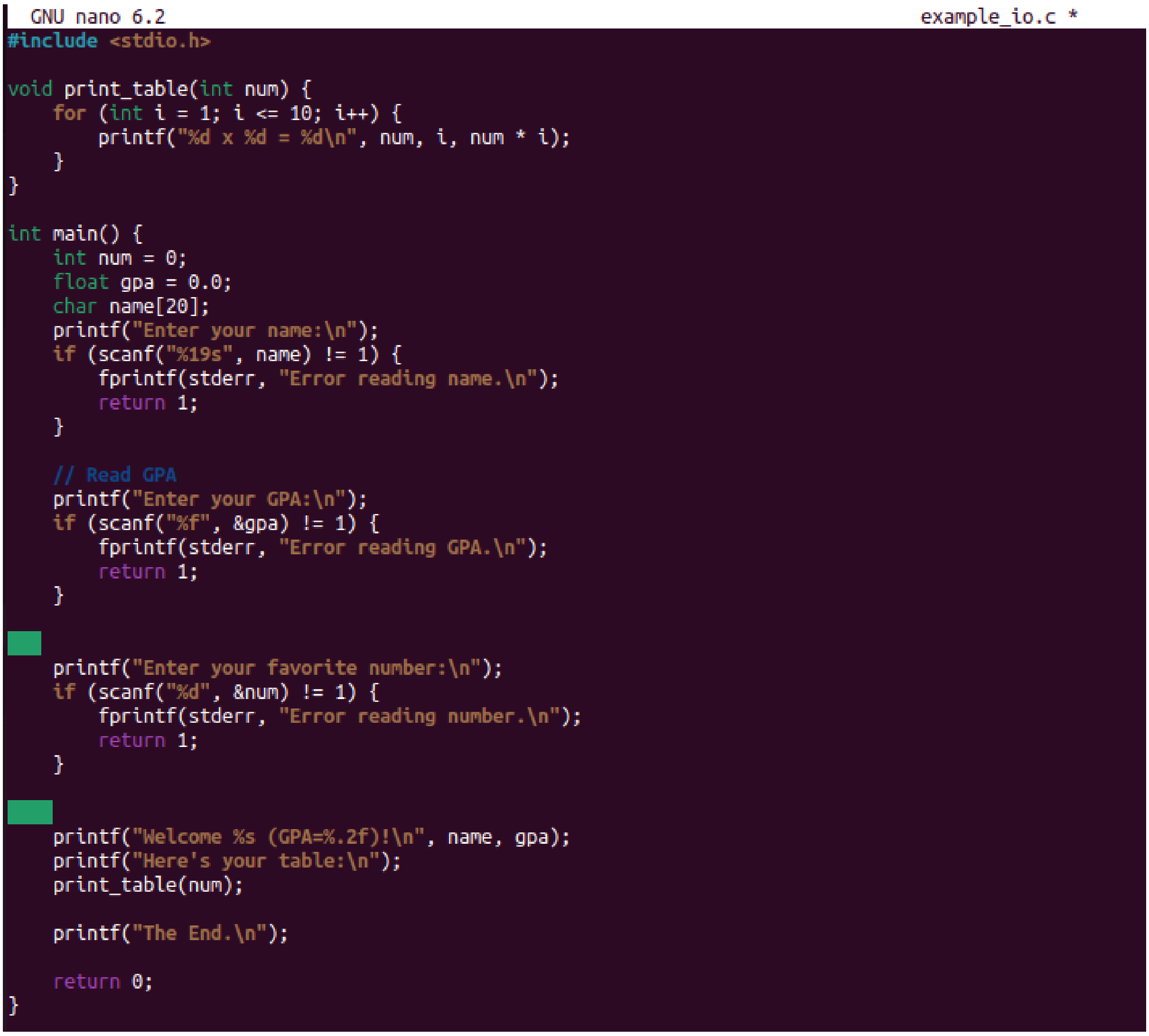
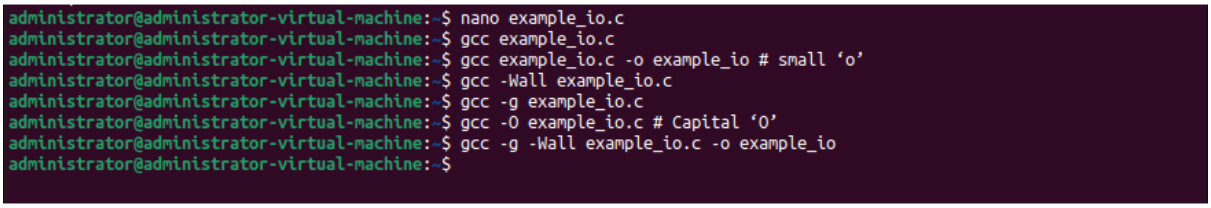
Q1



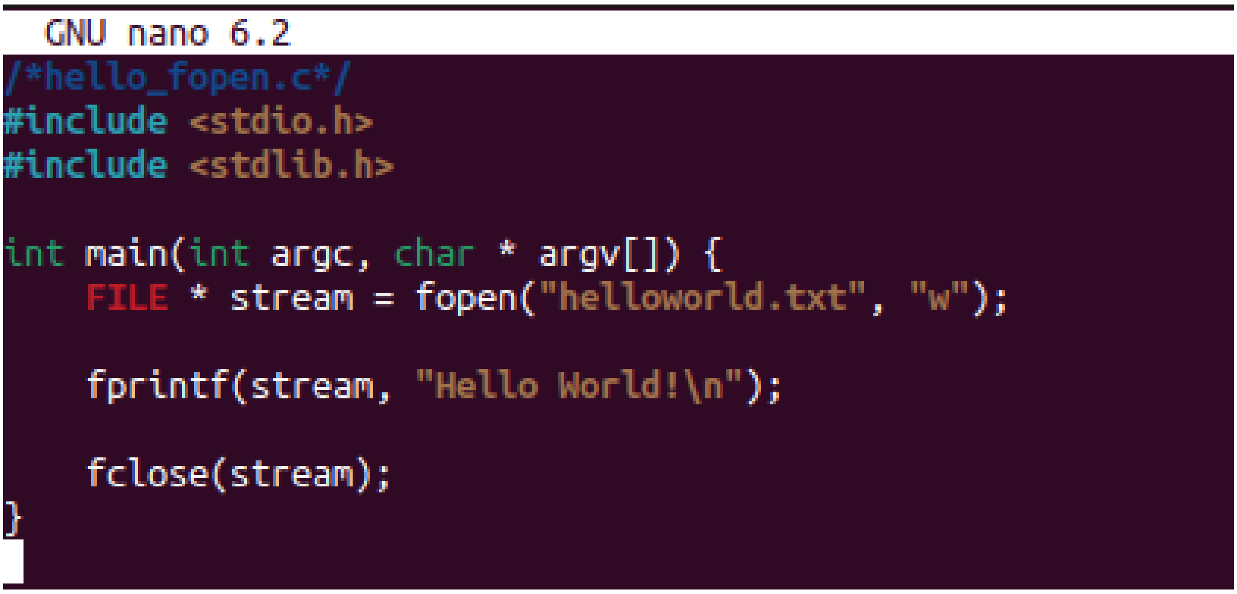
```
  GNU nano 6.2                                                          example_io.c *
#include <stdio.h>

void print_table(int num) {
    for (int i = 1; i <= 10; i++) {
        printf("%d x %d = %d\n", num, i, num * i);
    }
}

int main() {
    int num = 0;
    float gpa = 0.0;
    char name[20];
    printf("Enter your name:\n");
    if (scanf("%19s", name) != 1) {
        fprintf(stderr, "Error reading name.\n");
        return 1;
    }

    // Read GPA
    printf("Enter your GPA:\n");
    if (scanf("%f", &gpa) != 1) {
        fprintf(stderr, "Error reading GPA.\n");
        return 1;
    }


    printf("Enter your favorite number:\n");
    if (scanf("%d", &num) != 1) {
        fprintf(stderr, "Error reading number.\n");
        return 1;
