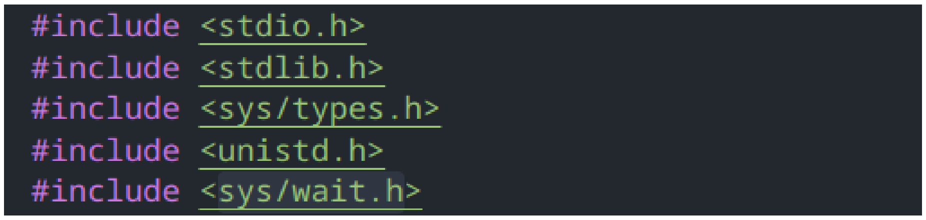
OS-Assignment-1(write-up)

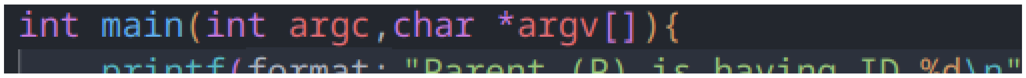
Ans 1 - 1)

1)In this part, first we include all the necessary libraries such as <stdio.h>, <stdlib.h> ,<sys/types.h>,<unistd.h> and <sys/wait.h>



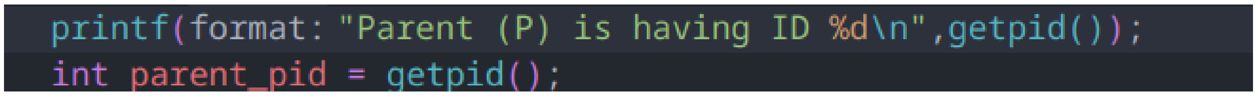
```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
```

2)Then , we defining the main function in which we take argc and char* argv[] as arguments.



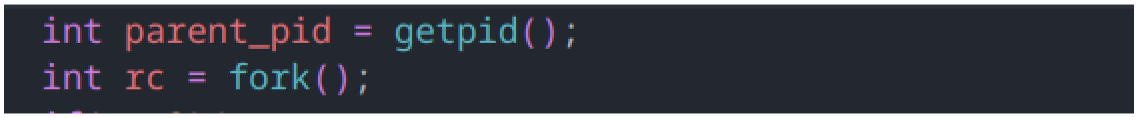
```
int main(int argc,char *argv[]){
    printf(format: "Parent (P) is having ID %d\n"
```

3)We then print the first statement (A) which gives us the parent's pid
now we store the parent's pid in the form of integer in the variable parent_pid



```
  printf(format: "Parent (P) is having ID %d\n",getpid());
  int parent_pid = getpid();
```

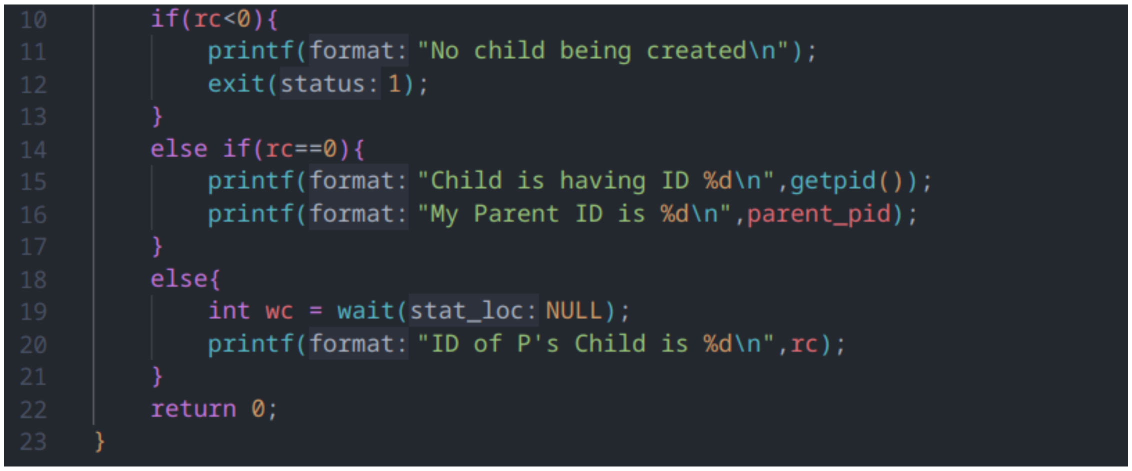
4)We then call fork system call to create a new child of the process and store the child's pid in variable rc



```
 int parent_pid = getpid();
 int rc = fork();
```

5)Now, we give 3 conditions to the , that is
if rc<0 , then no child is created because rc will be > 0 for parent process (as child must have some pid) else it would be 0 for child process and child doesn't have any child(so it results in rc value of 0)

6)So, to have the order of A,C,D,B we make the parent wait for the child using the wait() system call
So, the process goes into rc>0 condition and is directed to the child's process that is when rc = 0 condition

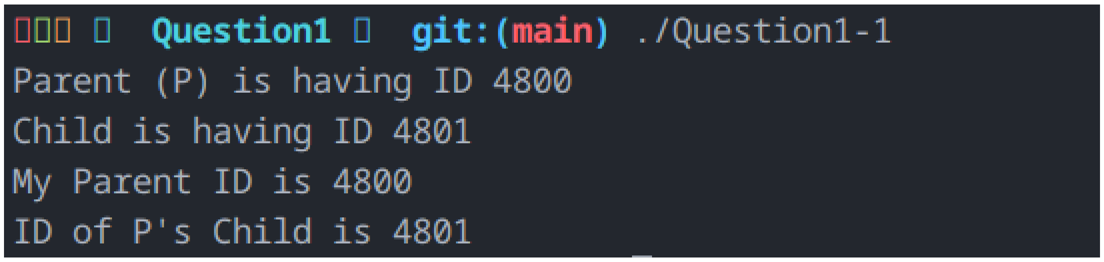


```c
    if(rc<0){
        printf(format: "No child being created\n");
        exit(status: 1);
    }
    else if(rc==0){
        printf(format: "Child is having ID %d\n",getpid());
        printf(format: "My Parent ID is %d\n",parent_pid);
    }
    else{
        int wc = wait(stat_loc: NULL);
        printf(format: "ID of P's Child is %d\n",rc);
    }
    return 0;
}
```

7)Now, the child prints the statement C by calling getpid() function in it and then prints the statement D by the parent_pid value which we stored at the starting of the program

8)Now, the child process terminates(because it has ended) and now the parent process starts executing , now the process starts after the wait call

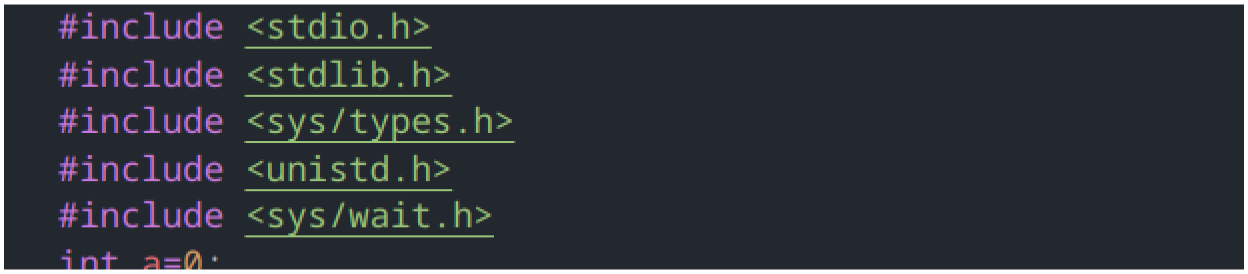
9)Now, the statement B is printed and we get the statements printed in the order A,C,D,B



```
Question1  git:(main) ./Question1-1
Parent (P) is having ID 4800
Child is having ID 4801
My Parent ID is 4800
ID of P's Child is 4801
```

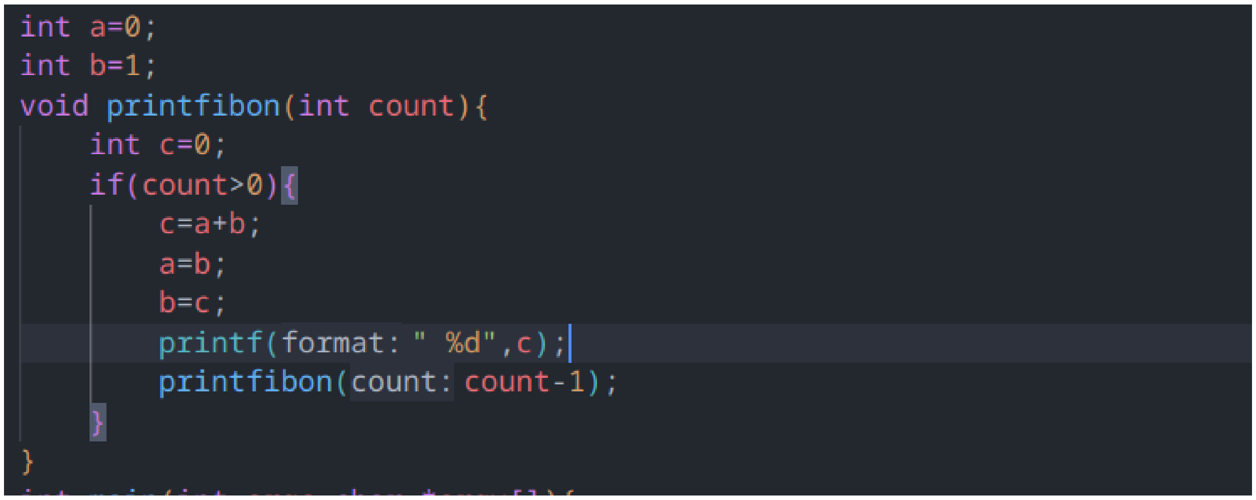
Ans (1 -2)

1) In this too, we first take the necessary c libraries and make the function which prints the first n elements of the fibonacci series .



