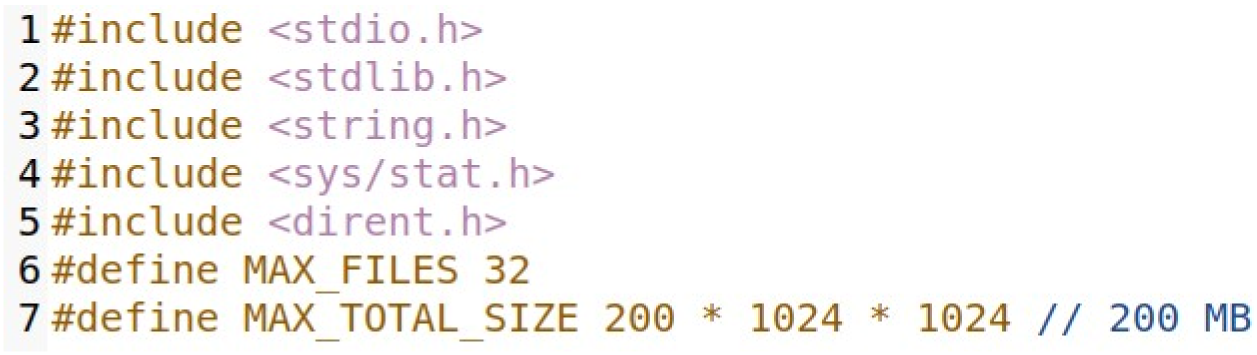
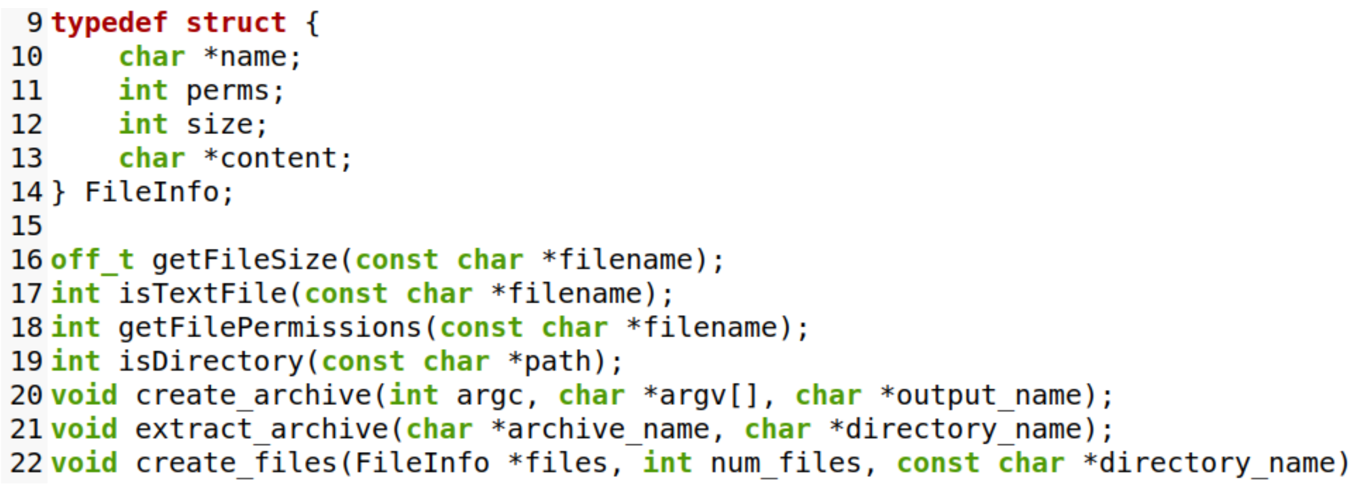
Tarsau Archiving Program

Tarsau is an archiving program works like tar or zip. But it merges only text files into a one file and extracts it on demand.
This project consist of various system calls in order to make operations on files.
In order to develop this project I had to learn few additional system calls, functions and libraries.
Here are functions that works in the program, outputs of them and couple of sentences describing why that code is used.



```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <dirent.h>
#define MAX_FILES 32
#define MAX_TOTAL_SIZE 200 * 1024 * 1024 // 200 MB
```

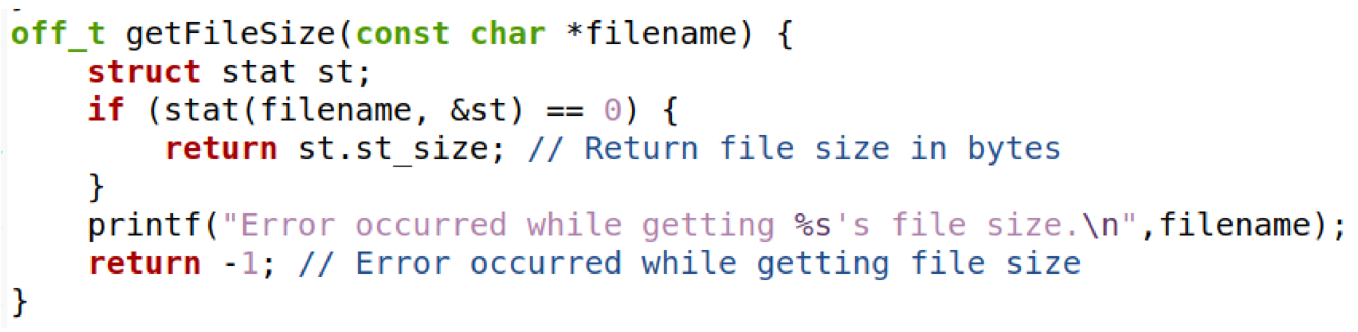
I’ve included necessary libraries for printing outputs, string operations, reading file statistics and making operations on directories. I’ve also defined MAX_FILES and MAX_TOTAL_SIZE for defining limits for the program.