    }


    printf("Welcome %s (GPA=%.2f)!\n", name, gpa);
    printf("Here's your table:\n");
    print_table(num);

    printf("The End.\n");

    return 0;
}
```



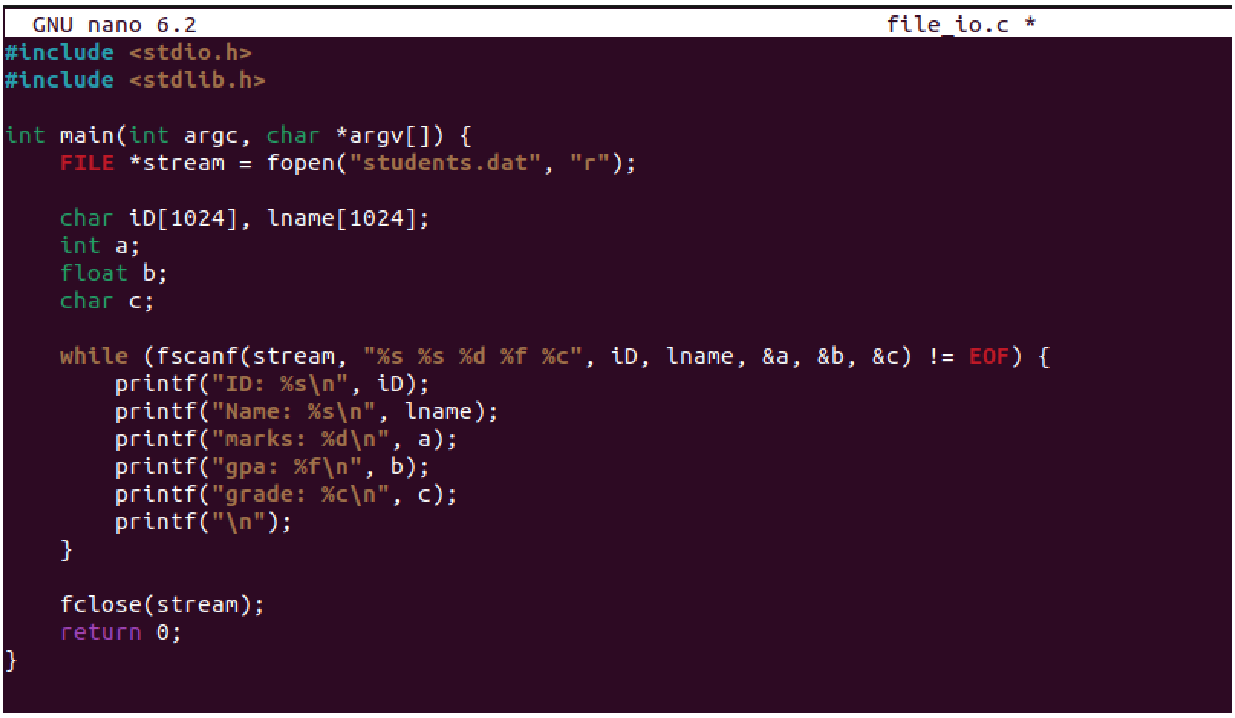
```
administrator@administrator-virtual-machine:~$ nano example_io.c
administrator@administrator-virtual-machine:~$ gcc example_io.c
administrator@administrator-virtual-machine:~$ gcc example_io.c -o example_io # small ‘o’
administrator@administrator-virtual-machine:~$ gcc -Wall example_io.c
administrator@administrator-virtual-machine:~$ gcc -g example_io.c
administrator@administrator-virtual-machine:~$ gcc -O example_io.c # Capital ‘O’
administrator@administrator-virtual-machine:~$ gcc -g -Wall example_io.c -o example_io
administrator@administrator-virtual-machine:~$
```

The above commands are many variations of the gcc command used to build C code. The **example_io.c** file is compiled by the first command, gcc example_io.c, creating an executable with the default name. The second command, **gcc example_io.c -o example_io**, compiles the same file but specifies the output file name in this case, example_io using the -o flag. The -Wall flag is added by the third command, **gcc -Wall example_io.c**, which helps the developer identify possible problems in the code by enabling all warnings during compilation. The -g flag is included in the fourth command, **gcc -g example_io.c**, which provides the executable with debugging information. The fifth command employs the -O flag , **gcc -O example_io.c** to enable optimization during compilation. Finally the sixth command **gcc -g -Wall example_io.c -o example_io** is the most comprehensive, as it includes optimizations for debugging and code warnings, along with an explicitly named output file.

