

VSCODE_PRINT_SCRIPT_TAGS

Selected files

8 printable files

Assignments\Assignment2\cmp.c
Assignments\Assignment2\codecA.c
Assignments\Assignment2\codecB.c
Assignments\Assignment2\copy.c
Assignments\Assignment2\decode.c
Assignments\Assignment2\encode.c
Assignments\Assignment2\Makefile
Assignments\Assignment2\stshell.c

Assignments\Assignment2\cmp.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  int cmp(char *file_name1, char *file_name2, int flagI){
6      FILE *file1, *file2;
7
8      file1 = fopen(file_name1, "rb");
9      if(file1 == NULL){
10         perror("file does not exist");
11         exit(1);
12     }
13
14     file2 = fopen(file_name2, "rb");
15     if(file1 == NULL){
16         perror("file does not exist");
17         exit(1);
18     }
19
20     while(!feof(file1) || !feof(file2)){
21         char b1, b2;
22         fread(&b1, sizeof(char), 1, file1);
23         fread(&b2, sizeof(char), 1, file2);
24         if(flagI){
25             if((b1 - b2 != 0) && (b1 - b2 != ('a' - 'A')) && (b1 - b2 != ('A' - 'a'))){
26                 fclose(file1);
27                 fclose(file2);
28                 return 1;
29             }
30         }
31         else if(b1 != b2){
32             fclose(file1);
33             fclose(file2);
34             return 1;
35         }
36     }
37
38     if(feof(file1) != feof(file2)){
39         fclose(file1);
40         fclose(file2);
41         return 1;
42     }
```

```

42     }
43     fclose(file1);
44     fclose(file2);
45     return 0;
46 }
47
48 int main(int argc, char *argv[]){
49     if (argc < 3){
50         printf("to use the program please use this format: ./cmp <file1> <file2>\n");
51         printf("add -i to ignore lower/upper case\n");
52         printf("add -v for verbal output\n");
53         return 1;
54     }
55     char* file_name1 = argv[1];
56     char* file_name2 = argv[2];
57     int flagI = 0, flagV = 0;
58     int func_ret;
59
60     for(int i = 3; i < argc; i++){
61         if(!strcmp(argv[i], "-v")){
62             flagV = 1;
63         }
64         if(!strcmp(argv[i], "-i")){
65             flagI = 1;
66         }
67     }
68
69     func_ret = cmp(file_name1, file_name2, flagI);
70
71     if(flagV){
72         if(!func_ret){
73             printf("equal\n");
74         }
75         else{
76             printf("distinct\n");
77         }
78     }
79     return func_ret;
80 }

```

Assignments\Assignment2\codecA.c

```

1  #define DIFF 'A' - 'a'
2
3  void encode(char* str){
4      for(int i = 0; str[i] != '\0'; i++){
5          if(str[i] >= 'a' && str[i] <= 'z'){
6              str[i] += DIFF;
7          }
8          else if(str[i] >= 'A' && str[i] <= 'Z'){
9              str[i] -= DIFF;
10         }
11     }
12 }
13
14 void decode(char* str){
15     encode(str);

```