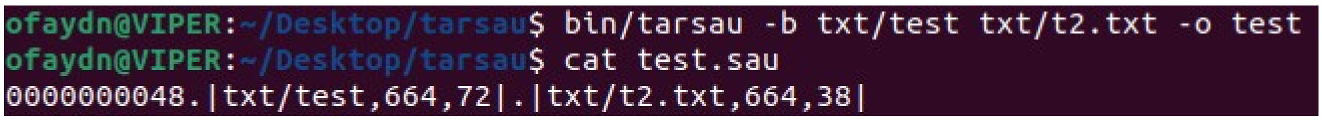
```c
typedef struct {
    char *name;
    int perms;
    int size;
    char *content;
} FileInfo;

off_t getFileSize(const char *filename);
int isTextFile(const char *filename);
int getFilePermissions(const char *filename);
int isDirectory(const char *path);
void create_archive(int argc, char *argv[], char *output_name);
void extract_archive(char *archive_name, char *directory_name);
void create_files(FileInfo *files, int num_files, const char *directory_name)
```

FileInfo struct is necessary for file operations which will contain information about text file. I will explain functions one by one.

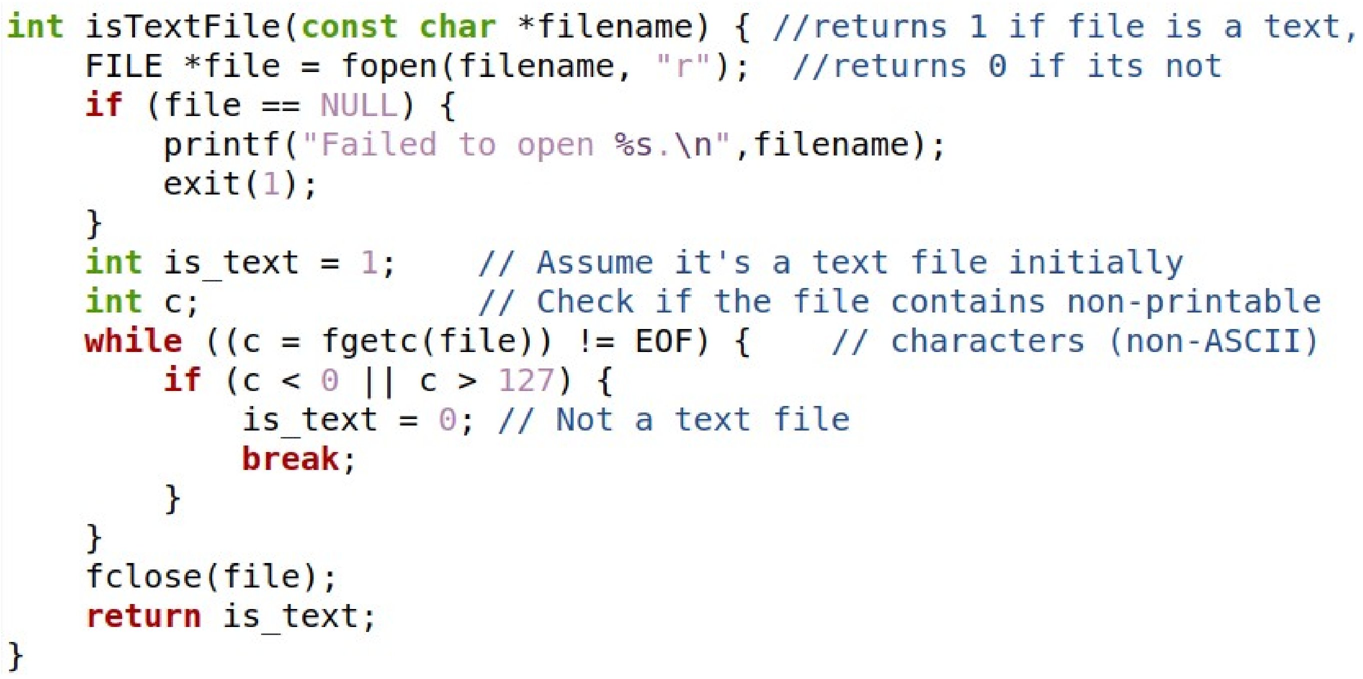


```c
off_t getFileSize(const char *filename) {
    struct stat st;
    if (stat(filename, &st) == 0) {
        return st.st_size; // Return file size in bytes
    }
    printf("Error occurred while getting %s's file size.\n",filename);
    return -1; // Error occurred while getting file size
}
```

