Sample C Code

I wanted to give out some sample C code to serve as a benchmark for what I consider to be good decomposition, documentation, and style. This code is taken from an old semi-functional Tetris project I had lying around. Here are the things the code hopefully gets right:

- I tried to give all types, field names, variables, parameters and functions meaningful names. I do use the meaningless variable name "i" in places, but always for its usual role— a loop counter for some range 0..(n-1).
- The larger sub-problems like ReadPiece and Synthesize are decomposed into smaller routines.
- The procedures have comments explaining what'g going on. Where the code is dense, there is either an implementation or inline comment to help explain. See ComputeSkirt, EqualPieces, or RotatePiece.
- There's an overview comment which describes what the package does and gives the basic definitions of its types.
- I tried to use consistent schemes for: whitespace, capitalization, and indentation.

```
/* piece.h                    Nick Parlante, Tetris Project, 10-12-93  */
/*
Overview:
The Piece type implements a tetris piece.
A piece is defined by the 4 points which make up its body with
(0,0) defining the lower left corner in the Piece coordinate system.
So for example, the body of the square tetris piece has the body:
(0,0), (1,0), (0,1), (1,1)

A piece also has a height, width, and "skirt."
The skirt of a piece is the subset of its body points
which are at the bottom of the piece.
The skirt of the square piece is:
(0,0), (1,0)

Piece is used by the Board ADT which manages
the board as a whole.
Having the skirt pre-computed in the Piece makes it
easier to compute how the piece falls in the Board later.

Piece uses the Point and Vector types below
to implement of the body and skirt. For standard Tetris,
pieces contain at most 4 points, although all these
routines will work with bigger pieces if you change
the MAXPOINTS constant.
*/

#include <stdio.h>

#define MAXPOINTS 4
```

```
/* A Point stores one (x,y) pair. */
typedef struct point {
      int x;
      int y;
} Point;

/* A Vector stores 4 points. */
typedef struct vector {
      struct point points[MAXPOINTS];
      int length;
} Vector;

/* See the overview above. */
typedef struct piece {
      int height;
      int width;
      Vector body;
      Vector skirt;
} Piece;

/*
Read in the definition of a piece from a file.
Allocates and returns a pointer to the new piece.
The caller is owns the new pointer and is responsible
for deallocating it.
*/
Piece *ReadPiece(FILE *file);

/* Rotate a piece 90 degrees clockwise. */
void RotatePiece(Piece *piece);

/* Compare two pieces- return true if the same, false otherwise. */
int EqualPieces(Piece *p1, Piece *p2);
```

```
/* piece.c */
#include <piece.h>
#include <limits.h>

/*
Utility- fills the given vector by reading pairs of numbers out
of the file. A -1 is the end-of-vector marker in the file.
*/
static void ReadVector(FILE *file, Vector *vector) {
      int x,y;

      vector->length = 0;
      while (fscanf(file, "%d %d",&x, &y)) {
            if (x==-1) break; /* check for sentinal */

            if (vector->length >= MAXPOINTS) {
                  fprintf(stderr,"Vector too long!\n"); /* could err better-- */
                  exit(-1);                            /* IO is a bore.      */
            }
            vector->points[vector->length].x=x;
            vector->points[vector->length].y=y;
            vector->length++;
      }
}
```

```
/* Utility- given a piece with a body and a width, compute its skirt. */
static void ComputeSkirt(Piece *piece) {
      int i, x, minY;


      for (x=0; x<piece->width; x++) {

            /* Find the min y in the body for the current x. */
            minY = INT_MAX;
            for (i=0; i<piece->body.length; i++) {
                  if (piece->body.points[i].x==x) {
                        minY = MIN(minY, piece->body.points[i].y);
                  }
            }

            piece->skirt.points[x].x=x;
            piece->skirt.points[x].y=minY;
      }
      piece->skirt.length=piece->width;
}


/* Works by finding the max x and y values. */
static void ComputeHeightWidth(Piece *piece) {
      int i;

      piece->width=0;
      piece->height=0;
      for (i=0; i<piece->body.length; i++) {
            piece->width = MAX(piece->width, piece->body.points[i].x);
            piece->height = MAX(piece->height, piece->body.points[i].y);
      }
}


/*
Given a piece where the body is right, this fills out
all the fields which are computed from the body.
Used by ReadPiece and RotatePiece.
*/
static void Synthesize(Piece *piece) {
      ComputeHeightWidth(piece);
      ComputeSkirt(piece);
}


/*
Read in the definition of a piece from a file.
*/
Piece *ReadPiece(FILE *file) {
      Piece *piece;

      piece = (Piece*)malloc(sizeof(struct piece));
      ReadVector(file, &(piece->body));
      Synthesize(piece);
      return(piece);
}
```

```
/*
Rotate a piece 90 degrees clockwise.
Works by swapping x and y, and then reflecting the
result around the x axis.
I tried it on paper and it works!
At most four rotations are required to get
back to the original piece.
*/
void RotatePiece(Piece *piece) {
      int i;
      struct point pt;

      for (i=0; i<piece->body.length; i++) {
            pt = piece->body.points[i];
            piece->body.points[i].x = pt.y;
            piece->body.points[i].y = (piece->width - pt.x - 1);
      }

      Synthesize(piece);
}


/* Utility- returns true if the vector includes the given point. */
static int IncludesPoint(Vector *vector, struct point pt) {
      int i;

      for (i=0; i<vector->length; i++) {
            if ((vector->points[i].x == pt.x) &&
                  (vector->points[i].y == pt.y)) {
                  return(1);
            }
      }
      return(0);
}


/*
Returns true if two pieces are the same-
which means their bodies contain the same points.
Interestingly, this is not the same as having exactly the
same bit patterns, since the points may not be
in the same order in their bodies.
*/
int EqualPieces(Piece *p1, Piece *p2) {
      int i;

      if (p1->body.length != p2->body.length) return(0);

      /* check that all of p1's points are present in p2 */
      for (i=0; i<p1->body.length; i++) {
            if (!IncludesPoint(&(p2->body), p1->body.points[i])) {
                  return(0);  /* bail immediately upon a difference */
            }
      }

      /* if we get to this point, they're the same */
      return(1);
}
```