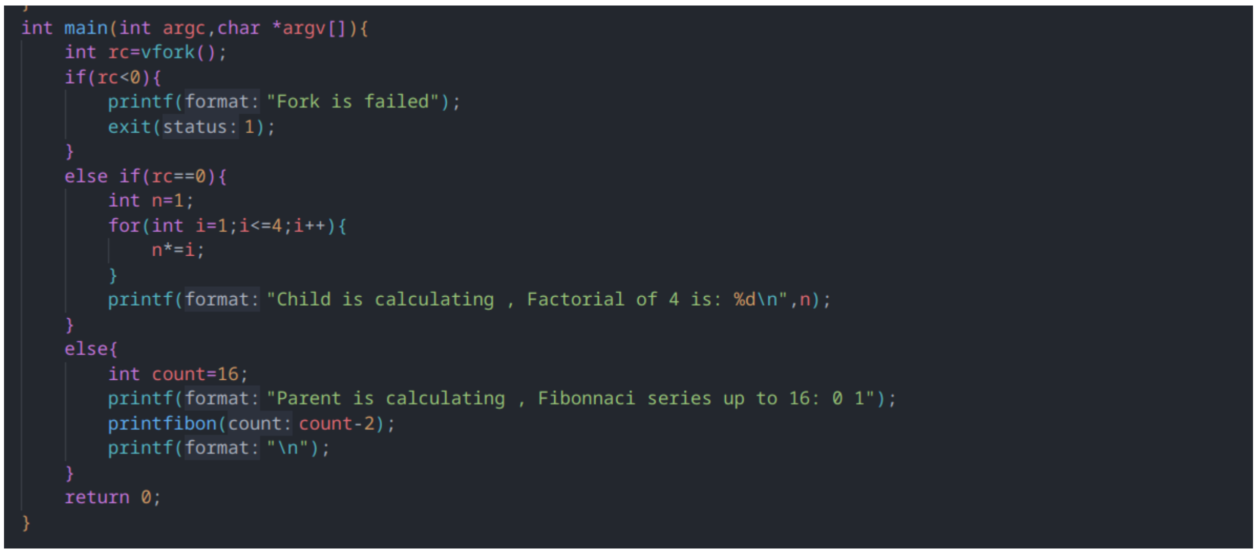
```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
int a=0;
```

2) Then , we use vfork() to make the child and vfork() terminates the parent till the child process is executed.



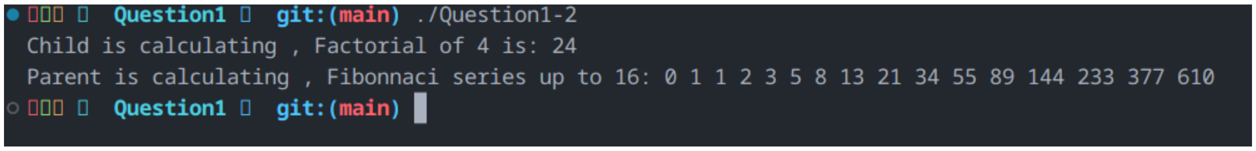
```
int a=0;
int b=1;
void printfibon(int count){
    int c=0;
    if(count>0){
        c=a+b;
        a=b;
        b=c;
        printf(format: " %d",c);
        printfibon(count: count-1);
    }
}
```

3) So, making the use of vfork() functionality , we can print the factorial of 4 in the child process that is for which rc == 0 and call the function which prints the first 16 elements of the fibonacci series in the parent function .



```
int main(int argc,char *argv[]){
    int rc=vfork();
    if(rc<0){
        printf(format: "Fork is failed");
        exit(status: 1);
    }
    else if(rc==0){
        int n=1;
        for(int i=1;i<=4;i++){
            n*=i;
        }
        printf(format: "Child is calculating , Factorial of 4 is: %d\n",n);
    }
    else{
        int count=16;
        printf(format: "Parent is calculating , Fibonnaci series up to 16: 0 1");
        printfibon(count: count-2);
        printf(format: "\n");
    }
    return 0;
}
```

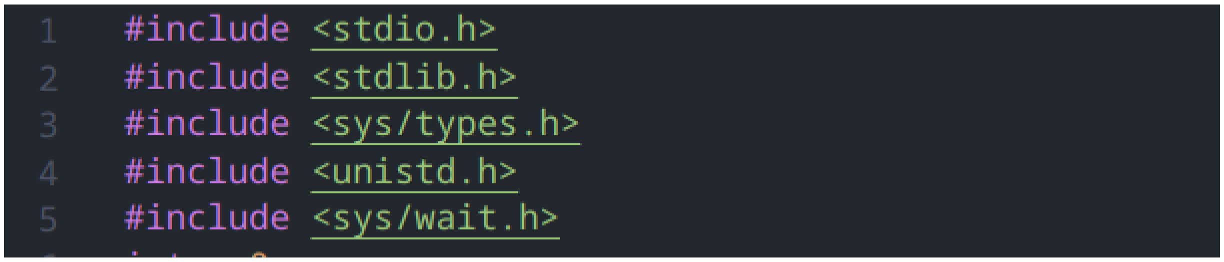
4) This will print the child process first and then the parent process.



```
● □□□ □  Question1 □  git:(main) ./Question1-2
  Child is calculating , Factorial of 4 is: 24
  Parent is calculating , Fibonnaci series up to 16: 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610
○ □□□ □  Question1 □  git:(main)
```

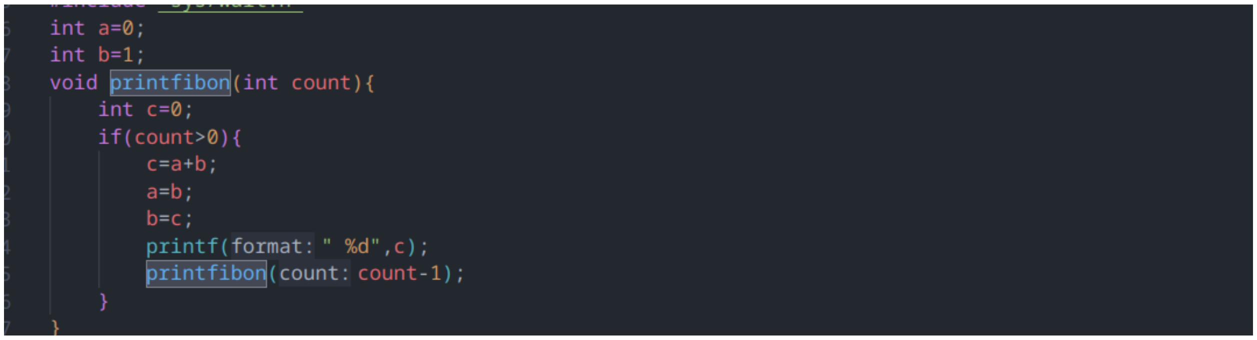
Ans (1-bonus)

1) In this too, we first include the necessary c libraries.



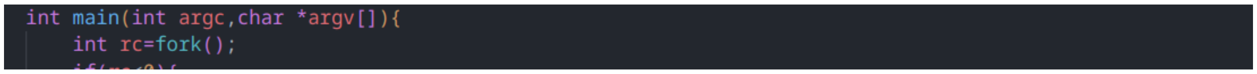
```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
```

2) Then similarly like in question (1-2) part we make the functions in this part too.



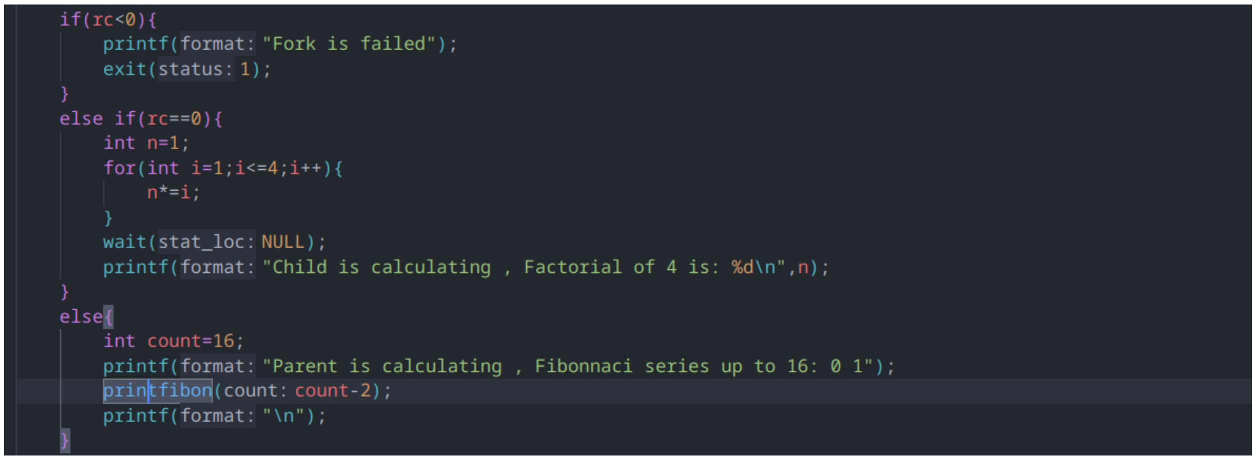
```c
int a=0;
int b=1;
void printfibon(int count){
    int c=0;
    if(count>0){
        c=a+b;
        a=b;
        b=c;
        printf(format: " %d",c);
        printfibon(count: count-1);
    }
}
```

3) We then call the fork() function here[it was mentioned that we can use either fork/vfork any of the two for this part]



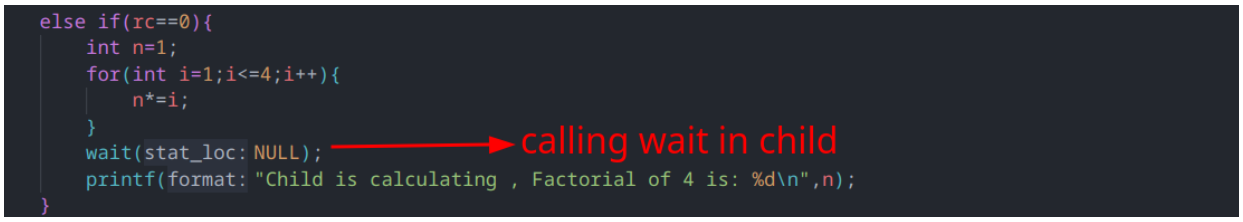
```c
int main(int argc,char *argv[]){
    int rc=fork();
```

4) Now, in the we make count the factorial of 4 in the child process[rc == 0] and then use the fibonacci function in the parent process to compute the first 16 elements of the fibonacci series. [rc>0].



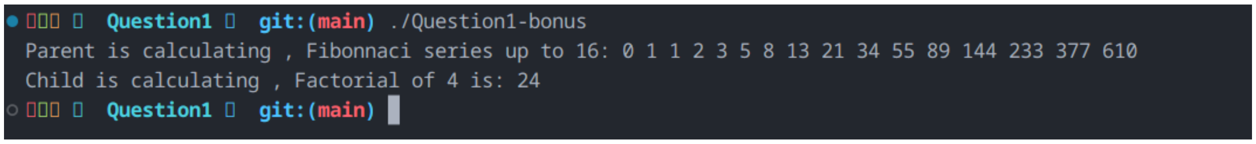
```c
    if(rc<0){
        printf(format: "Fork is failed");
        exit(status: 1);
    }
    else if(rc==0){
        int n=1;
        for(int i=1;i<=4;i++){
            n*=i;
        }
        wait(stat_loc: NULL);
        printf(format: "Child is calculating , Factorial of 4 is: %d\n",n);
    }
    else{
        int count=16;
        printf(format: "Parent is calculating , Fibonnaci series up to 16: 0 1");
        printfibon(count: count-2);
        printf(format: "\n");
    }
```

5) Now , to make the child wait for the parent so that it executes first ,we can make the child wait with the help of wait() system call .



