Programming Problem
for the
1998 Comprehensive Exam

Out: Monday 1/12/98 at 9:00 am
Due: Friday 1/16/98 at 5:00 pm

Department of Computer Science
Brown University
Providence, RI 02912

1.0 The Problem

Your job this week is to write a program that can solve a jigsaw puzzle. There are many programs available on the net or commercially that create jigsaw puzzles and let the user solve them. However, what you are going to do is just the opposite. We have created a set of puzzles already and want you to write a program that will put these puzzles together.

Each puzzle will be presented in a data file whose format is defined below. This file will define the size of the puzzle, the number of pieces in the puzzle, and then define each of the pieces essentially as a bitmap (although in an encoded form). The pieces should fit together precisely and without holes or overlap. Your task is to write a program to read the puzzle file and determine where each of the pieces fits, essentially reconstructing the original bitmap in the process. Your program should create the solution, display it to the user, and write out a solution file in the appropriate format.

2.0 Input Format

The input to your program consists of a single file for each puzzle. This file is an ASCII file consists of lines that contain a command followed by a sequence of arguments. The initial lines of the file provide general information about the puzzle. These include:

Command	Arguments	Description
DIM	<width> <height>	Provides the width and height of the overall bitmap in pixels. Both values are integers.
COUNT	<number>	Provides the number of pieces in the puzzle

These header lines are followed by descriptions of each of the pieces. Each piece consists of a header followed by a number of lines that define its bitmap. The general format is:

Command	Arguments	Description
PIECE	<piece number>	Start of a new piece, defines which piece. (The pieces are numbered consecutively so this value can be ignored.)
DIM	<width> <height>	The width and height of the bitmap for this piece. This should be the minimum space needed to contain the bitmap in its current form.
ROW	<number> <values...>	Defines the contents of one row of the bitmap.

The bitmap data is given in a run-length encoded form. Values here either consist of a negative number or two positive numbers. A negative number, say -5, indicates that the next 5 pixels in the row are not part of the piece. Two positive numbers, say 17 followed by 255, indicates that the next 17 pixels in the row have the color 255. Colors are encoded using 24 bit notation with the red value being in the low-order 8 bits, green in the middle 8 bits, and blue in the high-order 8 bits.

The file ends with the end of the last piece description.

Your program should both draw the resultant image (with an appropriate, albeit simple user interface) and create a solution file. The solution file should contain the following header lines:

Command	Arguments	Description
PUZZLE	<filename>	Provide the full path name of the file containing the puzzle that is being solved.
DIM	<width> <height>	Provide the size of the solved puzzle
COUNT	<number>	Provide the number of pieces

These are followed by one line per piece that defines the location of each piece. This line has the form:

Command	Arguments	Description
PIECE	<number> <xpos> <ypos> <orientation>	Defines the location and orientation of the piece in the solution.

The number here corresponds to the piece number in the original puzzle. The x and y positions define the location of the upper left hand corner of the bitmap of the piece. The final argument specifies how the piece is oriented. This can be 0, 1, 2, or 3 to indicate an orientation of 0°, 90°, 180°, and 270° respectively (where rotations are done in a counter-clockwise direction).

A number of sample puzzles are available in the directory /u/spr/comps/comps98/puzzles. New puzzles will be made upon request (provide an xwd bitmap and an estimate of the number of pieces desired). Your program may be tested (for the final grade) on these or other puzzles.

3.0 The Task

You should construct a program along the lines described above. Part of the problem is defining an appropriate user interface. It is up to you do define what an appropriate user interface is. You can use any available system (i.e. Sun, Mac, PC) that you normally have access to, and any programming language that you deem appropriate. You should get the basic program working before implementing any extensions. You should hand in a complete and commented program listing, brief documentation on how to use the program, and a description of the algorithm you are using to put the puzzle together. You should also electronically submit a working version of the program by copying all necessary files to the directory /u/spr/comps98/XXX where XXX is your login id. (The program will be run from that directory.) Alternatively, you can submit a Mac or PC floppy containing your program.

All work must be completed by the due date and time. All of your work on this problem must be yours and yours alone. Any discussion should be in the news group *brown.cs.comps* or via email to spr and should be limited to questions and not suggestions or design hints. Grading will be based on 1) the program working, 2) the program structure and algorithms, 3) the speed of the implementation, 4) the quality of the user interface, and 5) usefulness of the documentation, both internal and external (roughly in that order). Any questions should go to spr.