```
16 | }
```

Assignments\Assignment2\codecB.c

```
1 void encode(char* str){
2     for(int i = 0; str[i] != '\0'; i++){
3         str[i] += 3;
4     }
5 }
6
7 void decode(char* str){
8     for(int i = 0; str[i] != '\0'; i++){
9         str[i] -= 3;
10    }
11 }
12 }
13
```

Assignments\Assignment2\copy.c

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <unistd.h>
4
5 int copy(char* file_name1, char* file_name2, int flagF){
6     FILE *file1, *file2;
7
8     file1 = fopen(file_name1, "rb");
9     if(file1 == NULL){
10         perror("error while opening file");
11         return -1;
12     }
13
14     if(flagF || (access(file_name2, F_OK))){
15         file2 = fopen(file_name2, "wb");
16         if(file2 == NULL){
17             perror("error while opening file");
18             return -1;
19         }
20         size_t bytes_read;
21         char buffer[1];
22         while ((bytes_read = fread(buffer, sizeof(char), 1, file1)) == 1){
23             fwrite(buffer, sizeof(char), 1, file2);
24         }
25         fclose(file1);
26         fclose(file2);
27         return 0;
28     }
29
30     else{
31         fclose(file1);
32         return 1;
33     }
34 }
35
```

```

36 int main(int argc, char *argv[]){
37     if (argc < 3){
38         printf("to use the program please use this format: ./copy <file1> <file2>\n");
39         printf("add -f to compel the replication\n");
40         printf("add -v for verbal output\n");
41         return 1;
42     }
43
44     char* file_name1 = argv[1];
45     char* file_name2 = argv[2];
46     int flagF = 0, flagV = 0;
47     int rv;
48
49     for(int i = 3; i < argc; i++){
50         if(!strcmp(argv[i], "-v")){
51             flagV = 1;
52         }
53         if(!strcmp(argv[i], "-f")){
54             flagF = 1;
55         }
56     }
57
58     rv = copy(file_name1, file_name2, flagF);
59
60
61     if(flagV){
62         if(rv == -1){
63             printf("general failure\n");
64             return 1;
65         }
66         else if(rv == 0){
67             printf("success\n");
68         }
69         else{
70             printf("target file exist\n");
71         }
72     }
73     return rv;
74 }

```

Assignments\Assignment2\decode.c

```

1  #include <stdio.h>
2  #include <dlfcn.h>
3  #include "string.h"
4  #include <stdlib.h>
5
6  int main(int argc, char* argv[]){
7      if (argc < 2){
8          printf("please use the format: ./encode <codec> <message>\n");
9          return 1;
10     }
11
12     int total_length = 0;
13     for (int i = 1; i < argc; i++) {
14         total_length += strlen(argv[i]) + 1;
15     }
16

```

```

17     char* message = (char*) malloc(total_length);
18     if (!message) {
19         printf("malloc failed");
20         return 1;
21     }
22     message[0] = '\0';
23
24
25     for (int i = 2; i < argc ; ++i) {
26         strcat(message,argv[i]);
27         if(i+1 < argc)
28             strcat(message, " ");
29     }
30
31     strcat(message,"\0");
32
33     char * codec = argv[1];
34     char* library_name = (char*)malloc(strlen(codec) + 10);
35     sprintf(library_name, "./%s", codec);
36     void* program = dlopen(library_name, RTLD_NOW);
37     free(library_name);
38
39     if(!program){
40         printf("Error: %s\n", dlerror());
41         printf("error (1)\n");
42         return 1;
43     }
44     void (*decode)(char*) = dlsym(program,"decode");
45     if(!decode){
46         printf("error (2)\n");
47         return 1;
48     }
49     decode(message);
50     printf("decode %s %s\n",argv[1],message);
51     dlclose(program);
52     free(message);
53     return 0;
54 }

```

Assignments\Assignment2\encode.c

```

1  #include <stdio.h>
2  #include <dlfcn.h>
3  #include "string.h"
4  #include <stdlib.h>
5
6  int main(int argc, char* argv[]){
7      if (argc < 2){
8          printf("please use the format: ./encode <codec> <message>\n");
9          return 1;
10     }
11
12     int total_length = 0;
13     for (int i = 1; i < argc; i++) {
14         total_length += strlen(argv[i]) + 1;
15     }
16     char* message = (char*) malloc(total_length);
17     if (!message) {