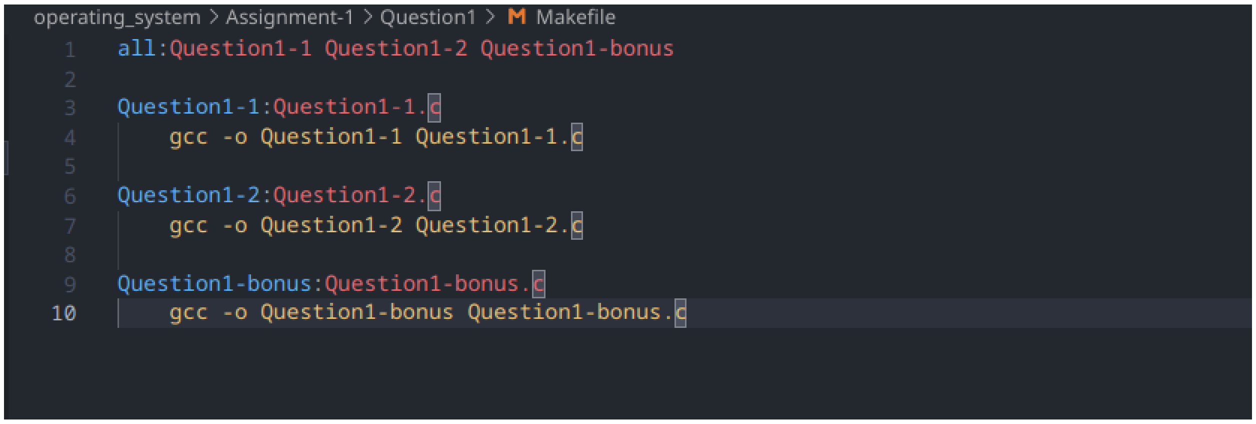
```
else if(rc==0){
    int n=1;
    for(int i=1;i<=4;i++){
        n*=i;
    }
    wait(stat_loc: NULL);
    printf(format: "Child is calculating , Factorial of 4 is: %d\n",n);
}
```

6) So, with this the child waits and we get the output that the parent is executed first and then the child is executed .



```
Question1  git:(main) ./Question1-bonus
Parent is calculating , Fibonnaci series up to 16: 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610
Child is calculating , Factorial of 4 is: 24
Question1  git:(main)
```

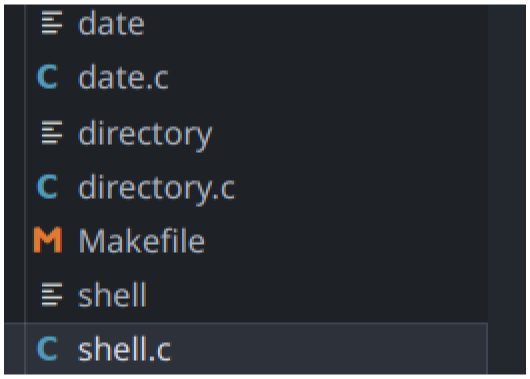
The Makefile for this question goes like :



```
all:Question1-1 Question1-2 Question1-bonus

Question1-1:Question1-1.c
	gcc -o Question1-1 Question1-1.c

Question1-2:Question1-2.c
	gcc -o Question1-2 Question1-2.c

Question1-bonus:Question1-bonus.c
	gcc -o Question1-bonus Question1-bonus.c
```

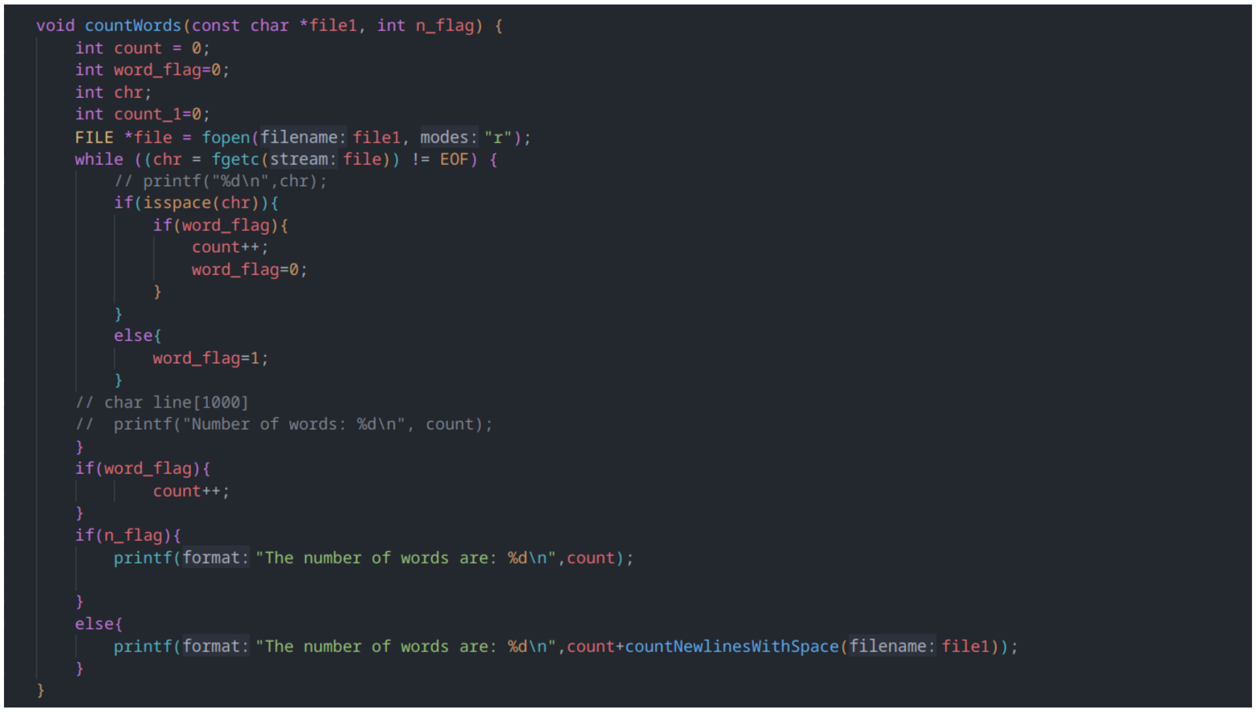
Ans 2)

Intro - this code has 3 components shell.c (which acts like a shell , that is it executes all the commands and it has word as internal command in it) , directory.c (it is an external command which gets called in shell.c using execvp) , date.c (it is also an external command which gets executed in shell.c using execvp).



```
date
date.c
directory
directory.c
Makefile
shell
shell.c
```

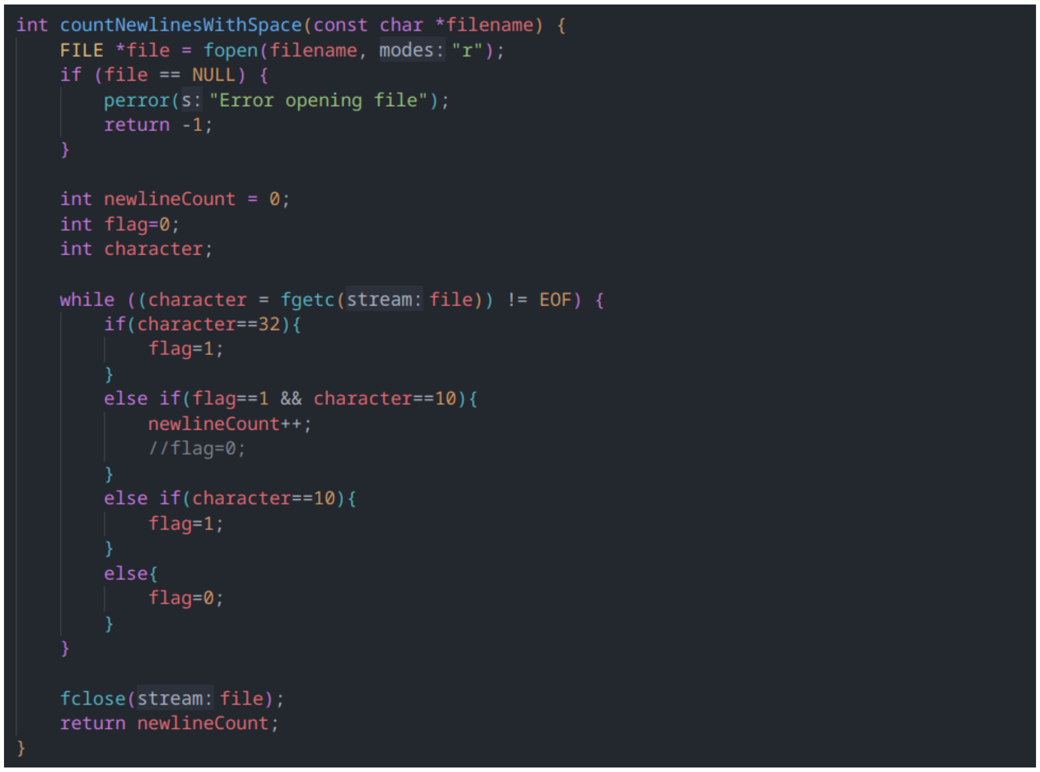
For shell.c
We include all the c libraries that are needed to make the 3 commands run
When we execute this file in the terminal using ./shell
So, lets start with word function :
To make the word function we make two functions , first that counts total number of words in the file , not including the newline characters .
So we name the function as countWords in which we read the file and count the words by the logic that we increase the count whenever we have set the flag of character as 1 and we encounter a whitespace .



```c
void countWords(const char *file1, int n_flag) {
    int count = 0;
    int word_flag=0;
    int chr;
    int count_1=0;
    FILE *file = fopen(filename: file1, modes: "r");
    while ((chr = fgetc(stream: file)) != EOF) {
        // printf("%d\n",chr);
        if(isspace(chr)){
            if(word_flag){
                count++;
                word_flag=0;
            }
        }
        else{
            word_flag=1;
        }
    // char line[1000]
    //  printf("Number of words: %d\n", count);
    }
    if(word_flag){
            count++;
    }
    if(n_flag){
        printf(format: "The number of words are: %d\n",count);

    }
    else{
        printf(format: "The number of words are: %d\n",count+countNewlinesWithSpace(filename: file1));
    }
}
```

