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

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

int cmp(char *file_name1, char *file_name2, int flagI){
    FILE *file1, *file2;

    file1 = fopen(file_name1, "rb");
    if(file1 == NULL){
        perror("file does not exist");
        exit(1);
    }

    file2 = fopen(file_name2, "rb");
    if(file1 == NULL){
        perror("file does not exist");
        exit(1);
    }

    while(!feof(file1) || !feof(file2)){
        char b1, b2;
        fread(&b1, sizeof(char), 1, file1);
        fread(&b2, sizeof(char), 1, file2);
        if(flagI){
            if((b1 - b2 != 0) && (b1 - b2 != ('a' - 'A')) && (b1 - b2 != ('A' - 'a'))){
                fclose(file1);
                fclose(file2);
                return 1;
            }
        }
        else if(b1 != b2){
            fclose(file1);
            fclose(file2);
            return 1;
        }
    }

    if(feof(file1) != feof(file2)){
        fclose(file1);
        fclose(file2);
        return 1;
```

```
    }
    fclose(file1);
    fclose(file2);
    return 0;
}

int main(int argc, char *argv[]){
    if (argc < 3){
        printf("to use the program please use this format: ./cmp <file1> <file2>\n");
        printf("add -i to ignore lower/upper case\n");
        printf("add -v for verbal output\n");
        return 1;
    }
    char* file_name1 = argv[1];
    char* file_name2 = argv[2];
    int flagI = 0, flagV = 0;
    int func_ret;

    for(int i = 3; i < argc; i++){
        if(!strcmp(argv[i], "-v")){
            flagV = 1;
        }
        if(!strcmp(argv[i], "-i")){
            flagI = 1;
        }
    }

    func_ret = cmp(file_name1, file_name2,flagI);

    if(flagV){
        if(!func_ret){
            printf("equal\n");
        }
        else{
            printf("distinct\n");
        }
    }
    return func_ret;
}
```

Assignments\Assignment2\codecA.c

```
#define DIFF 'A' - 'a'

void encode(char* str){
    for(int i = 0; str[i] != '\0'; i++){
        if(str[i] >= 'a' && str[i] <= 'z'){
            str[i] += DIFF;
        }
        else if(str[i] >= 'A' && str[i] <= 'Z'){
            str[i] -= DIFF;
        }
    }
}

void decode(char* str){
    encode(str);
```

```
}
```

Assignments\Assignment2\codecB.c

```
void encode(char* str){
    for(int i = 0; str[i] != '\0'; i++){
        str[i] += 3;
    }
}

void decode(char* str){
    for(int i = 0; str[i] != '\0'; i++){
        str[i] -= 3;

    }
}

```

Assignments\Assignment2\copy.c

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

int copy(char* file_name1, char* file_name2, int flagF){
    FILE *file1, *file2;

    file1 = fopen(file_name1, "rb");
    if(file1 == NULL){
        perror("error while opening file");
        return -1;
    }

    if(flagF || (access(file_name2, F_OK))){
        file2 = fopen(file_name2, "wb");
        if(file2 == NULL){
            perror("error while opening file");
            return -1;
        }
        size_t bytes_read;
        char buffer[1];
        while ((bytes_read = fread(buffer, sizeof(char), 1, file1)) == 1){
            fwrite(buffer, sizeof(char), 1, file2);
        }
        fclose(file1);
        fclose(file2);
        return 0;
    }

    else{
        fclose(file1);
        return 1;
    }
}

```

```c
int main(int argc, char *argv[]){
    if (argc < 3){
        printf("to use the program please use this format: ./copy <file1> <file2>\n");
        printf("add -f to compel the replication\n");
        printf("add -v for verbal output\n");
        return 1;
    }

    char* file_name1 = argv[1];
    char* file_name2 = argv[2];
    int flagF = 0, flagV = 0;
    int rv;

    for(int i = 3; i < argc; i++){
        if(!strcmp(argv[i], "-v")){
            flagV = 1;
        }
        if(!strcmp(argv[i], "-f")){
            flagF = 1;
        }
    }

    rv = copy(file_name1, file_name2,flagF);


    if(flagV){
        if(rv == -1){
            printf("general failure\n");
            return 1;
        }
        else if(rv == 0){
            printf("success\n");
        }
        else{
            printf("target file exist\n");
        }
    }
    return rv;
}
```