```

```

18     printf("malloc failed");
19     return 1;
20 }
21 message[0] = '\0';
22
23
24 for (int i = 2; i < argc ; ++i) {
25     strcat(message,argv[i]);
26     if(i+1 < argc)
27         strcat(message," ");
28
29 }
30 strcat(message,"\0");
31
32 char * codec = argv[1];
33 char* library_name = (char*)malloc(strlen(codec) + 10);
34 sprintf(library_name, "./%s", codec);
35 void* program = dlopen(library_name, RTLD_NOW);
36 free(library_name);
37
38 if(!program){
39     printf( "Error: %s\n", dlerror());
40     printf("error (1)\n");
41     return 1;
42 }
43 void (*encode)(char*) = dlsym(program,"encode");
44 if(!encode){
45     printf("error (2)\n");
46     return 1;
47 }
48 encode(message);
49 printf("encode %s %s\n",argv[1],message);
50 dlclose(program);
51 free(message);
52 return 0;
53 }

```

Assignments\Assignment2\Makefile

```

1 CC = gcc
2 CFLAGS = -c -Wall -Werror -fPIC
3 LDFLAGS = -shared -Wall -ldl
4 RM = rm -f
5
6
7 .PHONY: all clean cleantxt
8
9 all: PartA PartB PartC
10
11 PartA: cmp copy
12
13 PartB: codecA codecB encode decode
14
15 PartC: stshell
16
17 cmp: cmp.c
18     $(CC) -Wall -g cmp.c -o cmp
19

```

```

20 copy: copy.c
21     $(CC) -Wall -g copy.c -o copy
22
23 codecA.o: codecA.c
24     $(CC) ${CFLAGS} codecA.c -o codecA.o
25
26 codecA: codecA.o
27     $(CC) ${LDFLAGS} codecA.o -o codecA
28
29 codecB.o: codecB.c
30     $(CC) ${CFLAGS} codecB.c -o codecB.o
31
32 codecB: codecB.o
33     $(CC) ${LDFLAGS} codecB.o -o codecB
34
35 encode.o: encode.c
36     $(CC) -c -Wall encode.c -o encode.o
37
38 encode: encode.o
39     $(CC) -Wall -g encode.o -o encode
40
41 decode.o: decode.c
42     $(CC) -c -Wall -g decode.c -o decode.o
43
44 decode: decode.o
45     $(CC) -Wall -g decode.o -o decode
46
47 shell.o: stshell.c
48     $(CC) -c -Wall -g stshell.c -o shell.o
49
50 stshell: shell.o
51     $(CC) -Wall -g shell.o -o stshell
52
53 clean:
54     rm -rf *.o cmp copy encode decode codecA codecB stshell
55 cleantxt:
56     rm -rf *.txt

```

Assignments\Assignment2\stshell.c

```

1  #include <sys/stat.h>
2  #include <sys/wait.h>
3  #include <fcntl.h>
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <errno.h>
7  #include <unistd.h>
8  #include <string.h>
9  #include <signal.h>
10
11 int main() {
12     int status;
13     int argc;
14     char *argv[20];
15     char command[1024];
16     char *token;
17
18     // ignoring ctrl + c

```

```

19     signal(SIGINT, SIG_IGN);
20
21     while (1) {
22         char path[1024];
23         getcwd(path, sizeof(path));
24         printf("\033[1;32mAviv&Alon's_shell\033[0m:\033[1;34m~%s\033[37m$\033[0m ", path);
25         fgets(command, 1024, stdin);
26         command[strlen(command) - 1] = '\0'; // replace \n with \0
27
28         /* parse command line */
29         argc = 0;
30         token = strtok(command, " ");
31         while (token != NULL) {
32             argv[argc] = token;
33             token = strtok(NULL, " ");
34             argc++;
35         }
36         argv[argc] = NULL;
37         if (argc > 0) {
38             if (strcmp("exit", argv[0]) == 0) {
39                 exit(0);
40             }
41         }
42
43         /* Is command empty */
44         if (argv[0] == NULL)
45             continue;
46
47         int is_simple = 1;
48         int pipes_n = 0;
49         int pipes[2] = {-1, -1};
50         int redirect = 0;
51         int redirect_to = 0;
52
53         for (int j = 1; j < argc; ++j) {
54             if (strcmp(argv[j], ">") == 0) {
55                 redirect = 1;
56                 redirect_to = j + 1;
57                 is_simple = 0;
58                 // printf("found >\n");
59             } else if (strcmp(argv[j], ">>") == 0) {
60                 redirect = 2;
61                 redirect_to = j + 1;
62                 is_simple = 0;
63                 // printf("found >>\n");
64             } else if (strcmp(argv[j], "|") == 0) {
65                 // printf("found | in %d\n ", j);
66                 pipes[pipes_n++] = j;
67                 is_simple = 0;
68             }
69         }
70
71         /* for commands not part of the shell command language */
72         pid_t id1 = fork();
73         if (id1 != 0) {
74             wait(&status);
75             if (WIFEXITED(status)) {
76                 int exit_code = WEXITSTATUS(status);
77                 if (exit_code) {
78                     printf("\033[0;31merror: \033[0m");