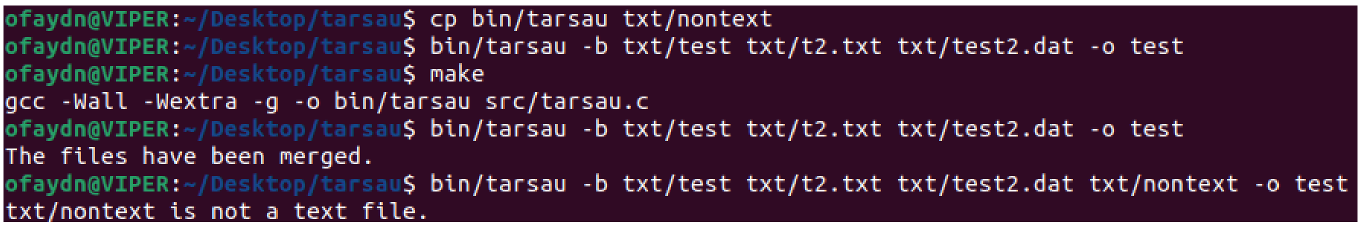


```
ofaydn@VIPER:~/Desktop/tarsau$ bin/tarsau -b txt/test txt/t2.txt -o test
ofaydn@VIPER:~/Desktop/tarsau$ cat test.sau
0000000048.|txt/test,664,72|.|txt/t2.txt,664,38|
```

getFileSize function gets file size from given file name. This function uses stat.h library. This size information is being used in checking inputs' size to not to exceed 200MBytes and creating archive.

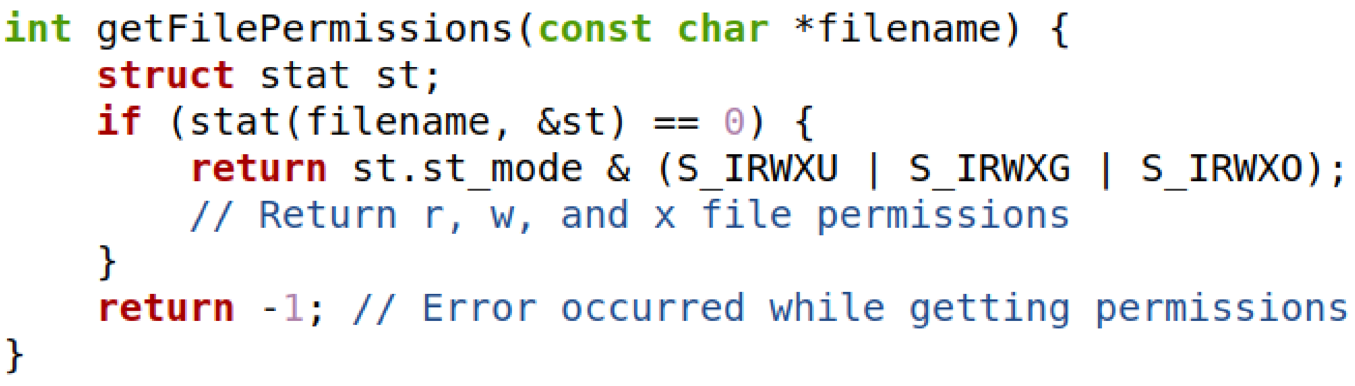


```c
int isTextFile(const char *filename) { //returns 1 if file is a text,
    FILE *file = fopen(filename, "r");  //returns 0 if its not
    if (file == NULL) {
        printf("Failed to open %s.\n",filename);
        exit(1);
    }
    int is_text = 1;     // Assume it's a text file initially
    int c;               // Check if the file contains non-printable
    while ((c = fgetc(file)) != EOF) {    // characters (non-ASCII)
        if (c < 0 || c > 127) {
            is_text = 0; // Not a text file
            break;
        }
    }
    fclose(file);
    return is_text;
}
```

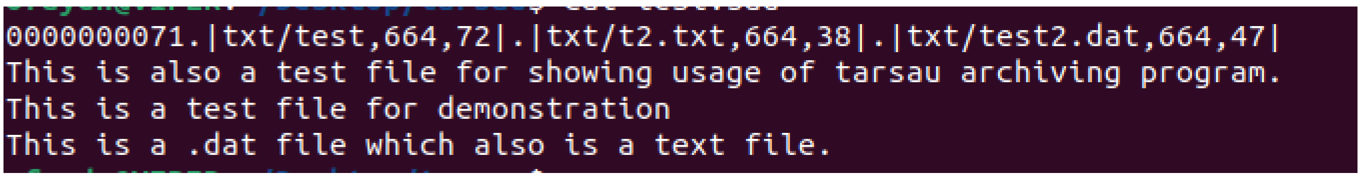


```
ofaydn@VIPER:~/Desktop/tarsau$ cp bin/tarsau txt/nontext
ofaydn@VIPER:~/Desktop/tarsau$ bin/tarsau -b txt/test txt/t2.txt txt/test2.dat -o test
ofaydn@VIPER:~/Desktop/tarsau$ make
gcc -Wall -Wextra -g -o bin/tarsau src/tarsau.c
ofaydn@VIPER:~/Desktop/tarsau$ bin/tarsau -b txt/test txt/t2.txt txt/test2.dat -o test
The files have been merged.
ofaydn@VIPER:~/Desktop/tarsau$ bin/tarsau -b txt/test txt/t2.txt txt/test2.dat txt/nontext -o test
txt/nontext is not a text file.
```

