

Branch: master ▾

Find file

Copy path

[cs-35l](#) / [assignment5](#) / [sfrobu.c](#)

prithvikannan Update sfrobu.c

d8aec6b on Nov 6

1 contributor

Raw Blame History



286 lines (250 sloc) 6.18 KB

```
1  #include <unistd.h>
2  #include <stdbool.h>
3  #include <string.h>
4  #include <stdlib.h>
5  #include <sys/stat.h>
6  #include <ctype.h>
7  #include <stdio.h>
8
9  bool isF = false;
10
11 // implements comparison between a and b without deobfuscating
12 int frobcmp(char const *a, char const *b)
13 {
14     // make sure pointers are not null
15     if (a != 0 && b != 0)
16     {
17         // iterate through char array with pointers a and b
18         while (*a != ' ' && *b != ' ')
19         {
20             // unfrobnicate a single byte
21
22             char a_i;
23             char b_i;
24
25             if (isF)
26             {
27                 a_i = toupper((unsigned char)(*a ^ 42));
28                 b_i = toupper((unsigned char)(*b ^ 42));
29             }
30             else
31             {
32                 a_i = *a ^ 42;
33                 b_i = *b ^ 42;
34             }
35
36             // compare a and b and check which ends first
37             if (a_i < b_i || *a == ' ')
38             {
39                 return -1;
40             }
41             else if (a_i > b_i || *b == ' ')
42             {
43                 return 1;
44             }
45
46             a++;
47             b++;
48         }
49     }
50     // a and b always equal
51     return 0;
```

```
52 }
53
54 // custom comparator that calls frobcmp
55 int cmp(const void *a, const void *b)
56 {
57     return frobcmp(*(char **)a, *(char **)b);
58 }
59
60 int main(int argc, const char *argv[])
61 {
62     switch (argc)
63     {
64     case 1:
65         isF = false;
66         break;
67     case 2:
68         if (argv[1][0] != '-' && argv[1][1] != 'f')
69         {
70             fprintf(stderr, "Invalid arguments");
71             exit(1);
72         }
73         else
74         {
75             isF = true;
76         }
77         break;
78     default:
79         fprintf(stderr, "Invalid number of arguments");
80         exit(1);
81     }
82
83     struct stat buf;
84     fstat(0, &buf);
85     size_t size;
86     if (fstat(0, &buf) < 0)
87     {
88         fprintf(stderr, "Unable to get info");
89         exit(1);
90     }
91
92     char *regFile;
93     char **arr = NULL;
94     int s = -1;
95     bool addNewString = true;
96
97     if (S_ISREG(buf.st_mode))
98     {
99         size = buf.st_size;
100
101         regFile = (char *)malloc(sizeof(char) * (size + 1));
102         if (read(0, regFile, size) < 0)
103         {
104             fprintf(stderr, "Unable to read");
105             exit(1);
106         }
107
108         int words = 0;
109         int i = 0;
110
111         while (i < size)
112         {
113             // catch first char space
114             if (i == 0 && regFile[i] != ' ')
115             {
116                 words++;
117             }
118         }
119     }
```

```
118
119     if (regFile[i] == ' ')
120     {
121         // handle consecutive spaces by skipping iteration
122         while (regFile[i] == ' ' && i < size)
123         {
124             i++;
125         }
126         if (i < size)
127         {
128             words++;
129         }
130     }
131     i++;
132 }
133 regFile[size] = ' ';
134
135 // allocate memory equal to words
136 arr = (char **)malloc(sizeof(char *) * words);
137 if (arr == NULL)
138 {
139     fprintf(stderr, "Memory allocation error");
140     exit(1);
141 }
142
143 // add words to array
144 for (i = 0; i < size; i++)
145 {
146     if (addNewString && regFile[i] != ' ')
147     {
148         s++;
149         addNewString = false;
150         arr[s] = &regFile[i];
151     }
152     if (!addNewString && regFile[i] == ' ')
153     {
154         addNewString = true;
155     }
156 }
157 }
158 else
159 {
160     arr = (char **)malloc(sizeof(char *));
161     if (arr == NULL)
162     {
163         fprintf(stderr, "Memory allocation error");
164         exit(1);
165     }
166 }
167
168 char *temp_string;
169 char input[1];
170 char current_char;
171 int char_ptr = 0;
172 while (true)
173 {
174
175     int r = read(0, input, 1);
176     if (r == 0)
177     {
178         break;
179     }
180     else if (r < 0)
181     {
182         fprintf(stderr, "Unable to read");
183         exit(1);
184     }
```

```
184     }
185
186     current_char = input[0];
187
188     if (!addNewString)
189     {
190         temp_string = (char *)realloc(temp_string, (char_ptr + 1) * sizeof(char));
191         if (temp_string == NULL)
192         {
193             fprintf(stderr, "Memory allocation error");
194             exit(1);
195         }
196
197         // space is delimiter of new strings
198         if (current_char == ' ')
199         {
200             addNewString = true;
201         }
202     }
203
204     else // if program must create a new string
205     {
206         char_ptr = 0;
207
208         // handle consecutive spaces by skipping iteration
209         if (current_char == ' ' && char_ptr == 0)
210         {
211             continue;
212         };
213
214         s++;
215
216         arr = (char **)realloc(arr, (s + 1) * sizeof(char *));
217         temp_string = (char *)malloc(sizeof(char));
218         if (arr == NULL || temp_string == NULL)
219         {
220             fprintf(stderr, "Memory allocation error");
221             exit(1);
222         }
223         addNewString = false;
224     }
225
226     // add new char after adjusting pointers and allocating memory
227     temp_string[char_ptr] = current_char;
228     arr[s] = temp_string;
229     char_ptr++;
230 }
231
232 // add trailing space if not present
233 if (s != -1 && arr[s][char_ptr - 1] != ' ')
234 {
235
236     temp_string = (char *)realloc(temp_string, (char_ptr + 1) * sizeof(char));
237     if (temp_string == NULL)
238     {
239         fprintf(stderr, "Memory allocation error");
240         exit(1);
241     }
242     temp_string[char_ptr] = ' ';
243     arr[s] = temp_string;
244 }
245
246 // use qsort to sort array of strings
247 qsort(arr, s + 1, sizeof(char *), cmp);
248
249 // print to stdout
```

```
250     int i = 0;
251     while (i < s + 1)
252     {
253
254         int j = 0;
255         while (true)
256         {
257             input[0] = arr[i][j];
258             if (write(1, input, 1) < 0)
259             {
260                 fprintf(stderr, "Unable to write");
261                 exit(1);
262             }
263             // if space then move to next line
264             if (arr[i][j] == ' ')
265             {
266                 break;
267             }
268             j++;
269         }
270
271         if (!S_ISREG(buf.st_mode))
272         {
273             free(arr[i]);
274         }
275         i++;
276     }
277
278     if (S_ISREG(buf.st_mode))
279     {
280         free(regFile);
281     }
282
283     free(arr);
284
285     exit(0);
286 }
```