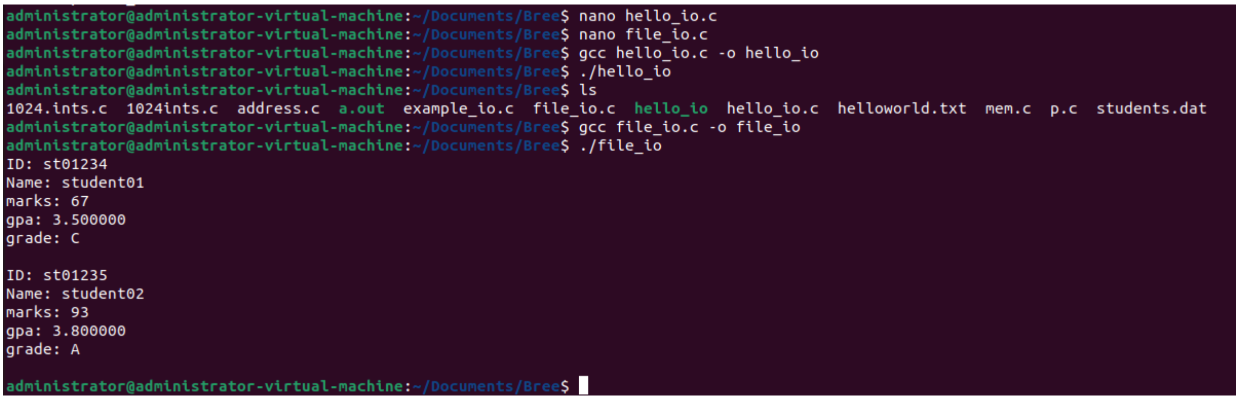
Q2



```
  GNU nano 6.2
/*hello_fopen.c*/
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char * argv[]) {
    FILE * stream = fopen("helloworld.txt", "w");

    fprintf(stream, "Hello World!\n");

    fclose(stream);
}
```

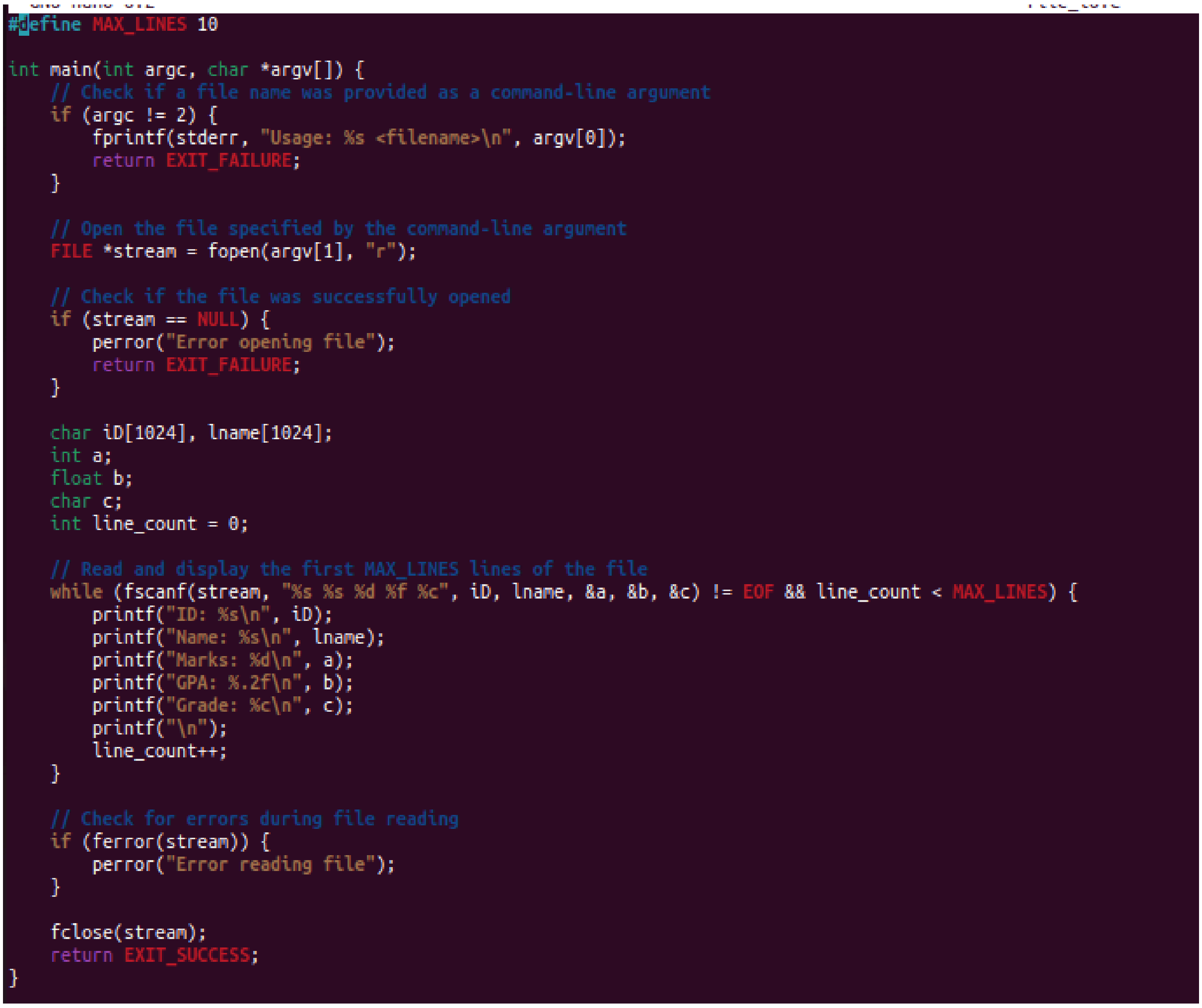


```
  GNU nano 6.2                                             file_io.c *
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    FILE *stream = fopen("students.dat", "r");

    char iD[1024], lname[1024];
    int a;
    float b;
    char c;

    while (fscanf(stream, "%s %s %d %f %c", iD, lname, &a, &b, &c) != EOF) {
        printf("ID: %s\n", iD);
        printf("Name: %s\n", lname);
        printf("marks: %d\n", a);
        printf("gpa: %f\n", b);