isTextFile function controls if given file is text or not. This function is crucial since we only want to archive text files. Returns 1 if given file is text, 0 otherwise.As you can see I used executable tarsau file as input and it didn't accept and returned the wrong file.

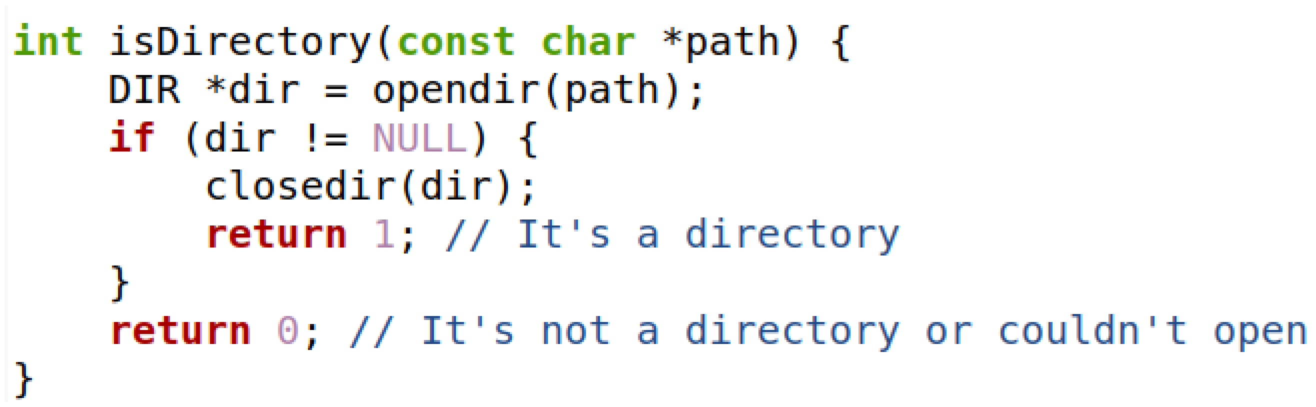


```c
int getFilePermissions(const char *filename) {
    struct stat st;
    if (stat(filename, &st) == 0) {
        return st.st_mode & (S_IRWXU | S_IRWXG | S_IRWXO);
        // Return r, w, and x file permissions
    }
    return -1; // Error occurred while getting permissions
}
```

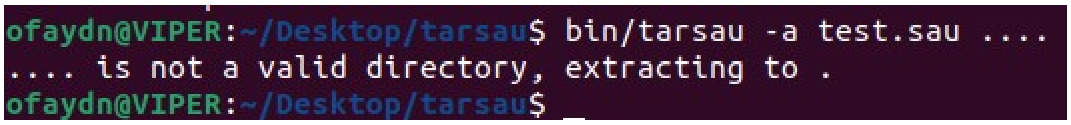


```
0000000071.|txt/test,664,72|.|txt/t2.txt,664,38|.|txt/test2.dat,664,47|
This is also a test file for showing usage of tarsau archiving program.
This is a test file for demonstration
This is a .dat file which also is a text file.
```

getFilePermissions gets permissions of given file. I had to indicate specific read, write and execute permission for proper display. Permissions is being used in archive's information section and creating file.

