```
#include <stdio.h>
#include <stdlib.h>
#include "fig.h"
#include "shape.h"
#include <math.h>
#define PI M_PI
#define MAXBUFFER 1024
#define MAX         64
#define NEWLINE printf("\n")



/*************************************************************************
*************************************************************************
******************
Misc. Functions created by Nathan Christian Copyright (c) 2017
*************************************************************************
*************************************************************************
******************/

int shape_sscanf(char *stream, Shape *sp){

      char type[MAX];
      sscanf(stream, "%s", type);

      if (strcmp(type, "circle")==0){
                  sp->kind=type_circle;
                  circle_sscanf(stream, &sp->object.circle);
      }
      return 0;
                                                            //scans buffer
for circles.
}

void print_help(){

      NEWLINE;
      printf("To begin this program, type one of the commands below:");
      NEWLINE;
      printf("\t -h\t This prints a help message.");
      NEWLINE;
      printf("\t -c CXX CXX\t This draws a connecting line between two
circles.");
      NEWLINE;
      printf("Substitute CXX with an appropriate circle name.");
      NEWLINE;
      printf("Appropriate names are c00-c39");
      NEWLINE;
      printf("an example of a working command is ' draw -c c00 c01 '.");
      exit(0);
                                                            //Help message
}
```

```
Shape* connect_line(Shape *ss1, Shape *ss2){

      Shape *p;
      double dist, theta, x1, x2, y1, y2, r1, r2;

      x1=ss1->object.circle.center.x;
      x2=ss2->object.circle.center.x;
      y1=ss1->object.circle.center.y;
      y2=ss2->object.circle.center.y;
      r1=ss1->object.circle.radius;
      r2=ss2->object.circle.radius;
                                                      //actual values
converted to x and y variables to shorten width of commands.

      dist=sqrt(pow((x1-x2), 2)+pow((y1-y2), 2));
                                                    //find the distance for trig
functions below.

      if (dist<(r1+r2)){
                  printf("These two circles overlap and a connecting line
cannot be drawn\n");
                  exit(0);
                                                                        //if
circles overlap, prints a message and exits.
      }

      else if (ss1->kind != type_circle || ss2->kind != type_circle){
                  printf("A shape other than circle was read. Connect-line
only works for circles");
                  exit(0);
                                                                        //Error
message if circles are not read.
      }

      else {
                  p = malloc(sizeof(Shape)*2);
                  p->object.polyline.vertex=malloc(sizeof(Point)*2);
                                                //allocate memory for Shape
Pointer

                  p->kind=type_polyline;
                  theta=atan((fabs((y2-y1)/(x2-x1))));
                                                      //finds the angle used
for trig functions below

                  if (x1<x2&&y1>y2) {

      p->object.polyline.vertex->x=(x1+r1*cos(theta));
                              p->object.polyline.vertex->y=y1-
r1*sin(theta);
                              p->object.polyline.vertex++;
```

```
                              p->object.polyline.vertex->x=x2-
r2*cos(theta);

      p->object.polyline.vertex->y=y2+r2*sin(theta);
                  }
                  if (x1<x2&&y1<y2) {

      p->object.polyline.vertex->x=(x1+r1*cos(theta));

      p->object.polyline.vertex->y=y1+r1*sin(theta);
                              p->object.polyline.vertex++;
                              p->object.polyline.vertex->x=x2-
r2*cos(theta);
                              p->object.polyline.vertex->y=y2-
r2*sin(theta);
                  }
                  if (x1>x2&&y1<y2) {
                              p->object.polyline.vertex->x=(x1-
r1*cos(theta));

      p->object.polyline.vertex->y=y1+r1*sin(theta);
                              p->object.polyline.vertex++;

      p->object.polyline.vertex->x=x2+r2*cos(theta);
                              p->object.polyline.vertex->y=y2-
r2*sin(theta);
                  }
                  if (x1>x2&&y1>y2) {
                              p->object.polyline.vertex->x=(x1-
r1*cos(theta));
                              p->object.polyline.vertex->y=y1-
r1*sin(theta);
                              p->object.polyline.vertex++;

      p->object.polyline.vertex->x=x2+r2*cos(theta);

      p->object.polyline.vertex->y=y2+r2*sin(theta);
                                    //Trig functions to find xy coordinates
on the radius of the circle.
                  }
                  p->object.polyline.vertex--;
      }
      return p;
}
```