        printf("grade: %c\n", c);
        printf("\n");
    }

    fclose(stream);
    return 0;
}
```



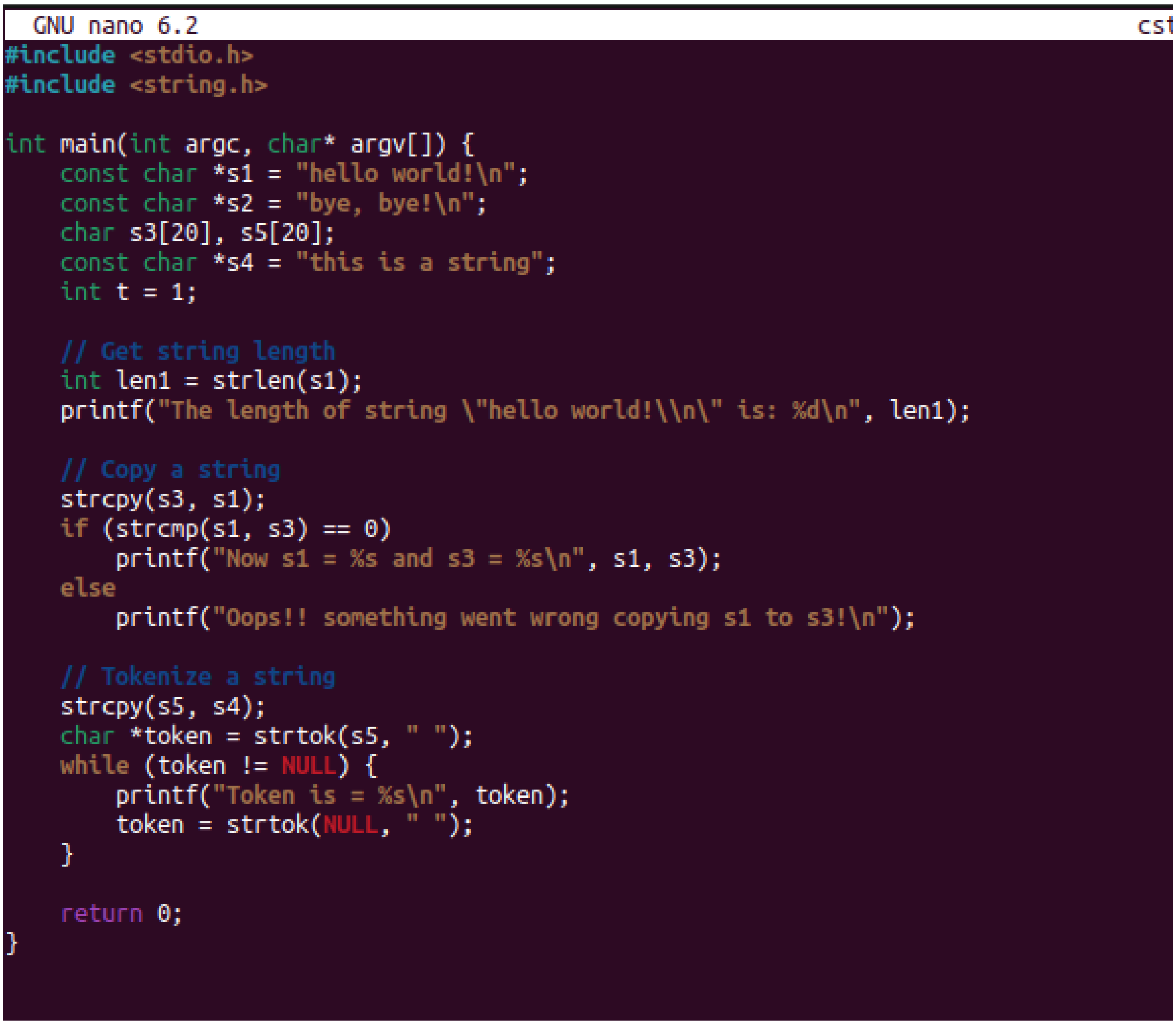
```
administrator@administrator-virtual-machine:~/Documents/Bree$ nano hello_io.c
administrator@administrator-virtual-machine:~/Documents/Bree$ nano file_io.c
administrator@administrator-virtual-machine:~/Documents/Bree$ gcc hello_io.c -o hello_io
administrator@administrator-virtual-machine:~/Documents/Bree$ ./hello_io
administrator@administrator-virtual-machine:~/Documents/Bree$ ls
1024.ints.c  1024ints.c  address.c  a.out  example_io.c  file_io.c  hello_io  hello_io.c  helloworld.txt  mem.c  p.c  students.dat
administrator@administrator-virtual-machine:~/Documents/Bree$ gcc file_io.c -o file_io