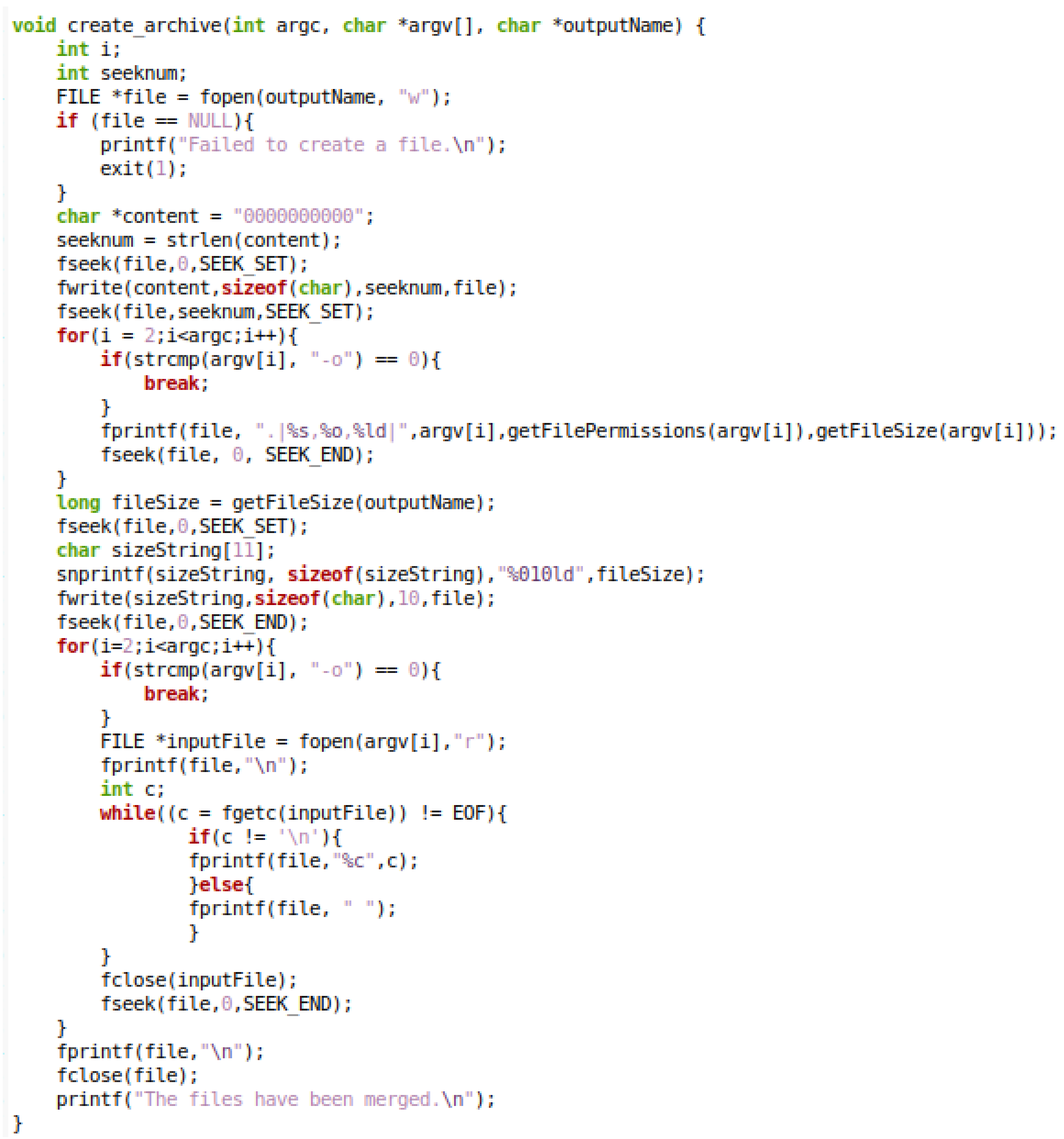


```c
int isDirectory(const char *path) {
    DIR *dir = opendir(path);
    if (dir != NULL) {
        closedir(dir);
        return 1; // It's a directory
    }
    return 0; // It's not a directory or couldn't open
}
```



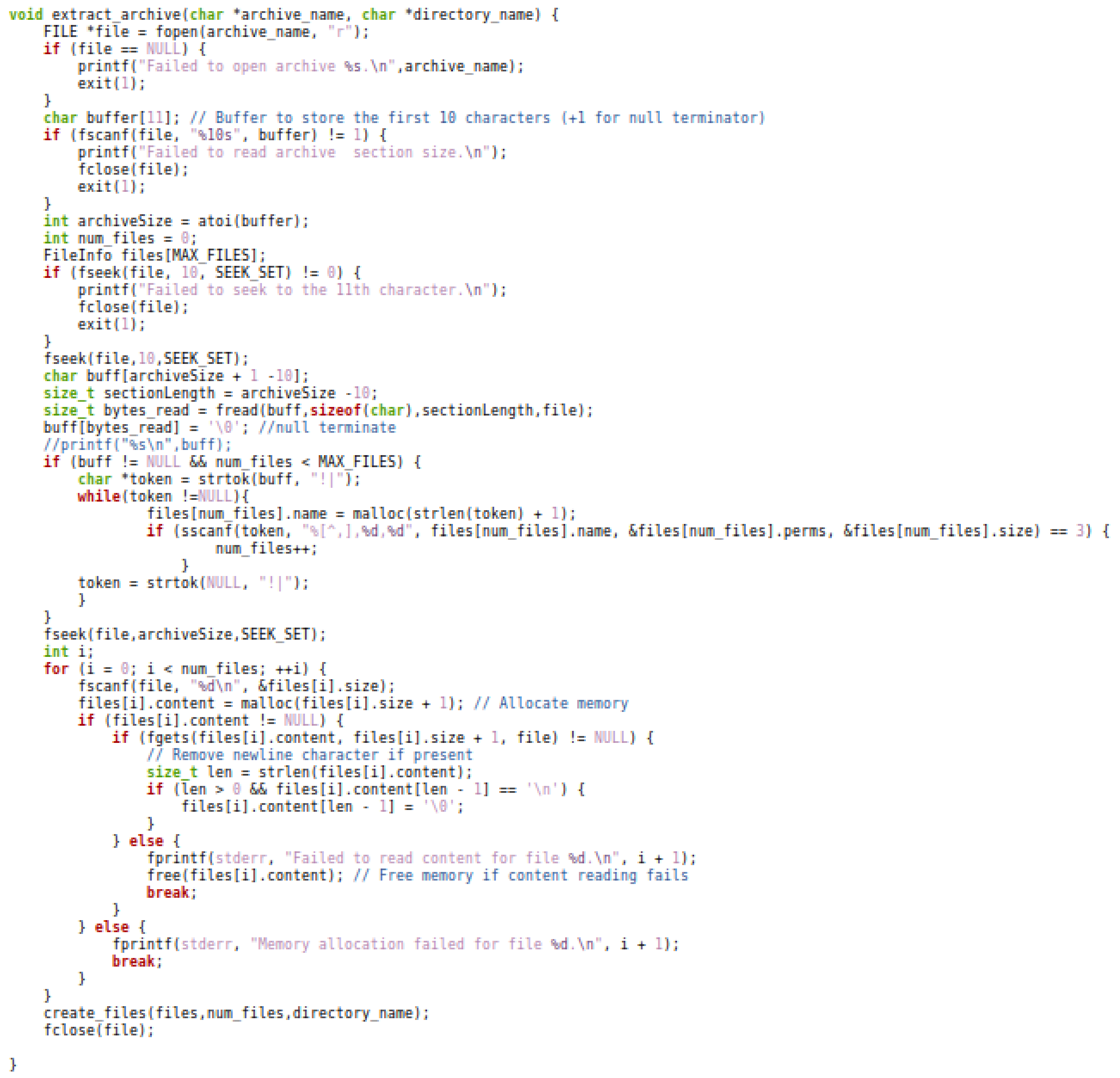
```
ofaydn@VIPER:~/Desktop/tarsau$ bin/tarsau -a test.sau ....
.... is not a valid directory, extracting to .
ofaydn@VIPER:~/Desktop/tarsau$ _
```