Now , we make the function which calculates the number of newline character in the file.
So we implement the logic that whenever we have space we set the flag as 1

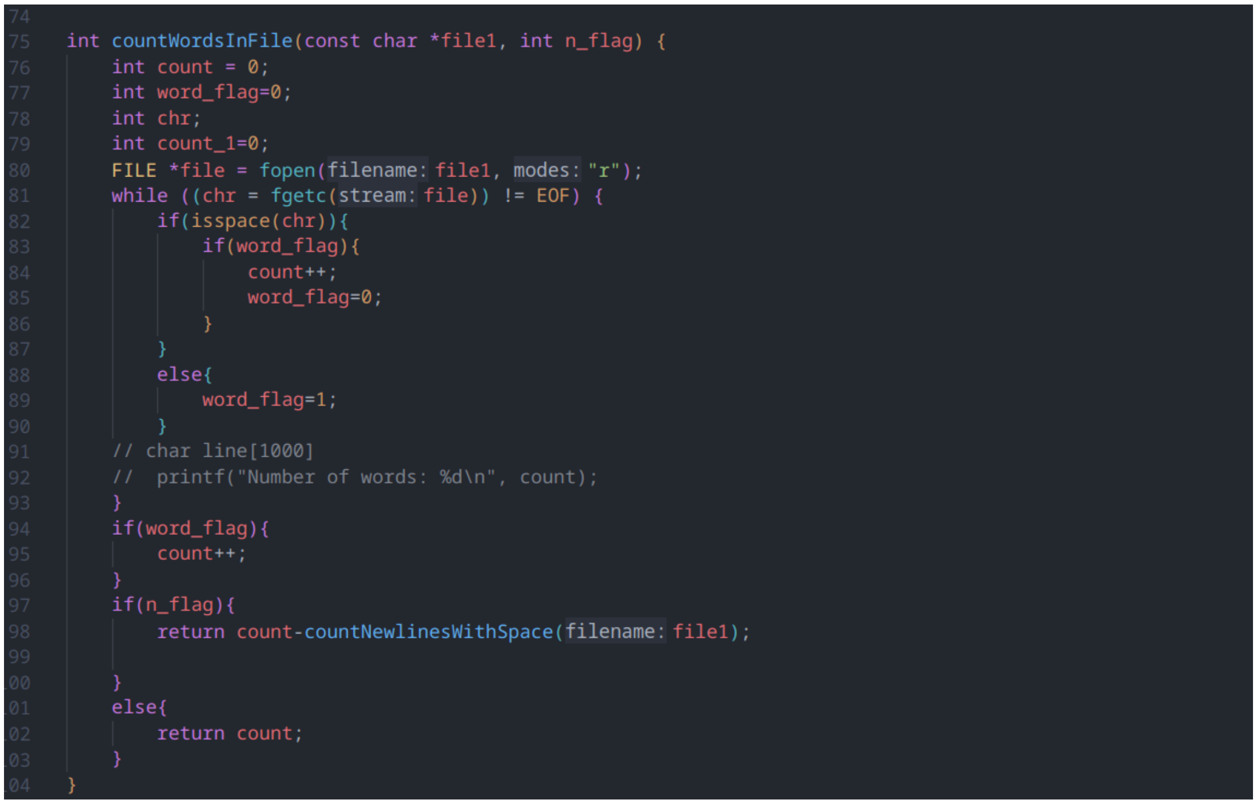
and if we encounter that \n character after that we make the counter increase by +1(that is \n character preceded by a space)



```
int countNewlinesWithSpace(const char *filename) {
    FILE *file = fopen(filename, modes: "r");
    if (file == NULL) {
        perror(s: "Error opening file");
        return -1;
    }

    int newlineCount = 0;
    int flag=0;
    int character;

    while ((character = fgetc(stream: file)) != EOF) {
        if(character==32){
            flag=1;
        }
        else if(flag==1 && character==10){
            newlineCount++;
            //flag=0;
        }
        else if(character==10){
            flag=1;
        }
        else{
            flag=0;
        }
    }

    fclose(stream: file);
    return newlineCount;
}
```

(Assumption - we are treating the newline(that is when we press enter) as a newline character and it is counted when we don't pass -n and it is not counted when -n is passed as an option)

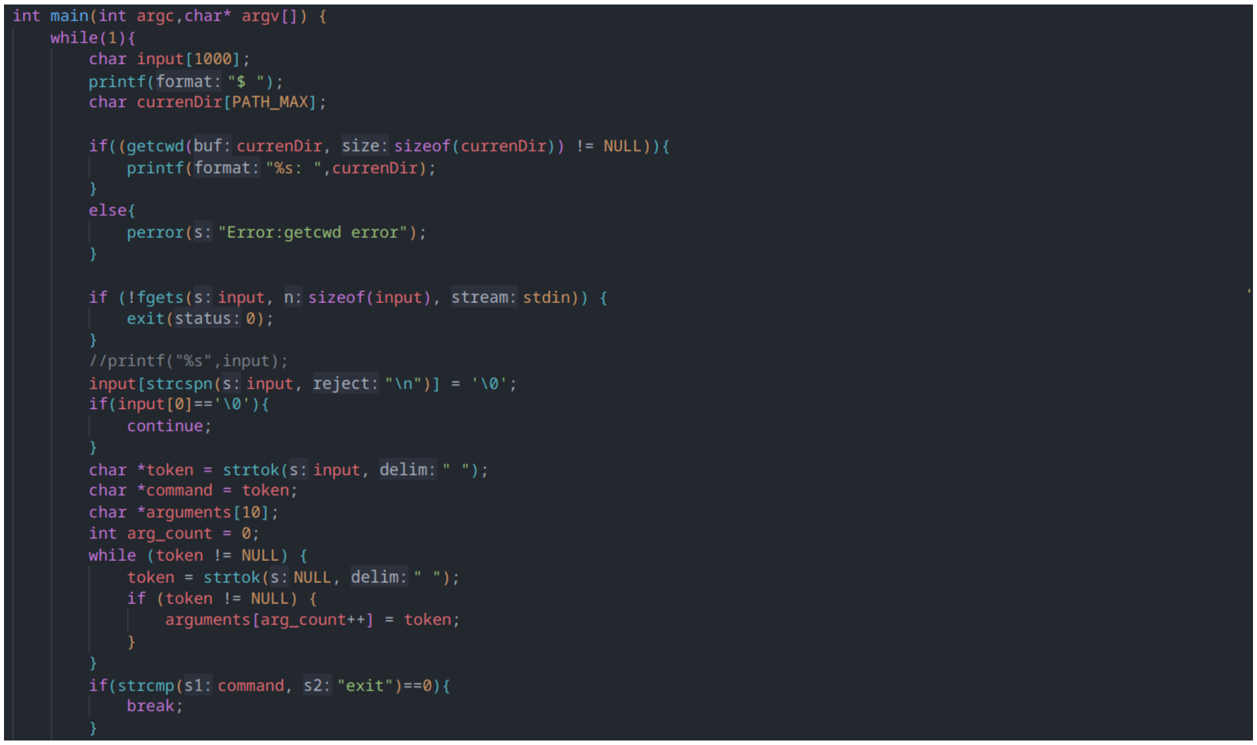
Now we make exact similar function as Countwords but the return type is different as it can be stored in an integer variable so that we can calculate the difference between the number of words in the two files (-d option).



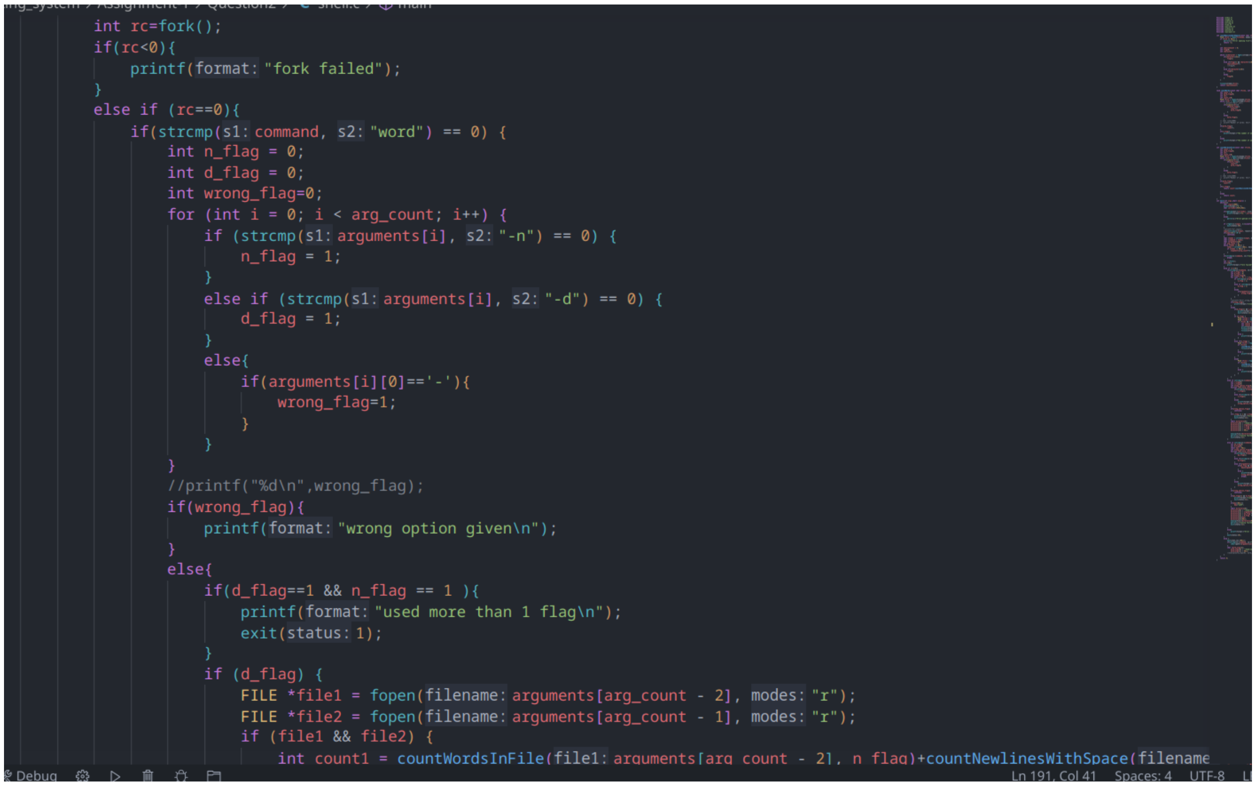
```
int countWordsInFile(const char *file1, int n_flag) {
    int count = 0;
    int word_flag=0;
    int chr;
    int count_1=0;
    FILE *file = fopen(filename: file1, modes: "r");
    while ((chr = fgetc(stream: file)) != EOF) {
        if(isspace(chr)){
            if(word_flag){
                count++;
                word_flag=0;
            }
        }
        else{
            word_flag=1;
        }
    // char line[1000]
    //  printf("Number of words: %d\n", count);
    }
    if(word_flag){
        count++;
    }
    if(n_flag){
        return count-countNewlinesWithSpace(filename: file1);

    }
    else{
        return count;
    }
}
```