```



```

79         if(exit_code == 1){
80             printf("Failed while openning pipes\n");
81         }
82         if(exit_code == 2){
83             printf("Missing file name after `>`\n");
84         }
85         if(exit_code == 3){
86             printf("Missing file name after `>>`\n");
87         }
88         printf("(Exit code %d)\n", exit_code);
89     }
90 }
91 }
92 else {
93     signal(SIGINT, SIG_DFL);
94
95     if (is_simple) {
96         execvp(argv[0], argv);
97     } else if (pipes_n == 1) {
98         int fd[2];
99         if (pipe(fd) == -1) {
100             exit(1);
101         }
102
103         pid_t id2 = fork();
104         if (id2 == 0) {
105             close(fd[0]);
106             dup2(fd[1], 1);
107             close(fd[1]);
108             char *argv2[10];
109             int j;
110             for (j = 0; j < pipes[0]; ++j) {
111                 argv2[j] = argv[j];
112             }
113             argv2[j] = NULL;
114             execvp(argv2[0], argv2);
115         } else {
116             pid_t id3 = fork();
117             if (id3 == 0) {
118                 waitpid(id2, NULL, 0);
119                 close(fd[1]);
120                 dup2(fd[0], 0);
121                 close(fd[0]);
122
123                 int till = argc;
124                 if (redirect) {
125                     till = redirect_to - 1;
126                     if(redirect == 1){
127                         if (argc <= redirect_to) {
128                             printf("\033[0;31merror: \033[0m");
129                             printf("Missing file name after `>`\n");
130                             printf("(Exit code 2)\n");
131                             exit(2);
132                         } else {
133                             FILE *fd = fopen(argv[redirect_to], "w");
134                             int fout = fileno(fd);
135                             dup2(fout, 1);
136                             fclose(fd);
137                         }
138                     }

```

```

139         if(redirect == 2){
140             if ( argc <= redirect_to) {
141                 printf("\033[0;31merror: \033[0m");
142                 printf("Missing file name after `>>`\n");
143                 printf("(Exit code 3)\n");
144                 exit(3);
145             } else {
146                 FILE *fd = fopen(argv[redirect_to], "a");
147                 int fout = fileno(fd);
148                 dup2(fout, 1);
149                 fclose(fd);
150             }
151         }
152     }
153
154     char *argv3[10];
155     int i, k;
156     for (i = pipes[0] + 1, k = 0; i < till; ++i, k++) {
157         argv3[k] = argv[i];
158     }
159     argv3[k] = NULL;
160     execvp(argv3[0], argv3);
161 } else {
162     close(fd[0]);
163     close(fd[1]);
164     waitpid(id3, NULL, 0);
165 }
166 }
167 } else if (pipes_n == 2) {
168     int fd1[2], fd2[2];
169     if (pipe(fd1) == -1 || pipe(fd2) == -1) {
170         exit(1);
171     }
172
173     pid_t id2 = fork();
174     if (id2 == 0) {
175         close(fd2[0]);
176         close(fd2[1]);
177
178         close(fd1[0]);
179         dup2(fd1[1], 1);
180         close(fd1[1]);
181
182         char *argv2[10];
183         int j, k;
184         for (j = 0, k = 0; j < pipes[0]; ++j, ++k) {
185             argv2[k] = argv[j];
186         }
187         argv2[k] = NULL;
188         execvp(argv2[0], argv2);
189     } else {
190         pid_t id3 = fork();
191         if (id3 == 0) {
192             waitpid(id2, NULL, 0);
193             close(fd1[1]);
194             dup2(fd1[0], 0);
195             close(fd1[0]);
196
197             close(fd2[0]);
198             dup2(fd2[1], 1);