isDirectory function is used for checking the directory if it exists or not. This function is being used in extracting the archive in order to verify the directory.



```c
void create_archive(int argc, char *argv[], char *outputName) {
    int i;
    int seeknum;
    FILE *file = fopen(outputName, "w");
    if (file == NULL){
        printf("Failed to create a file.\n");
        exit(1);
    }
    char *content = "0000000000";
    seeknum = strlen(content);
    fseek(file,0,SEEK_SET);
    fwrite(content,sizeof(char),seeknum,file);
    fseek(file,seeknum,SEEK_SET);
    for(i = 2;i<argc;i++){
        if(strcmp(argv[i], "-o") == 0){
            break;
        }
        fprintf(file, ".|%s,%o,%ld|",argv[i],getFilePermissions(argv[i]),getFileSize(argv[i]));
        fseek(file, 0, SEEK_END);
    }
    long fileSize = getFileSize(outputName);
    fseek(file,0,SEEK_SET);
    char sizeString[11];
    snprintf(sizeString, sizeof(sizeString),"%010ld",fileSize);
    fwrite(sizeString,sizeof(char),10,file);
    fseek(file,0,SEEK_END);
    for(i=2;i<argc;i++){
        if(strcmp(argv[i], "-o") == 0){
            break;
        }
        FILE *inputFile = fopen(argv[i],"r");
        fprintf(file,"\n");
        int c;
        while((c = fgetc(inputFile)) != EOF){
                if(c != '\n'){
                fprintf(file,"%c",c);
                }else{
                fprintf(file, " ");
                }
        }
        fclose(inputFile);
        fseek(file,0,SEEK_END);
    }
    fprintf(file,"\n");
    fclose(file);
    printf("The files have been merged.\n");
}
```