<u>Taking inputs:</u>

For input we have taken a parent process and then use fork to create a child process, then we execute our all the commands in the child process and make parent wait using wait() system call so that the commands execute and in the parent we can change directory(using chdir- which will be used in dir command)
We take input with the help of fgets to read the file and use strtok to split the string and compare the string to see which command to be executed or which flags should be up for a given command.
Some snapshots of the main() function of shell.c is:

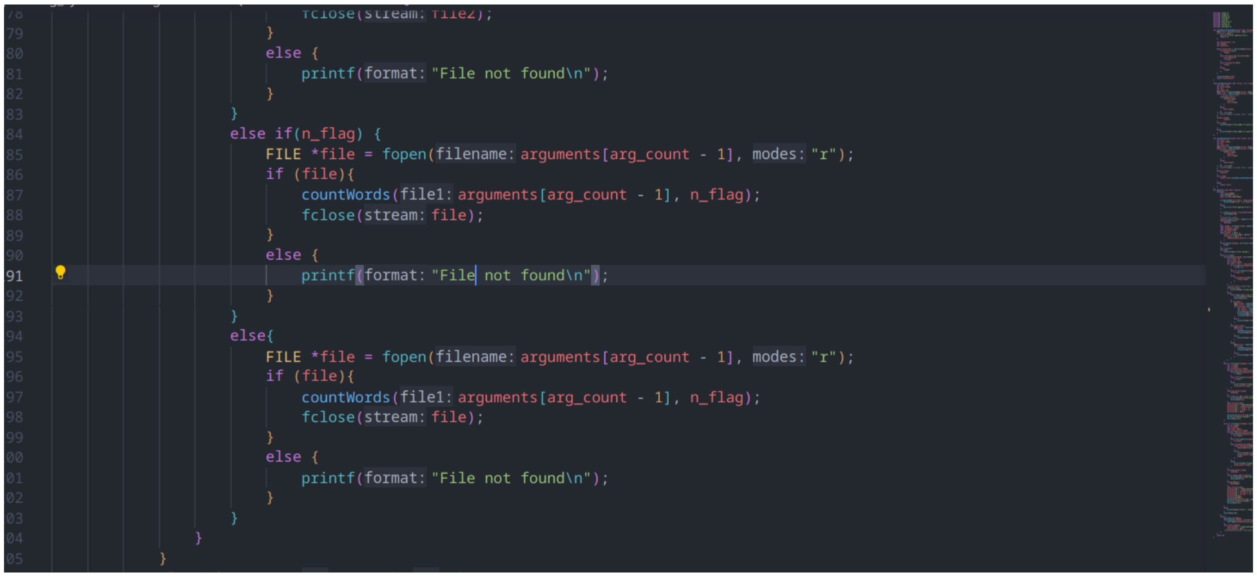


```c
int main(int argc,char* argv[]) {
    while(1){
        char input[1000];
        printf(format: "$ ");
        char currenDir[PATH_MAX];

        if((getcwd(buf: currenDir, size: sizeof(currenDir)) != NULL)){
            printf(format: "%s: ",currenDir);
        }
        else{
            perror(s: "Error:getcwd error");
        }

        if (!fgets(s: input, n: sizeof(input), stream: stdin)) {
            exit(status: 0);
        }
        //printf("%s",input);
        input[strcspn(s: input, reject: "\n")] = '\0';
        if(input[0]=='\0'){
            continue;
        }
        char *token = strtok(s: input, delim: " ");
        char *command = token;
        char *arguments[10];
        int arg_count = 0;
        while (token != NULL) {
            token = strtok(s: NULL, delim: " ");
            if (token != NULL) {
                arguments[arg_count++] = token;
            }
        }
        if(strcmp(s1: command, s2: "exit")==0){
            break;
        }
```



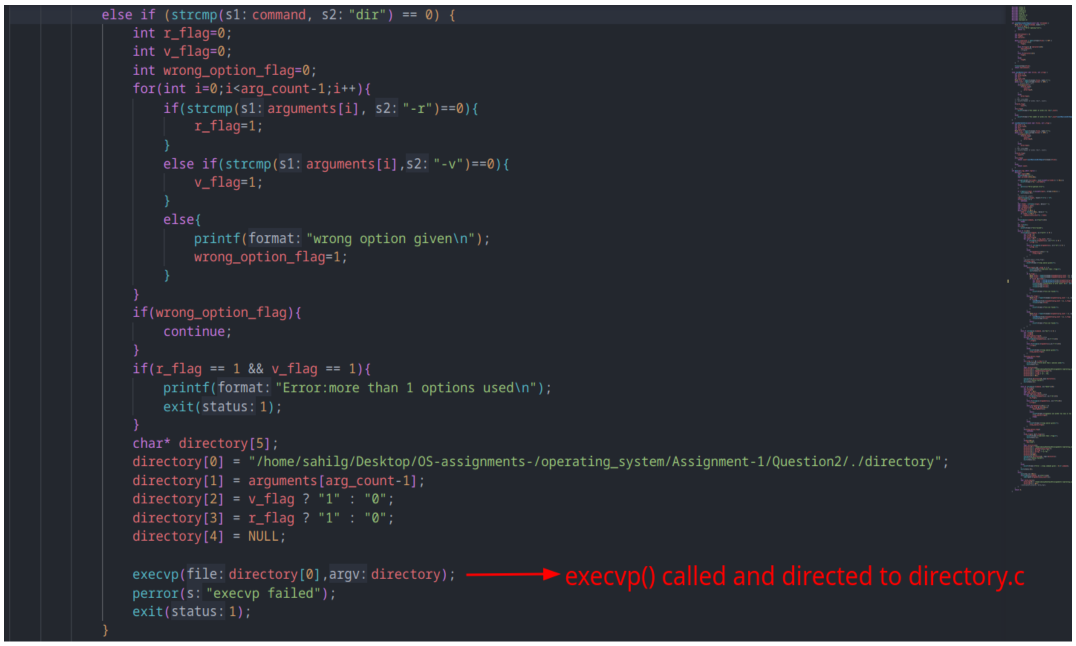
```c
        int rc=fork();
        if(rc<0){
            printf(format: "fork failed");
        }
        else if (rc==0){
            if(strcmp(s1: command, s2: "word") == 0) {
                int n_flag = 0;
                int d_flag = 0;
                int wrong_flag=0;
                for (int i = 0; i < arg_count; i++) {
                    if (strcmp(s1: arguments[i], s2: "-n") == 0) {
                        n_flag = 1;
                    }
                    else if (strcmp(s1: arguments[i], s2: "-d") == 0) {
                        d_flag = 1;
                    }
                    else{
                        if(arguments[i][0]=='-'){
                            wrong_flag=1;
                        }
                    }
                }
                //printf("%d\n",wrong_flag);
                if(wrong_flag){
                    printf(format: "wrong option given\n");
                }
                else{
                    if(d_flag==1 && n_flag == 1 ){
                        printf(format: "used more than 1 flag\n");
                        exit(status: 1);
                    }
                    if (d_flag) {
                        FILE *file1 = fopen(filename: arguments[arg_count - 2], modes: "r");
                        FILE *file2 = fopen(filename: arguments[arg_count - 1], modes: "r");
                        if (file1 && file2) {
                            int count1 = countWordsInFile(file1: arguments[arg_count - 2], n_flag)+countNewlinesWithSpace(filename
```

```
Debug                                                    Ln 191, Col 41   Spaces: 4   UTF-8
```



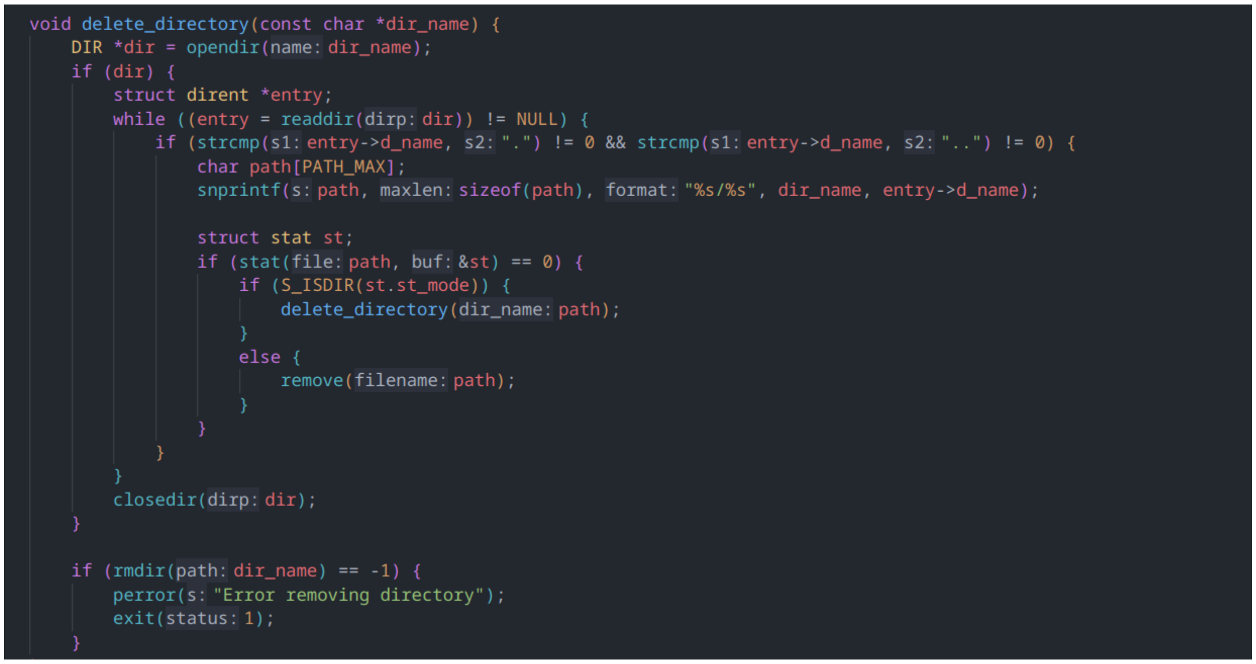
```
                        fclose(stream: file2);
                    }
                    else {
                        printf(format: "File not found\n");
                    }
                }
                else if(n_flag) {
                    FILE *file = fopen(filename: arguments[arg_count - 1], modes: "r");
                    if (file){
                        countWords(file1: arguments[arg_count - 1], n_flag);
                        fclose(stream: file);
                    }
                    else {
                        printf(format: "File not found\n");
                    }
                }
                else{
                    FILE *file = fopen(filename: arguments[arg_count - 1], modes: "r");
                    if (file){
                        countWords(file1: arguments[arg_count - 1], n_flag);
                        fclose(stream: file);
                    }
                    else {
                        printf(format: "File not found\n");
                    }
                }
            }
        }
```

Now for the two external commands - dir and date
We have created two external files dir.c and date.c which run in the child process using execvp() command . we give the arguments of file and argv in the execvp() system call so that it can be directed to the directory.c and date.c file .