administrator@administrator-virtual-machine:~/Documents/Bree$ ./file_io
ID: st01234
Name: student01
marks: 67
gpa: 3.500000
grade: C

ID: st01235
Name: student02
marks: 93
gpa: 3.800000
grade: A

administrator@administrator-virtual-machine:~/Documents/Bree$ 
```

Modified Code

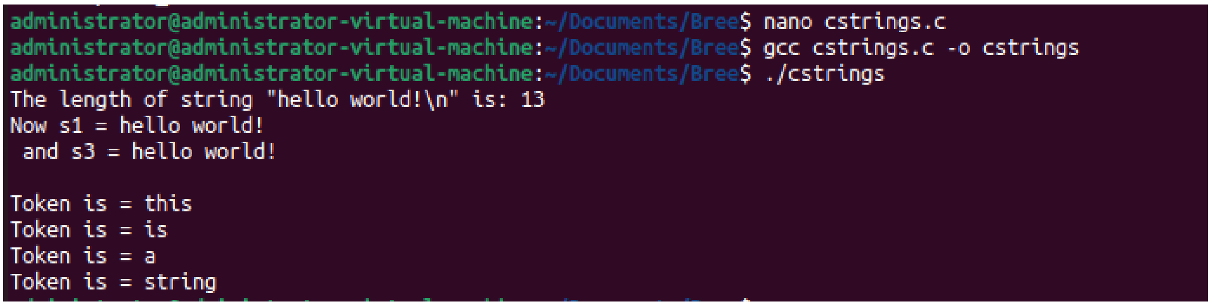


```c
#define MAX_LINES 10

int main(int argc, char *argv[]) {
    // Check if a file name was provided as a command-line argument
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <filename>\n", argv[0]);
        return EXIT_FAILURE;
    }

    // Open the file specified by the command-line argument
    FILE *stream = fopen(argv[1], "r");

    // Check if the file was successfully opened
    if (stream == NULL) {