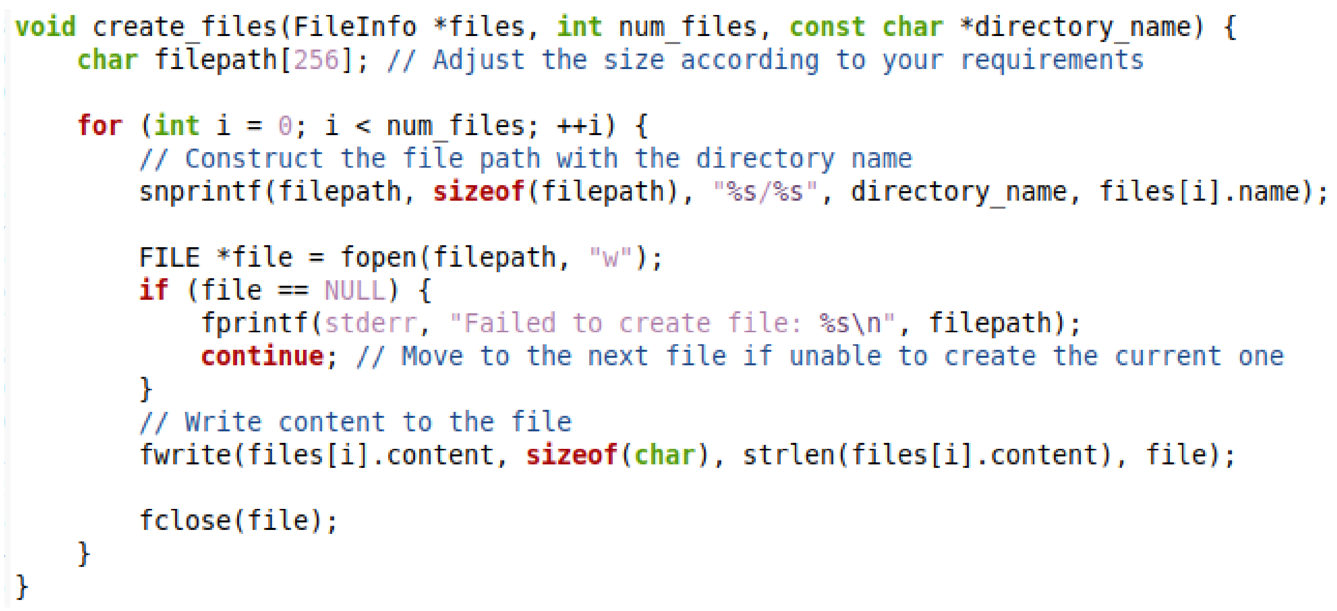
create_archive function basically creates the sau archive. First it creates the file and checks it. Then it sets archive info section by making a place holder 10 characters length. Later it puts filename, permissions and file size using other functions into sau archive. After completing the archive info section it takes its size and places it into first 10 characters. In any -o is confronted during creating archive, operation is stopped. Than every file that is to be merged are opened and their content is written into sau archive one by one. After all operations complete, both input and output files are closed.



```c
void extract_archive(char *archive_name, char *directory_name) {
    FILE *file = fopen(archive_name, "r");
    if (file == NULL) {
        printf("Failed to open archive %s.\n",archive_name);
        exit(1);
    }
    char buffer[11]; // Buffer to store the first 10 characters (+1 for null terminator)
    if (fscanf(file, "%10s", buffer) != 1) {
        printf("Failed to read archive  section size.\n");
        fclose(file);
        exit(1);
    }
    int archiveSize = atoi(buffer);
    int num_files = 0;
    FileInfo files[MAX_FILES];
    if (fseek(file, 10, SEEK_SET) != 0) {
        printf("Failed to seek to the 11th character.\n");
        fclose(file);
        exit(1);
    }
    fseek(file,10,SEEK_SET);
    char buff[archiveSize + 1 -10];
    size_t sectionLength = archiveSize -10;
    size_t bytes_read = fread(buff,sizeof(char),sectionLength,file);
    buff[bytes_read] = '\0'; //null terminate
    //printf("%s\n",buff);
    if (buff != NULL && num_files < MAX_FILES) {
        char *token = strtok(buff, "!|");
        while(token !=NULL){
                files[num_files].name = malloc(strlen(token) + 1);
                if (sscanf(token, "%[^,],%d,%d", files[num_files].name, &files[num_files].perms, &files[num_files].size) == 3) {
                        num_files++;
                    }
        token = strtok(NULL, "!|");
        }
    }
    fseek(file,archiveSize,SEEK_SET);
    int i;
    for (i = 0; i < num_files; ++i) {
        fscanf(file, "%d\n", &files[i].size);
        files[i].content = malloc(files[i].size + 1); // Allocate memory
        if (files[i].content != NULL) {
            if (fgets(files[i].content, files[i].size + 1, file) != NULL) {
                // Remove newline character if present
                size_t len = strlen(files[i].content);
                if (len > 0 && files[i].content[len - 1] == '\n') {
                    files[i].content[len - 1] = '\0';
                }
            } else {
                fprintf(stderr, "Failed to read content for file %d.\n", i + 1);
                free(files[i].content); // Free memory if content reading fails
                break;
            }
        } else {
            fprintf(stderr, "Memory allocation failed for file %d.\n", i + 1);
            break;
        }
    }
    create_files(files,num_files,directory_name);
    fclose(file);

}
```

extract_archive method is a bit more complicated. First it opens the archive, if can't error is displayed. Then it reads first 10 characters since I made it as archive's info section's size. After that, a new FileInfo struct is created to store file informations when extracting. I've created buff as archive' info section without size parameter as defined in first 10 characters. Then I read buff with strtok function using tokens. Those tokens made seperation process easier. Then I wrote those values into struct I've created earlier. In the last loop, I've added contents of the text files to the struct's content variable. At the end I had to make another function to create the files.



