```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define MAX 10000
#define DEBUG 0
int num_strings = 0;
/*
if(DEBUG == 1)
    printf("1");
*/

//define String Structure
struct String {
    char *s;
    struct String *next;
};

struct String *list;
void printstruct(struct String *list);

void input(char **str)
{
    char input[MAX] = {0}; // important to initlize or you will get wierd ? characters
    char c = 0;
    int i = 0;
    START: while ((c = getchar()) != EOF)
    {
        input[i] = c;
        if (c == ';')
        {
            if ((strcmp(input, ";")) == 0)
            {
                str[num_strings] = (char *)malloc(MAX * sizeof(char));
                strcpy(str[num_strings], "");
                for (int f = 0; f < i+2; f++) memset(&input[0], 0, sizeof(input));
                //clear input
                num_strings++;
                i = 0;
                goto START; // if a double ; is found return to getchar
            }
            input[i] = '\0'; //set ; to NULL
            str[num_strings] = (char *)malloc(MAX * sizeof(char));//allocate a memory
            to a node
            strcpy(str[num_strings], input);
            num_strings++;
            for (int f = 0; f < i; f++) memset(&input[0], 0, sizeof(input));//clear input
            i = 0;
            goto START; // don't continue if you capture a word before a ; return to
            getchar
        }
        else
            i++; //if no ; is found then go to next character
    }
    // if EOF is hit then everything left in input must be what's left to parse. and no
    ; remain
    str[num_strings] = (char *)malloc(MAX * sizeof(char));
    strcpy(str[num_strings], input);
    num_strings++;
    for (int f = 0; f < i; f++)
        memset(&input[0], 0, sizeof(input));
}

void get_strings(char **str)
    {
        struct String *temp = (struct String *)malloc(sizeof(struct String));
        //allocate memory to new temp linked list
        temp = NULL;
        list = NULL;
        int i;
```

```
65          for (i = 0; i < num_strings; i++)
66          {
67              struct String *node = (struct String *)malloc(sizeof(struct String));
                //allocate memory to new node linked list
68              node->s = (char *)malloc(strlen(str[i])); //allocate memory for the text of
                new node
69              strcpy(node->s, str[i]);
70              node->next = temp; //set the node into the stack of linked list
71              temp = node; //set the node to start at this new node (LIFO)
72          }
73          //reverse node so that FIFO occurs. Optional
74          for (i = 0; i < num_strings; i++)
75          {
76              struct String *node = (struct String *)malloc(sizeof(struct String));
77              node->s = (char *)malloc(strlen(str[i]));
78              strcpy(node->s, temp->s);
79              node->next = list;
80              list = node;
81              temp = temp->next;
82          }
83  }
84
85  void sort_strings(struct String *list)
86  {
87      struct String *node = (struct String *)malloc(sizeof(struct String));
88      node->s = (char *)malloc(sizeof(char));
89      strcpy(node->s, "");
90      node->next = list;
91      list = node;
92      int i, j;
93      char *a = (char *)malloc(sizeof(char));
94
95      struct String *temp1;
96      struct String *temp2;
97      struct String *pos;
98      struct String *min;
99      int length = num_strings;
100     // bubble sort array
101     temp1 = list;
102     temp2 = temp1->next;
103     i = 1;
104     while (i)
105     {
106         i = 0;
107         while (1)
108         {
109             if (temp2 == NULL) {
110                 break;
111             }
112             if ((strlen(temp1->s)) > (strlen(temp2->s)))
113             {
114                 a = temp1->s;
115                 temp1->s = temp2->s;
116                 temp2->s = a;
117                 i = 1;
118             }
119             temp1 = temp1->next;
120             temp2 = temp1->next;
121         }
122           temp1 = list;
123           temp2 = temp1;
124     }
125 }
126
127 void printstruct(struct String *list)
128 {
129     struct String *temp;
130     struct String *last;
131     temp = list;
```

```
132         while (1) //get last element of linked list
133         {
134             if (temp->next == NULL) break;
135             temp = temp->next;
136         }
137         last = temp;
138         temp = list;
139         while (1)
140         {
141             next:
142                 if (temp == NULL)
143                     break;
144                 if ((strcmp(temp->s, "")) == 0) // if a " " is found that means there was a
                    ;;. print a \n
145                 {
146                     printf("\n");
147                     temp = temp->next;
148                     goto next;
149                 }
150                 else if (temp != last) { // check if we're not on the last node
151                 printf("%s\n", temp->s);
152                 }
153                 else printf("%s", last->s);
154                 temp = temp->next; // increment linked list
155         }
156     }
157
158     void freeStruct(struct String *list) {
159         struct String *runner = list;
160         struct String *next_pointer;
161         while (1)
162         {
163             if (runner == NULL)
164                 break;
165             next_pointer = runner->next;
166             free(runner->s);
167             free(runner);
168             runner = next_pointer;
169         }
170     }
171
172     int main() {
173         char **str;
174         str = (char**)malloc(MAX * sizeof(char*));
175         input(str);
176         get_strings(str);
177         sort_strings(list);
178         printstruct(list);
179         return 0;
180     }
```