/* Write a C program called PizzaCalc to identify the most cost-effective pizza one can order1. The tool will take a file as an input (eg., "./PizzaCalc pizzainfo.txt"). The format of this file is as follows. The file is a series of pizza stats, where each entry is 3 lines long. The first line is a name to identify the pizza (a string with no spaces), the second line is the diameter of the pizza in inches (a float),
and the third line is the cost of the pizza in dollars (another float). line of the file is the string "DONE". For example:

```
DominosLarge
14
7.99
DominosMedium
12
6.99 DONE
```

After the last pizza in the list, the last
Your program should output a number of lines equal to the number of pizzas, and each line is the pizza's name and pizza-per-dollar (in2/USD). The lines should be sorted in descending order of pizza-per-dollar, and you must write your own sorting function (you can't just use the qsort library function). Pizzas with equal pizza-per-dollar should be sorted alphabetically (e.g. based on the strcmp function). For example:

```
DominosLarge 19.26633793
DominosMedium 16.17987633
BobsPizza 11.2
JimsPizza 11.2
```

To refresh your memory, the formula for the area of a circle is $\pi r2$, and to compute area based on diameter, $\pi 4\ d2$. Then just divide by cost to get pizza-per-dollar. You should compute using 32-bit floats and define π with:

```
#define PI 3.14159265358979323846
```

You may assume that pizza names will be less than 64 characters.
To mitigate the divide-by-zero situation, if the cost of the pizza is zero, the pizza-per-dollar should simply be zero (as the free pizza must be some kind of trap). A pizza with diameter zero will naturally also have a pizza-per-dollar of zero, as mathematical points are not edible. If your program is fed an empty file the program should print the following and exit:

```
PIZZA FILE IS EMPTY
```

1 This program only optimizes single pizza prices, but of course most pizza deals involve getting multiple pizzas; we'll ignore this fact. We also are ignoring toppings, pizza thickness, quality, etc.
*/

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define PI 3.14159265358979323846

struct Pizza
{
  char pizza_name[64];
```

```
   float diameter;
   float cost;
   struct Pizza* next;
};

struct Pizza*
create_new_node(char name[64], float dia, float cost)
{
   struct Pizza * _node = (struct Pizza*) malloc(sizeof(struct Pizza));
   strcpy(_node->pizza_name, name);
   _node->diameter = dia;
   _node->cost = cost;
   _node->next = NULL;
   return _node;
}

float
get_cost(float dia, float cost)
{
   if(cost == 0) return 0;
   return (PI * dia * dia)/(4 * cost);
}

struct Pizza*
insert_new_node(struct Pizza* head, struct Pizza* node)
{
   float node_cost = get_cost(node->diameter, node->cost);

   struct Pizza * _node = head;
   while ( _node->next &&
         node_cost < get_cost(_node->next->diameter, _node->next->cost))
         _node = _node->next;
   while ( _node -> next &&
         node_cost == get_cost(_node->next->diameter, _node->next->cost) &&
         strcmp(node->pizza_name, _node->next->pizza_name) > 0 )
      _node = _node -> next;

   if(_node == head && node_cost > get_cost(head->diameter, head->cost) ) {
      node->next = head;
      return node;
   }

   struct Pizza* next_node = _node->next;
   _node -> next = node;
   node->next = next_node;
   return head;
}
```

```
int
main(int argc, char const *argv[])
{
   if (argc != 2) return EXIT_FAILURE;
   FILE *fp = fopen(argv[1], "rb");
   size_t nread = 0;
   size_t len = 0;
   char *name = NULL,
        *cost = NULL,
        *diameter = NULL;
   struct Pizza* head = NULL;
   while (1)
   {
      nread = getline(&name, &len, fp);
      if (nread == -1) {
         printf("PIZZA FILE IS EMPTY\n");
         break;
      } //check for empty file
      if(name[0] == '\n') continue; //skip blank line
      char *pos = strchr(name,'\n');
      *pos = '\0'; //getline  created a \n terminated string
      if(strcmp(name,"DONE") == 0) break; // reached end of file
      getline(&diameter, &len, fp);
      getline(&cost, &len, fp);
      if (!head) head = create_new_node(name, atof(diameter), atof(cost));
      else {
         struct Pizza* n = create_new_node(name, atof(diameter), atof(cost));
         head = insert_new_node(head,n);
      }
   }
   fclose(fp);
   free(name);
   free(cost);
   free(diameter);

   struct Pizza* prev_node = head;
   while(head) {
      printf("%s %f\n", head->pizza_name, get_cost(head->diameter, head->cost));
      prev_node = head;
      head = head->next;
      free(prev_node);
   }
   free(head);
   return EXIT_SUCCESS;
}
```