then it is on the edge for sure. A white pixel is never considered on a edge, the edges will be shown as black pixels at the output picture.

3.2.10.1 Solution in C

```
for(i=0; i<lastRowIndex; i++)
{
  for(j=0; j<lastColIndex+1; j++)
  {
    if(picInMemory.pixels[i][j] == '#')
    {
      //if the black pixel is on the edges of the picture, then it is a part of edge
      if(i==0 || j==0 || i==lastRowIndex || j==lastColIndex)
        edgePic.pixels[i][j] = '#';
      else
      {
        /* if all the pixels around a black pixel is black, then
           it is not a edge, otherwise it is considered edge  */
        if(picInMemory.pixels[i+1][j] == '#'
             && picInMemory.pixels[i-1][j] == '#'
             && picInMemory.pixels[i][j+1] == '#'
             && picInMemory.pixels[i][j-1] == '#'
             && picInMemory.pixels[i-1][j-1] == '#'
             && picInMemory.pixels[i+1][j+1] == '#')
          edgePic.pixels[i][j] = '.';
        else
          edgePic.pixels[i][j] = '#';
      }
    }
    else
      edgePic.pixels[i][j] = '.';
  } // j for end
} //i for end
```

3.2.10.2 Solution in Java

```
picture edgePic = new picture(picInMemory.getRows(), picInMemory.getCols());

int j, i;
int lastColIndex = picInMemory.getCols()-1;
int lastRowIndex = picInMemory.getRows()-1;

for(i=0; i<lastRowIndex+1; i++)
{
  for(j=0; j<lastColIndex+1; j++)
  {
    if(picInMemory.pixels[i][j] == '#')
    {
      //if the black pixel is on the edges of the picture, then it is a part of edge
      if(i==0 || j==0 || i==lastRowIndex || j==lastColIndex)
        edgePic.pixels[i][j] = '#';
      else
      {
        /* if all the pixels around a black pixel is black, then
           it is not a edge, otherwise it is considered edge  */
        if(picInMemory.pixels[i+1][j] == '#'
```

```
            && picInMemory.pixels[i-1][j] == '#'
            && picInMemory.pixels[i][j+1] == '#'
            && picInMemory.pixels[i][j-1] == '#'
            && picInMemory.pixels[i-1][j-1] == '#'
            && picInMemory.pixels[i+1][j+1] == '#')
          edgePic.pixels[i][j] = '.';
        else
          edgePic.pixels[i][j] = '#';
      }
    }
    else
     edgePic.pixels[i][j] = '.';

  } // j for end
} //i for end
```

3.2.10.3 Solution in Python

```
edgePic = picture(pic.rows, pic.cols)

for i in range(pic.rows):
  column = []
  for j in range(pic.cols):
    if pic.pixels[i][j] == '#':
      #if the black pixel is on the edges of the picture, then it is a part of edge
      if i==0 or j==0 or i==pic.rows-1 or j==pic.cols-1:
        column.append('#')
        # if all the pixels around a black pixel is black, then
        #   it is not a edge, otherwise it is considered edge
      else:
        if pic.pixels[i+1][j] == '#' and pic.pixels[i-1][j] == '#' and pic.pixels[i][j+1]  ↩
            == '#' and pic.pixels[i][j-1] == '#' and pic.pixels[i-1][j-1] == '#' and pic. ↩
            pixels[i+1][j+1] == '#':
          column.append('.')
        else:
          column.append('#')
    else:
      column.append('.')

edgePic.pixels.append(column)
```

3.2.10.4 Output picture (picture is 3.txt)

```
##########
#........#
#........#
#........#
##########
```