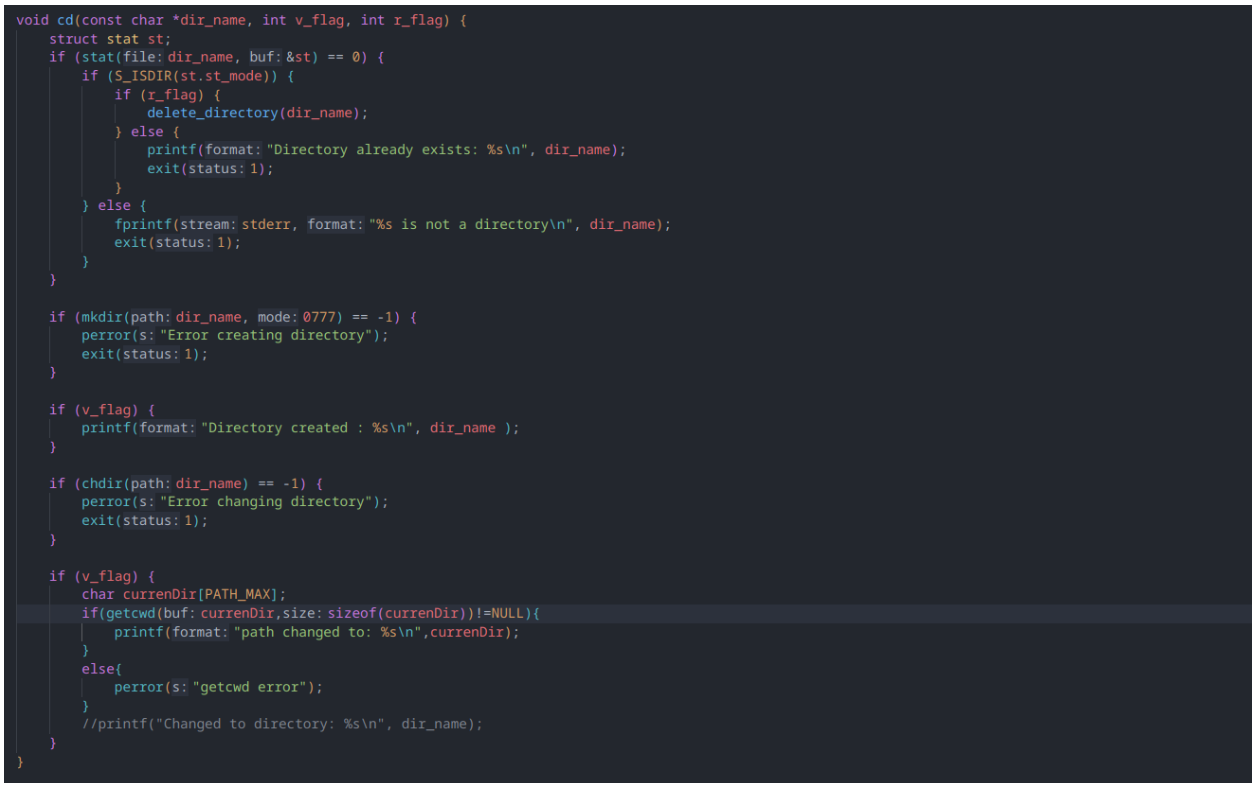
```
else if (strcmp(s1: command, s2: "dir") == 0) {
    int r_flag=0;
    int v_flag=0;
    int wrong_option_flag=0;
    for(int i=0;i<arg_count-1;i++){
        if(strcmp(s1: arguments[i], s2: "-r")==0){
            r_flag=1;
        }
        else if(strcmp(s1: arguments[i],s2: "-v")==0){
            v_flag=1;
        }
        else{
            printf(format: "wrong option given\n");
            wrong_option_flag=1;
        }
    }
    if(wrong_option_flag){
        continue;
    }
    if(r_flag == 1 && v_flag == 1){
        printf(format: "Error:more than 1 options used\n");
        exit(status: 1);
    }
    char* directory[5];
    directory[0] = "/home/sahilg/Desktop/OS-assignments-/operating_system/Assignment-1/Question2/./directory";
    directory[1] = arguments[arg_count-1];
    directory[2] = v_flag ? "1" : "0";
    directory[3] = r_flag ? "1" : "0";
    directory[4] = NULL;

    execvp(file: directory[0],argv: directory);
    perror(s: "execvp failed");
    exit(status: 1);
}
```

Now , in directory.c file we are making 2 functions , first delete_directory which deletes the contents of the directory when -r option is used , this recursively searches for the folder and files in the directory and deletes them ,



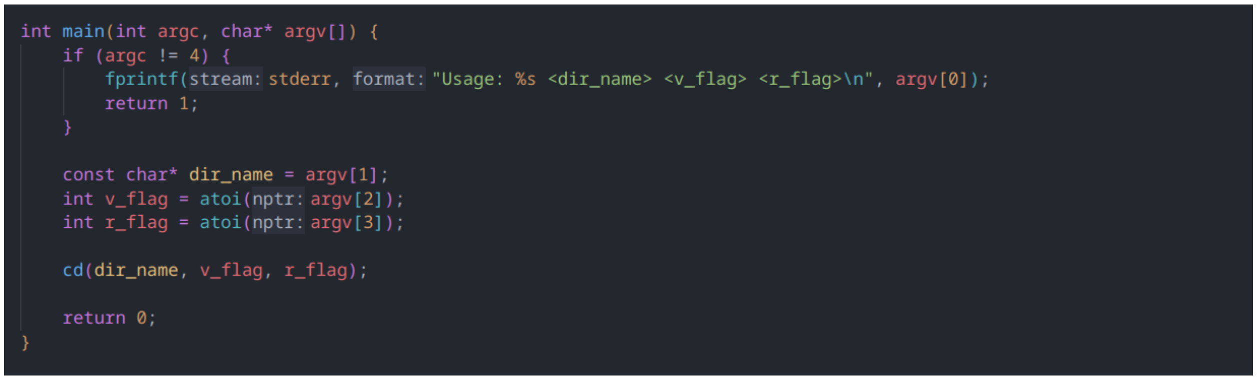
```c
void delete_directory(const char *dir_name) {
    DIR *dir = opendir(name: dir_name);
    if (dir) {
        struct dirent *entry;
        while ((entry = readdir(dirp: dir)) != NULL) {
            if (strcmp(s1: entry->d_name, s2: ".") != 0 && strcmp(s1: entry->d_name, s2: "..") != 0) {
                char path[PATH_MAX];
                snprintf(s: path, maxlen: sizeof(path), format: "%s/%s", dir_name, entry->d_name);

                struct stat st;
                if (stat(file: path, buf: &st) == 0) {
                    if (S_ISDIR(st.st_mode)) {
                        delete_directory(dir_name: path);
                    }
                    else {
                        remove(filename: path);
                    }
                }
            }
        }
        closedir(dirp: dir);
    }

    if (rmdir(path: dir_name) == -1) {
        perror(s: "Error removing directory");
        exit(status: 1);
    }
```

Next we have the cd command , which is made for making the directory,calling delete_directory function if -r flag is up and handling the possible error we may encounter during creating the directory .



```c
void cd(const char *dir_name, int v_flag, int r_flag) {
    struct stat st;
    if (stat(file: dir_name, buf: &st) == 0) {
        if (S_ISDIR(st.st_mode)) {
            if (r_flag) {
                delete_directory(dir_name);
            } else {
                printf(format: "Directory already exists: %s\n", dir_name);
                exit(status: 1);
            }
        } else {
            fprintf(stream: stderr, format: "%s is not a directory\n", dir_name);
            exit(status: 1);
        }
    }

    if (mkdir(path: dir_name, mode: 0777) == -1) {
        perror(s: "Error creating directory");
        exit(status: 1);
    }

    if (v_flag) {
        printf(format: "Directory created : %s\n", dir_name );
    }

    if (chdir(path: dir_name) == -1) {
        perror(s: "Error changing directory");
        exit(status: 1);
    }

    if (v_flag) {
        char currenDir[PATH_MAX];
        if(getcwd(buf: currenDir,size: sizeof(currenDir))!=NULL){
            printf(format: "path changed to: %s\n",currenDir);
        }
        else{
            perror(s: "getcwd error");
        }
        //printf("Changed to directory: %s\n", dir_name);
    }
}
```