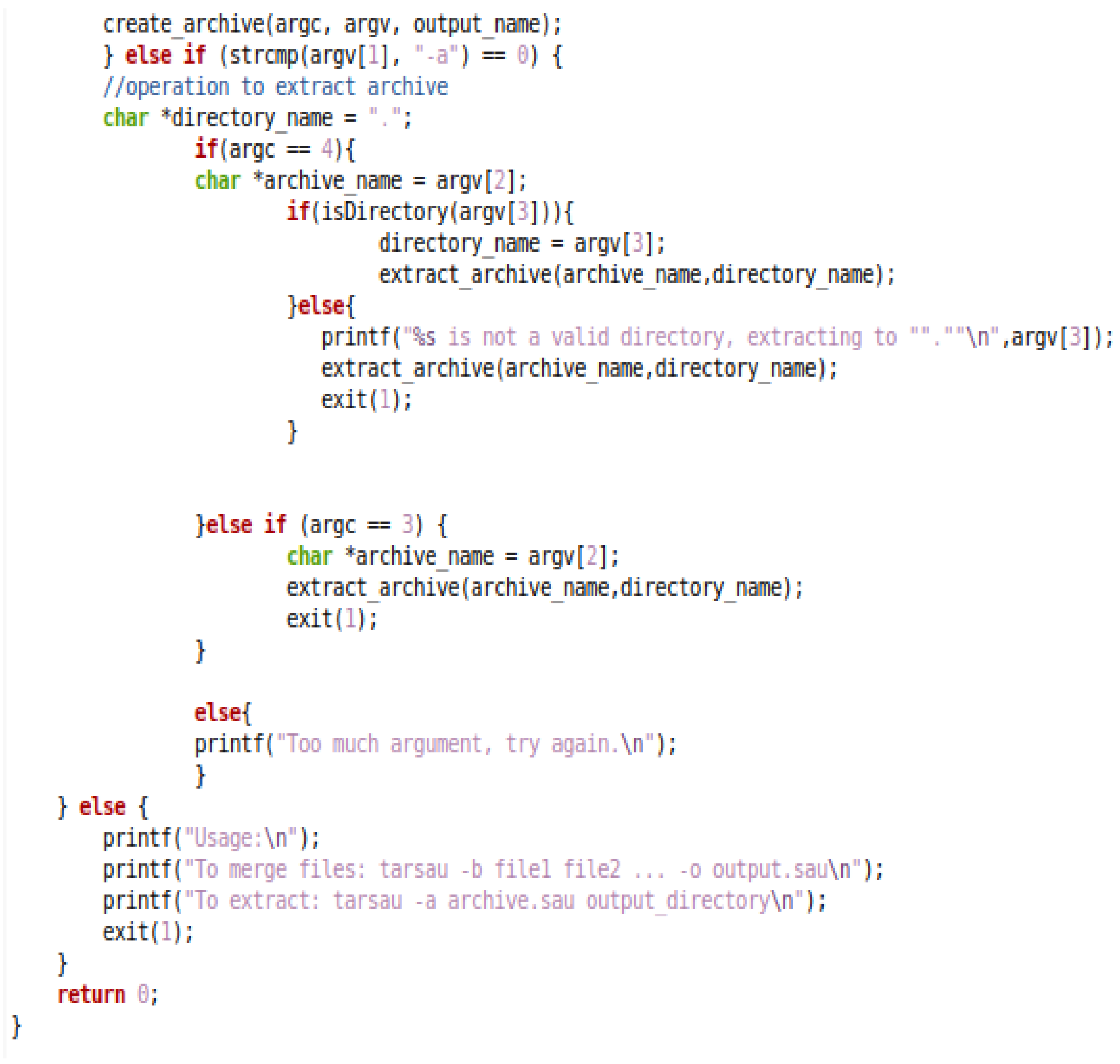
```c
void create_files(FileInfo *files, int num_files, const char *directory_name) {
    char filepath[256]; // Adjust the size according to your requirements

    for (int i = 0; i < num_files; ++i) {
        // Construct the file path with the directory name
        snprintf(filepath, sizeof(filepath), "%s/%s", directory_name, files[i].name);

        FILE *file = fopen(filepath, "w");
        if (file == NULL) {
            fprintf(stderr, "Failed to create file: %s\n", filepath);
            continue; // Move to the next file if unable to create the current one
        }
        // Write content to the file
        fwrite(files[i].content, sizeof(char), strlen(files[i].content), file);

        fclose(file);
    }
}
```

create_files function takes files struct, number of files and directory as parameter. Then creates the files one by one to given output.



```c
        create_archive(argc, argv, output_name);
        } else if (strcmp(argv[1], "-a") == 0) {
        //operation to extract archive
        char *directory_name = ".";
                if(argc == 4){
                char *archive_name = argv[2];
                        if(isDirectory(argv[3])){
                                directory_name = argv[3];
                                extract_archive(archive_name,directory_name);
                        }else{
                           printf("%s is not a valid directory, extracting to ".""\n",argv[3]);
                           extract_archive(archive_name,directory_name);
                           exit(1);
                        }


                }else if (argc == 3) {
                        char *archive_name = argv[2];
                        extract_archive(archive_name,directory_name);
                        exit(1);
                }

                else{
                printf("Too much argument, try again.\n");
                }
    } else {
        printf("Usage:\n");
        printf("To merge files: tarsau -b file1 file2 ... -o output.sau\n");
        printf("To extract: tarsau -a archive.sau output_directory\n");
        exit(1);
    }
    return 0;
}
```

I’ve handled errors and inputs and args in main function