Assignments\Assignment2\decode.c

```c
#include <stdio.h>
#include <dlfcn.h>
#include "string.h"
#include <stdlib.h>

int main(int argc, char* argv[]){
    if (argc < 2){
        printf("please use the format: ./encode <codec> <message>\n");
        return 1;
    }

    int total_length = 0;
    for (int i = 1; i < argc; i++) {
        total_length += strlen(argv[i]) + 1;
    }

```

```
    char* message = (char*) malloc(total_length);
    if (!message) {
        printf("malloc failed");
        return 1;
    }
    message[0] = '\0';


    for (int i = 2; i < argc ; ++i) {
        strcat(message,argv[i]);
        if(i+1 < argc)
            strcat(message," ");

    }
    strcat(message,"\0");

    char * codec = argv[1];
    char* library_name = (char*)malloc(strlen(codec) + 10);
    sprintf(library_name, "./%s", codec);
    void* program = dlopen(library_name, RTLD_NOW);
    free(library_name);

    if(!program){
        printf( "Error: %s\n", dlerror());
        printf("error (1)\n");
        return 1;
    }
    void (*decode)(char*) = dlsym(program,"decode");
    if(!decode){
        printf("error (2)\n");
        return 1;
    }
    decode(message);
    printf("decode %s %s\n",argv[1],message);
    dlclose(program);
    free(message);
    return 0;
}
```

Assignments\Assignment2\encode.c

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

int main(int argc, char* argv[]){
    if (argc < 2){
        printf("please use the format: ./encode <codec> <message>\n");
        return 1;
    }

    int total_length = 0;
    for (int i = 1; i < argc; i++) {
        total_length += strlen(argv[i]) + 1;
    }
    char* message = (char*) malloc(total_length);
    if (!message) {
```

```
        printf("malloc failed");
        return 1;
    }
    message[0] = '\0';


    for (int i = 2; i < argc ; ++i) {
        strcat(message,argv[i]);
        if(i+1 < argc)
            strcat(message," ");

    }
    strcat(message,"\0");

    char * codec = argv[1];
    char* library_name = (char*)malloc(strlen(codec) + 10);
    sprintf(library_name, "./%s", codec);
    void* program = dlopen(library_name, RTLD_NOW);
    free(library_name);

    if(!program){
        printf( "Error: %s\n", dlerror());
        printf("error (1)\n");
        return 1;
    }
    void (*encode)(char*) = dlsym(program,"encode");
    if(!encode){
        printf("error (2)\n");
        return 1;
    }
    encode(message);
    printf("encode %s %s\n",argv[1],message);
    dlclose(program);
    free(message);
    return 0;
}
```

Assignments\Assignment2\Makefile

```
CC = gcc
CFLAGS = -c -Wall -Werror -fPIC
LDFLAGS = -shared -Wall -ldl
RM = rm -f


.PHONY: all clean cleantxt

all: PartA PartB PartC

PartA: cmp copy

PartB: codecA codecB encode decode

PartC: stshell

cmp: cmp.c
    $(CC) -Wall -g cmp.c -o cmp

```

```
copy: copy.c
    $(CC) -Wall -g copy.c -o copy

codecA.o: codecA.c
    $(CC) ${CFLAGS}  codecA.c -o codecA.o

codecA: codecA.o
    $(CC) ${LDFLAGS} codecA.o -o codecA

codecB.o: codecB.c
    $(CC) ${CFLAGS}  codecB.c -o codecB.o

codecB: codecB.o
    $(CC) ${LDFLAGS} codecB.o -o codecB

encode.o: encode.c
    $(CC) -c -Wall encode.c -o encode.o

encode: encode.o
    $(CC) -Wall -g encode.o -o encode

decode.o: decode.c
    $(CC) -c -Wall -g decode.c -o decode.o

decode: decode.o
    $(CC) -Wall -g decode.o -o decode

shell.o: stshell.c
    $(CC) -c -Wall -g stshell.c -o shell.o

stshell: shell.o
    $(CC) -Wall -g shell.o -o stshell

clean:
    rm -rf *.o cmp copy encode decode codecA codecB stshell
cleantxt:
    rm -rf *.txt
```

Assignments\Assignment2\stshell.c