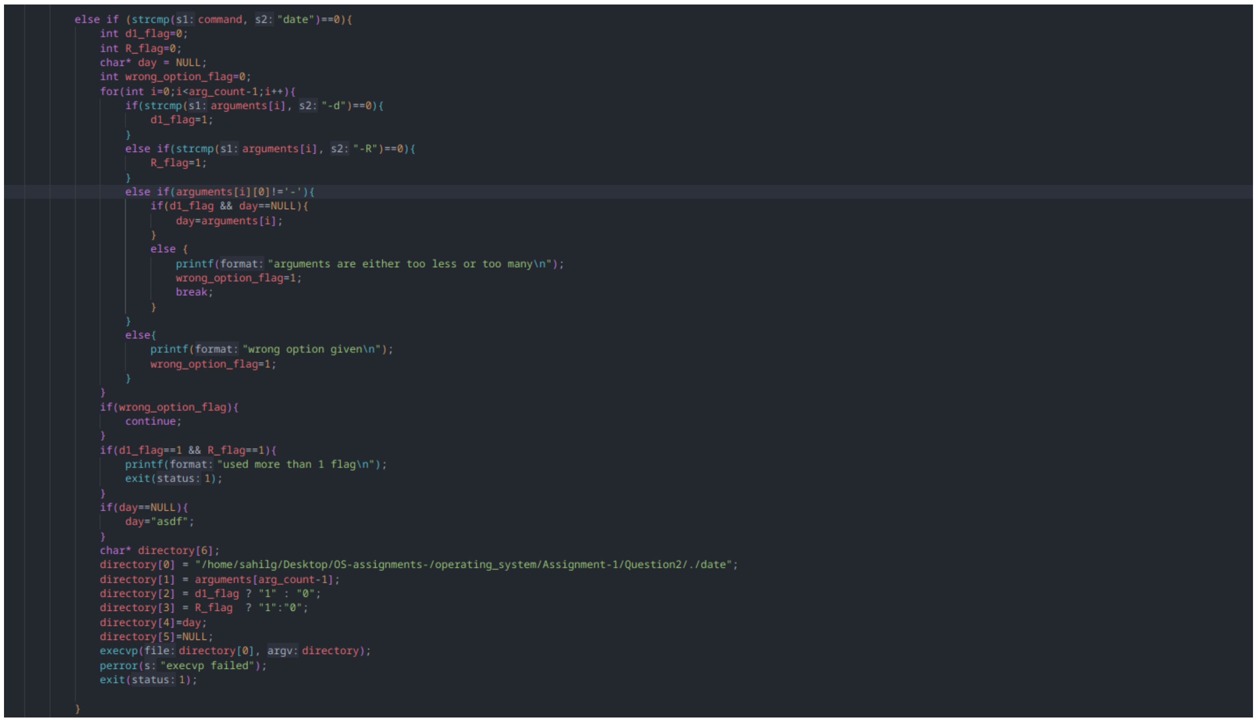
Then in the int main() function of this file we give the integer argc and pointer to char array which stores the necessary input like file name , option(-v and -r) and calls the cd function .



```c
int main(int argc, char* argv[]) {
    if (argc != 4) {
        fprintf(stream: stderr, format: "Usage: %s <dir_name> <v_flag> <r_flag>\n", argv[0]);
        return 1;
    }

    const char* dir_name = argv[1];
    int v_flag = atoi(nptr: argv[2]);
    int r_flag = atoi(nptr: argv[3]);

    cd(dir_name, v_flag, r_flag);

    return 0;
}
```

Now for the date.c , we similarly compares that if the command is “date” we call the date.c file using execvp(similar to that of directory.c)
Here we compare the flags -d and -R so that to see the correct output format ,also if we have given -d flag we have made a char* day which holds the relative day information about the output format which is optional .

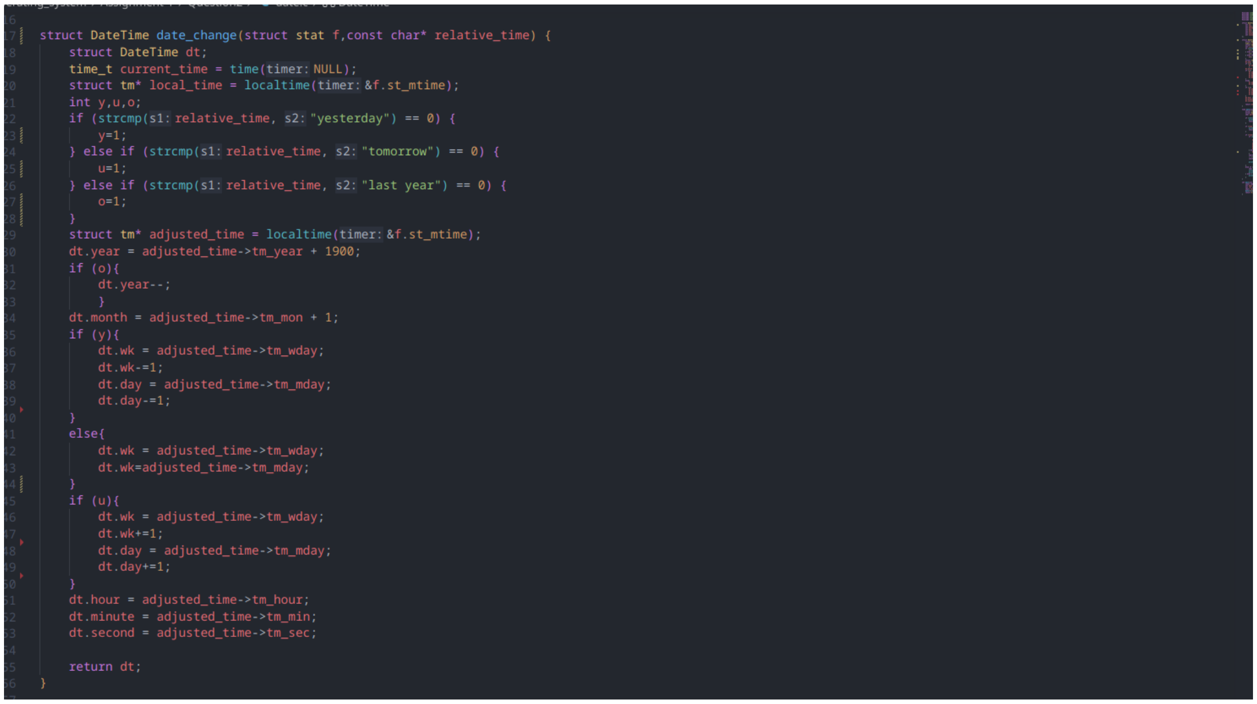
**Assumption- String can only be yesterday or tomorrow.



```c
        else if (strcmp(s1: command, s2: "date")==0){
            int d1_flag=0;
            int R_flag=0;
            char* day = NULL;
            int wrong_option_flag=0;
            for(int i=0;i<arg_count-1;i++){
                if(strcmp(s1: arguments[i], s2: "-d")==0){
                    d1_flag=1;
                }
                else if(strcmp(s1: arguments[i], s2: "-R")==0){
                    R_flag=1;
                }
                else if(arguments[i][0]!='-'){
                    if(d1_flag && day==NULL){
                        day=arguments[i];
                    }
                    else {
                        printf(format: "arguments are either too less or too many\n");
                        wrong_option_flag=1;
                        break;
                    }
                }
                else{
                    printf(format: "wrong option given\n");
                    wrong_option_flag=1;
                }
            }
            if(wrong_option_flag){
                continue;
            }
            if(d1_flag==1 && R_flag==1){
                printf(format: "used more than 1 flag\n");
                exit(status: 1);
            }
            if(day==NULL){
                day="asdf";
            }
            char* directory[6];
            directory[0] = "/home/sahilg/Desktop/OS-assignments-/operating_system/Assignment-1/Question2/./date";
            directory[1] = arguments[arg_count-1];
            directory[2] = d1_flag ? "1" : "0";
            directory[3] = R_flag  ? "1":"0";
            directory[4]=day;
            directory[5]=NULL;
            execvp(file: directory[0], argv: directory);
            perror(s: "execvp failed");
            exit(status: 1);

        }
```

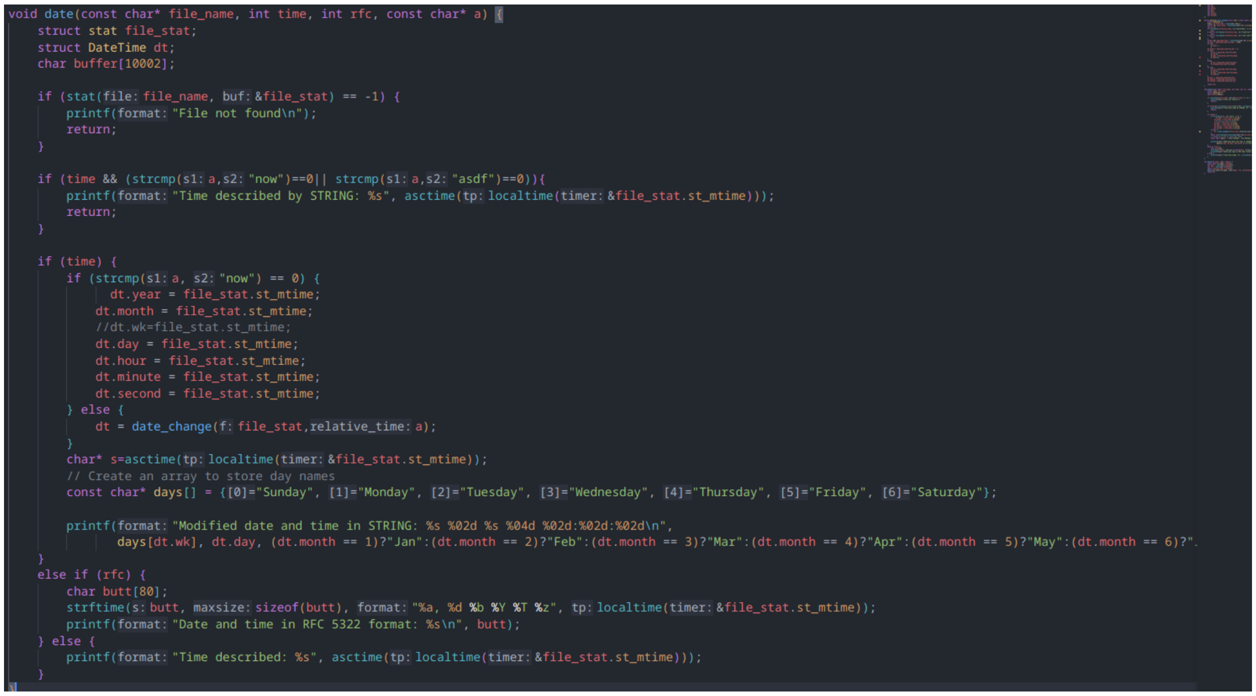
Now in the date.c , we have made a struct in which we store the all the necessary information about the date output wk, year , month ,day etc ..
Here we have created 2 functions named date_change and date .

date_change function is responsible if -d and the string is passed we have to change the date according to the date the file was last modified .
It stores the week ,day etc , so we change it accordingly by subtracting[for yesterday] or adding[tomorrow] in the date , and time here remains unchanged.



```c
struct DateTime date_change(struct stat f,const char* relative_time) {
    struct DateTime dt;
    time_t current_time = time(timer: NULL);
    struct tm* local_time = localtime(timer: &f.st_mtime);
    int y,u,o;
    if (strcmp(s1: relative_time, s2: "yesterday") == 0) {
        y=1;
    } else if (strcmp(s1: relative_time, s2: "tomorrow") == 0) {
        u=1;
    } else if (strcmp(s1: relative_time, s2: "last year") == 0) {
        o=1;
    }
    struct tm* adjusted_time = localtime(timer: &f.st_mtime);
    dt.year = adjusted_time->tm_year + 1900;
    if (o){
        dt.year--;
        }
    dt.month = adjusted_time->tm_mon + 1;
    if (y){
        dt.wk = adjusted_time->tm_wday;
        dt.wk-=1;
        dt.day = adjusted_time->tm_mday;
        dt.day-=1;
    }
    else{
        dt.wk = adjusted_time->tm_wday;
        dt.wk=adjusted_time->tm_mday;
    }
    if (u){
        dt.wk = adjusted_time->tm_wday;
        dt.wk+=1;
        dt.day = adjusted_time->tm_mday;
        dt.day+=1;
    }
    dt.hour = adjusted_time->tm_hour;
    dt.minute = adjusted_time->tm_min;
    dt.second = adjusted_time->tm_sec;

    return dt;
}
```

