Midterm 1 Programming Test
Group 4

Q1

```
51
52 int cmp(const void *a, const void *b){
53     // printf("%s : %s\n", *(const char **)a, *(const char **)b);
54     return strcmp(*(const char **)a, *(const char **)b);
55 }
56
```

To implement sort q1.txt, we first declare a cmp function to compare strings.

```
137     // Sort Function
138     qsort(list, tot_index, sizeof(const char *), cmp);
```

Then, we use qsort to sort strings.

```
146     // Unique Function
147     for (int i = tot_index - 1; i >= 0; i--)
148     {
149         if(i == 0)
150         {
151             print[i] = 1;
152         }
153         else
154         {
155
156             if (find_func(list, list[i], count, i))
157             {
158                 print[i] = 1;
159             }
160         }
161
162     }
```

After that, we implement **uniq** – function using find_func and compare_func

```
28 int find_func(char **list, char * tar, int *count, int cur_index){
29
30     // return 0 if it is not unique
31     // return 1 if it is unique
32
33     if (compare_func(list[cur_index - 1], tar) != 0)
34     {
35
36         if (params_u == 1 && cur_index + 1 < tot_index  && compare_func(list[cur_index + 1], tar) == 0)
37         {
38             return 0;
39         }
40         else
41         {
42             return 1;
43         }
44     }
45     else
46     {
47         count[cur_index - 1] += count[cur_index];
48         return 0;
49     }
50 }
```

```
16 int compare_func(char *s1, char *s2){
17
18     if(params_i == 1){
19         return strcasecmp(s1, s2);
20     }
21     else{
22         return strcmp(s1, s2);
23     }
24 }
```

In compare_func, we take -i parameter into consideration to make sure which string compare function to use. strcasecmp for ignoring case, while strcmp for case sensitive.

```
166     // Print Function
167     for(int i = 0;i < tot_index;i++){
168
169         if(print[i] == 1){
170             if (params_c == 1)
171             {
172                 printf("      %d %s", count[i], list[i]);
173             }
174             else
175             {
176                 printf("%s", list[i]);
177             }
178         }
179
180     }
```

Finally, we print the result. If the parameter -c is added, we print the number of repeated strings.

Result of q1:

```
freebsd@generic:~/Advanced-UNIX-Programming_Student/midterm1 % ./q1 q1.txt
Bird
Camel
Elephant
Fish
KOALA
Koala
Shark
Whale
alligator
bear
camel
dolphin
fish
koala
lion
shark
whale
```

-c parameter

```
freebsd@generic:~/Advanced-UNIX-Programming_Student/midterm1 % ./q1 -c q1.txt
      1 Bird
      1 Camel
      1 Elephant
      1 Fish
      1 KOALA
      1 Koala
      1 Shark
      2 Whale
      1 alligator
      3 bear
      2 camel
      1 dolphin
      2 fish
      1 koala
      1 lion
      1 shark
      2 whale
```

-i parameter

```
freebsd@generic:~/Advanced-UNIX-Programming_Student/midterm1 % ./q1 -i q1.txt
Bird
Camel
Elephant
Fish
KOALA
Shark
Whale
alligator
bear
camel
dolphin
fish
koala
lion
shark
whale
```

-u parameter

```
freebsd@generic:~/Advanced-UNIX-Programming_Student/midterm1 % ./q1 -u q1.txt
Bird
Camel
Elephant
Fish
KOALA
Koala
Shark
alligator
dolphin
koala
lion
shark
```

Q2

1. The memory size significantly affects the CPU time in user mode when using fread and fwrite. Because the fread and fwrite involve system calls (e.g., read and write) to communicate with the operating system's kernel, In this scenario, it will spend a lot of time in user mode.
2. fgets and fputs are also user mode operations. They indirectly lead to system mode operations when data is transferred between user-level buffers and physical storage devices via system calls. However, we open the buffer size as 4096 bytes, so it will not take a long time in user mode.
3. fgetc and fputc are also commands in user mode. They read and write data character by character, so it takes a lot of time to loop to read and write in user mode.

	User CPU (s)	System CPU (s)	Clock Time (s)
fread/fwrite with 1 byte	10.639	0.028	10.667
fread/fwrite with 32 bytes	0.319	0.033	0.352
fread/fwrite with 1024 bytes	0.017	0.028	0.045
fread/fwrite with 4096 bytes	0.006	0.033	0.039
fgets/fputs with 4096 bytes	0.033	0.023	0.056
fgetc/fputc	11.214	0.036	11.250