```

```

199         close(fd2[1]);
200
201         int till = pipes[1];
202         char *argv3[10];
203         int i, k;
204         for (i = pipes[0] + 1, k = 0; i < till; ++i, ++k) {
205             argv3[k] = argv[i];
206         }
207         argv3[k] = NULL;
208         execvp(argv3[0], argv3);
209     } else {
210         pid_t id4 = fork();
211         if (id4 == 0) {
212             waitpid(id3, NULL, 0);
213             close(fd1[0]);
214             close(fd1[1]);
215
216             close(fd2[1]);
217             dup2(fd2[0], 0);
218             close(fd2[0]);
219
220             int till = argc;
221             if (redirect) {
222                 till = redirect_to - 1;
223                 if(redirect == 1){
224                     if ( argc <= redirect_to) {
225                         printf("\033[0;31merror: \033[0m");
226                         printf("Missing file name after `>`\n");
227                         printf("(Exit code 2)\n");
228                         exit(2);
229                     } else {
230                         FILE *fd = fopen(argv[redirect_to], "w");
231                         int fout = fileno(fd);
232                         dup2(fout, 1);
233                         fclose(fd);
234                     }
235                 }
236                 if(redirect == 2){
237                     if ( argc <= redirect_to) {
238                         printf("\033[0;31merror: \033[0m");
239                         printf("Missing file name after `>>`\n");
240                         printf("(Exit code 3)\n");
241                         exit(3);
242                     } else {
243                         FILE *fd = fopen(argv[redirect_to], "a");
244                         int fout = fileno(fd);
245                         dup2(fout, 1);
246                         fclose(fd);
247                     }
248                 }
249             }
250
251             char *argv4[10];
252             int i, k;
253             for (i = pipes[1] + 1, k = 0; i < till; ++i, ++k) {
254                 argv4[k] = argv[i];
255             }
256             argv4[k] = NULL;
257             execvp(argv4[0], argv4);
258         } else {

```

```
259         close(fd1[0]);
260         close(fd1[1]);
261         close(fd2[1]);
262         close(fd2[0]);
263         waitpid(id4, NULL, 0);
264     }
265 }
266 }
267 } else if (redirect == 1) {
268     if ( argc <= redirect_to) {
269         exit(2);
270     } else {
271         char *argv2[10];
272         int i;
273         for(i = 0; i < redirect_to - 1; i++){
274             argv2[i] = argv[i];
275         }
276         argv2[i] = NULL;
277         FILE *fd = fopen(argv[redirect_to], "w");
278         int fout = fileno(fd);
279         dup2(fout, 1);
280         fclose(fd);
281         execvp(argv2[0], argv2);
282     }
283 } else if (redirect == 2) {
284     if ( argc <= redirect_to) {
285         exit(3);
286     } else {
287         char *argv2[10];
288         int i;
289         for(i = 0; i < redirect_to - 1; i++){
290             argv2[i] = argv[i];
291         }
292         argv2[i] = NULL;
293         FILE *fd = fopen(argv[redirect_to], "a");
294         int fout = fileno(fd);
295         dup2(fout, 1);
296         fclose(fd);
297         execvp(argv2[0], argv2);
298     }
299 }
300 wait(NULL);
301 exit(0);
302 }
303 }
304 }
```