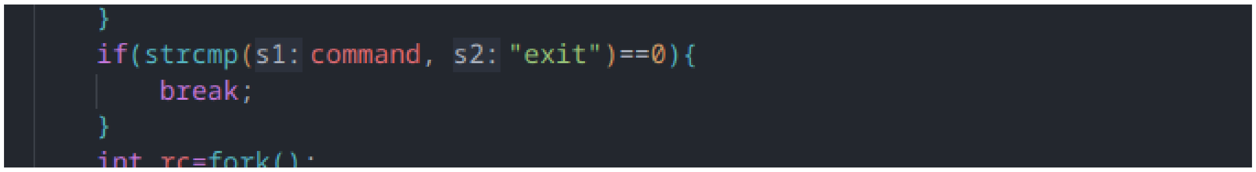
In the date function we normally print the date in the requested format using localtime function and if -R flag is up it goes in a condition to print in rc 5322 format .



```c
void date(const char* file_name, int time, int rfc, const char* a) {
    struct stat file_stat;
    struct DateTime dt;
    char buffer[10002];

    if (stat(file: file_name, buf: &file_stat) == -1) {
        printf(format: "File not found\n");
        return;
    }

    if (time && (strcmp(s1: a,s2: "now")==0|| strcmp(s1: a,s2: "asdf")==0)){
        printf(format: "Time described by STRING: %s", asctime(tp: localtime(timer: &file_stat.st_mtime)));
        return;
    }

    if (time) {
        if (strcmp(s1: a, s2: "now") == 0) {
              dt.year = file_stat.st_mtime;
            dt.month = file_stat.st_mtime;
            //dt.wk=file_stat.st_mtime;
            dt.day = file_stat.st_mtime;
            dt.hour = file_stat.st_mtime;
            dt.minute = file_stat.st_mtime;
            dt.second = file_stat.st_mtime;
        } else {
            dt = date_change(f: file_stat,relative_time: a);
        }
        char* s=asctime(tp: localtime(timer: &file_stat.st_mtime));
        // Create an array to store day names
        const char* days[] = {[0]="Sunday", [1]="Monday", [2]="Tuesday", [3]="Wednesday", [4]="Thursday", [5]="Friday", [6]="Saturday"};

        printf(format: "Modified date and time in STRING: %s %02d %s %04d %02d:%02d:%02d\n",
               days[dt.wk], dt.day, (dt.month == 1)?"Jan":(dt.month == 2)?"Feb":(dt.month == 3)?"Mar":(dt.month == 4)?"Apr":(dt.month == 5)?"May":(dt.month == 6)?".
    }
    else if (rfc) {
        char butt[80];
        strftime(s: butt, maxsize: sizeof(butt), format: "%a, %d %b %Y %T %z", tp: localtime(timer: &file_stat.st_mtime));
        printf(format: "Date and time in RFC 5322 format: %s\n", butt);
    } else {
        printf(format: "Time described: %s", asctime(tp: localtime(timer: &file_stat.st_mtime)));
    }
```

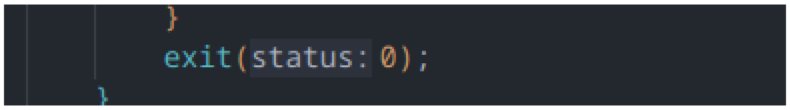
At last , in int main we have passed the necessary inputs for the files in argc and char* argv[].

Then in the else condition of the shell.c file we print “Error : wrong command given”
Also to exit the code using the exit command if we are giving exit at the start of the shell , we can write



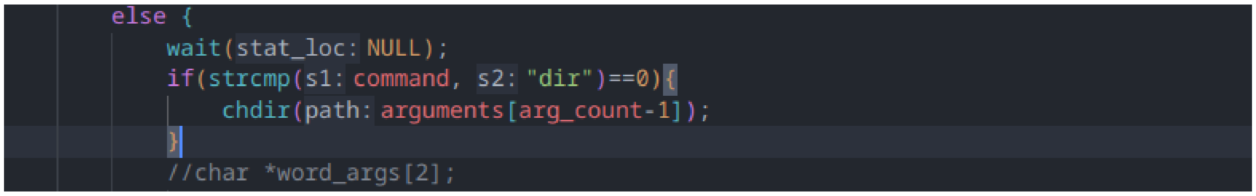
```
        }
        if(strcmp(s1: command, s2: "exit")==0){
            break;
        }
        int rc=fork();
```

in the start of this while loop[which goes till infinity]
Now if the child process is to be exited using the exit command we can pass



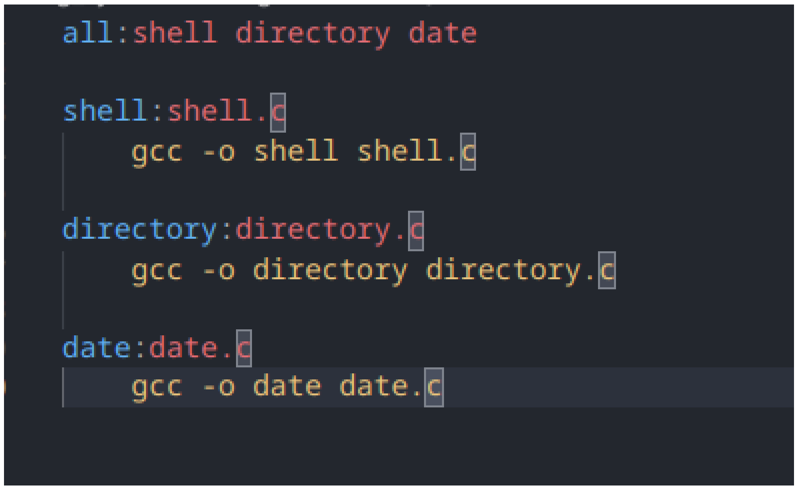
```
            }
            exit(status: 0);
        }
```

Now to change the directory , we have used chdir in the parent process



```
        else {
            wait(stat_loc: NULL);
            if(strcmp(s1: command, s2: "dir")==0){
                chdir(path: arguments[arg_count-1]);
            }
            //char *word_args[2];
```

This changes the directory whenever we are using dir command .
The Makefile of this goes like:



```
all:shell directory date

shell:shell.c
	gcc -o shell shell.c

directory:directory.c
	gcc -o directory directory.c

date:date.c
	gcc -o date date.c
```

Ans 3)
In this we used bash scripting to make a calculator that either compares, do xor or do product of the two numbers as per the input given in the input.txt in each line .
Using shebang #!/bin/bash at the starting of the bash script.

```
#!/bin/bash
```

So , first we store the path of the directory Result in which we will make a output.txt file which will have the output.txt file which will store the output for the calculation done for the input file .

```
out_directory="/home/sahilg/Desktop/OS-assignments-/operating_system/Assignment-1/Question3/Result"

out_file="$out_directory/output.txt"
```

Then in the out_file we store the path of the file output.txt which will be created not present[same goes for the Result directory which will be created if not present]
We use mkdir to make the directory if it not already present and use > to make out_file or to rewrite the file everytime we do make.

```
mkdir -p "$out_directory"

> "$out_file"
#echo "$out_file"
```

Now to read the lines of the file line by line ,we use

```
while IFS= read -r line || [ -n "$line" ]; do
```

Till we get a newline

```
if [ -n "$line" ]; then
```

Then we space separate the line using awk command which the string ,

```
        x=$(echo "$line" | awk '{print $1}')
        y=$(echo "$line" | awk '{print $2}')
        operation=$(echo "$line" | awk '{print $3}')
```

Here the 1st argument is x , 2nd argument is y and the 3rd argument is the operator .

Then , we compare operator using string comparison =, and if then operator . so we store the result in the result variable and write the result in the output.txt file using >> operator which appends the variable in the file in each line , we also give else statement that returns “No such operator” when the operator doesnt match with any of the compare, xor or product

```
        if [ "$operation" = "xor" ]; then
            result=$((y^x))
            echo "Result of xor $x $y : $result" >> "$out_file"
        elif [ "$operation" = "product" ]; then
            result=$((x * y))
            echo "Result of product $x $y : $result" >> "$out_file"
        elif [ "$operation" = "compare" ];then
            if [ "$x" -gt "$y" ]; then
                echo "Result of compare $x $y : $x" >> "$out_file"
            else
```

```
                echo "Result of compare $x $y : $y" >> "$out_file"
            fi
        else
            echo "No such operation" >> "$out_file"
        fi
```

At the end , we close the while loop with done in which we write the path of the input file .

```
done <
"/home/sahilg/Desktop/OS-assignments-/operating_system/Assignment-1/Questi
on3/input.txt"
```

So the whole code goes as :

```
#!/bin/bash
out_directory="/home/sahilg/Desktop/OS-assignments-/operating_system/Assig
nment-1/Question3/Result"

out_file="$out_directory/output.txt"

mkdir -p "$out_directory"

> "$out_file"
#echo "$out_file"
while IFS= read -r line || [ -n "$line" ]; do
   if [ -n "$line" ]; then
       x=$(echo "$line" | awk '{print $1}')
       y=$(echo "$line" | awk '{print $2}')
       operation=$(echo "$line" | awk '{print $3}')
       #echo "$operation"
       if [ "$operation" = "xor" ]; then
           result=$((y^x))
           echo "Result of xor $x $y : $result" >> "$out_file"
       elif [ "$operation" = "product" ]; then
           result=$((x * y))
           echo "Result of product $x $y : $result" >> "$out_file"
       elif [ "$operation" = "compare" ];then
           if [ "$x" -gt "$y" ]; then
               echo "Result of compare $x $y : $x" >> "$out_file"
           else
               echo "Result of compare $x $y : $y" >> "$out_file"
```

```
            fi
        else
            echo "No such operation" >> "$out_file"
        fi
    fi
done <
"/home/sahilg/Desktop/OS-assignments-/operating_system/Assignment-1/Questi
on3/input.txt"
```

The Makefile of this code goes like:

```
all:Calculator
Calculator:Calculator.sh
   sh Calculator.sh
```