```
#include <sys/stat.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
#include <string.h>
#include <signal.h>

int main() {
    int status;
    int argc;
    char *argv[20];
    char command[1024];
    char *token;

    // ignoring ctrl + c
```

```
    signal(SIGINT, SIG_IGN);

    while (1) {
        char path[1024];
        getcwd(path, sizeof(path));
        printf("\033[1;32mAviv&Alon's_shell\033[0m:\033[1;34m~%s\033[37m$\033[0m ", path);
        fgets(command, 1024, stdin);
        command[strlen(command) - 1] = '\0'; // replace \n with \0

        /* parse command line */
        argc = 0;
        token = strtok(command, " ");
        while (token != NULL) {
            argv[argc] = token;
            token = strtok(NULL, " ");
            argc++;
        }
        argv[argc] = NULL;
        if (argc > 0) {
            if (strcmp("exit", argv[0]) == 0) {
                exit(0);
            }
        }

        /* Is command empty */
        if (argv[0] == NULL)
            continue;

        int is_simple = 1;
        int pipes_n = 0;
        int pipes[2] = {-1, -1};
        int redirect = 0;
        int redirect_to = 0;

        for (int j = 1; j < argc; ++j) {
            if (strcmp(argv[j], ">") == 0) {
                redirect = 1;
                redirect_to = j + 1;
                is_simple = 0;
                // printf("found >\n");
            } else if (strcmp(argv[j], ">>") == 0) {
                redirect = 2;
                redirect_to = j + 1;
                is_simple = 0;
                // printf("found >>\n");
            } else if (strcmp(argv[j], "|") == 0) {
                // printf("found | in %d\n ", j);
                pipes[pipes_n++] = j;
                is_simple = 0;
            }
        }

        /* for commands not part of the shell command language */
        pid_t id1 = fork();
        if (id1 != 0) {
            wait(&status);
            if (WIFEXITED(status)) {
                int exit_code = WEXITSTATUS(status);
                if (exit_code) {
                    printf("\033[0;31merror: \033[0m");
```

```
                    if(exit_code == 1){
                        printf("Failed while openning pipes\n");
                    }
                    if(exit_code == 2){
                        printf("Missing file name after `>`\n");
                    }
                    if(exit_code == 3){
                        printf("Missing file name after `>>`\n");
                    }
                    printf("(Exit code %d)\n", exit_code);
                }
            }
        }
        else {
            signal(SIGINT, SIG_DFL);

            if (is_simple) {
                execvp(argv[0], argv);
            } else if (pipes_n == 1) {
                int fd[2];
                if (pipe(fd) == -1) {
                    exit(1);
                }

                pid_t id2 = fork();
                if (id2 == 0) {
                    close(fd[0]);
                    dup2(fd[1], 1);
                    close(fd[1]);
                    char *argv2[10];
                    int j;
                    for (j = 0; j < pipes[0]; ++j) {
                        argv2[j] = argv[j];
                    }
                    argv2[j] = NULL;
                    execvp(argv2[0], argv2);
                } else {
                    pid_t id3 = fork();
                    if (id3 == 0) {
                        waitpid(id2, NULL, 0);
                        close(fd[1]);
                        dup2(fd[0], 0);
                        close(fd[0]);

                        int till = argc;
                        if (redirect) {
                            till = redirect_to - 1;
                            if(redirect == 1){
                                if (argc <= redirect_to) {
                                    printf("\033[0;31merror: \033[0m");
                                    printf("Missing file name after `>`\n");
                                    printf("(Exit code 2)\n");
                                    exit(2);
                                } else {
                                    FILE *fd = fopen(argv[redirect_to], "w");
                                    int fout = fileno(fd);
                                    dup2(fout, 1);
                                    fclose(fd);
                                }
                            }
```

```
                        if(redirect == 2){
                            if ( argc <= redirect_to) {
                                printf("\033[0;31merror: \033[0m");
                                printf("Missing file name after `>>`\n");
                                printf("(Exit code 3)\n");
                                exit(3);
                            } else {
                                FILE *fd = fopen(argv[redirect_to], "a");
                                int fout = fileno(fd);
                                dup2(fout, 1);
                                fclose(fd);
                            }
                        }
                    }

                    char *argv3[10];
                    int i, k;
                    for (i = pipes[0] + 1, k = 0; i < till; ++i, k++) {
                        argv3[k] = argv[i];
                    }
                    argv3[k] = NULL;
                    execvp(argv3[0], argv3);
                } else {
                    close(fd[0]);
                    close(fd[1]);
                    waitpid(id3, NULL, 0);
                }
            }
        } else if (pipes_n == 2) {
            int fd1[2], fd2[2];
            if (pipe(fd1) == -1 || pipe(fd2) == -1) {
                exit(1);
            }

            pid_t id2 = fork();
            if (id2 == 0) {
                close(fd2[0]);
                close(fd2[1]);

                close(fd1[0]);
                dup2(fd1[1], 1);
                close(fd1[1]);

                char *argv2[10];
                int j, k;
                for (j = 0, k = 0; j < pipes[0]; ++j, ++k) {
                    argv2[k] = argv[j];
                }
                argv2[k] = NULL;
                execvp(argv2[0], argv2);
            } else {
                pid_t id3 = fork();
                if (id3 == 0) {
                    waitpid(id2, NULL, 0);
                    close(fd1[1]);
                    dup2(fd1[0], 0);
                    close(fd1[0]);

                    close(fd2[0]);
                    dup2(fd2[1], 1);
```

```
                    close(fd2[1]);

                    int till = pipes[1];
                    char *argv3[10];
                    int i, k;
                    for (i = pipes[0] + 1, k = 0; i < till; ++i, ++k) {
                        argv3[k] = argv[i];
                    }
                    argv3[k] = NULL;
                    execvp(argv3[0], argv3);
                } else {
                    pid_t id4 = fork();
                    if (id4 == 0) {
                        waitpid(id3, NULL, 0);
                        close(fd1[0]);
                        close(fd1[1]);

                        close(fd2[1]);
                        dup2(fd2[0], 0);
                        close(fd2[0]);

                        int till = argc;
                        if (redirect) {
                            till = redirect_to - 1;
                            if(redirect == 1){
                                if ( argc <= redirect_to) {
                                    printf("\033[0;31merror: \033[0m");
                                    printf("Missing file name after `>`\n");
                                    printf("(Exit code 2)\n");
                                    exit(2);
                                } else {
                                    FILE *fd = fopen(argv[redirect_to], "w");
                                    int fout = fileno(fd);
                                    dup2(fout, 1);
                                    fclose(fd);
                                }
                            }
                            if(redirect == 2){
                                if ( argc <= redirect_to) {
                                    printf("\033[0;31merror: \033[0m");
                                    printf("Missing file name after `>>`\n");
                                    printf("(Exit code 3)\n");
                                    exit(3);
                                } else {
                                    FILE *fd = fopen(argv[redirect_to], "a");
                                    int fout = fileno(fd);
                                    dup2(fout, 1);
                                    fclose(fd);
                                }
                            }
                        }

                        char *argv4[10];
                        int i, k;
                        for (i = pipes[1] + 1, k = 0; i < till; ++i, ++k) {
                            argv4[k] = argv[i];
                        }
                        argv4[k] = NULL;
                        execvp(argv4[0], argv4);
                    } else {
```

```
                            close(fd1[0]);
                            close(fd1[1]);
                            close(fd2[1]);
                            close(fd2[0]);
                            waitpid(id4, NULL, 0);
                        }
                    }
                }
            } else if (redirect == 1) {
                if ( argc <= redirect_to) {
                    exit(2);
                } else {
                    char *argv2[10];
                    int i;
                    for(i = 0; i < redirect_to - 1; i++){
                        argv2[i] = argv[i];
                    }
                    argv2[i] = NULL;
                    FILE *fd = fopen(argv[redirect_to], "w");
                    int fout = fileno(fd);
                    dup2(fout, 1);
                    fclose(fd);
                    execvp(argv2[0], argv2);
                }
            } else if (redirect == 2) {
                if ( argc <= redirect_to) {
                    exit(3);
                } else {
                    char *argv2[10];
                    int i;
                    for(i = 0; i < redirect_to - 1; i++){
                        argv2[i] = argv[i];
                    }
                    argv2[i] = NULL;
                    FILE *fd = fopen(argv[redirect_to], "a");
                    int fout = fileno(fd);
                    dup2(fout, 1);
                    fclose(fd);
                    execvp(argv2[0], argv2);
                }
            }
            wait(NULL);
            exit(0);
        }
    }
}
```