        perror("Error opening file");
        return EXIT_FAILURE;
    }

    char iD[1024], lname[1024];
    int a;
    float b;
    char c;
    int line_count = 0;

    // Read and display the first MAX_LINES lines of the file
    while (fscanf(stream, "%s %s %d %f %c", iD, lname, &a, &b, &c) != EOF && line_count < MAX_LINES) {
        printf("ID: %s\n", iD);
        printf("Name: %s\n", lname);
        printf("Marks: %d\n", a);
        printf("GPA: %.2f\n", b);
        printf("Grade: %c\n", c);
        printf("\n");
        line_count++;
    }

    // Check for errors during file reading
    if (ferror(stream)) {
        perror("Error reading file");
    }

    fclose(stream);
    return EXIT_SUCCESS;
}
```

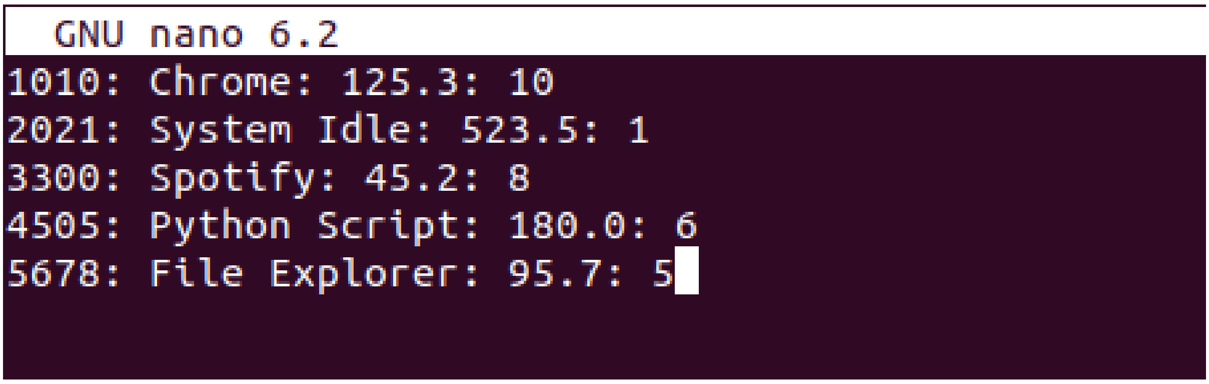
Q3



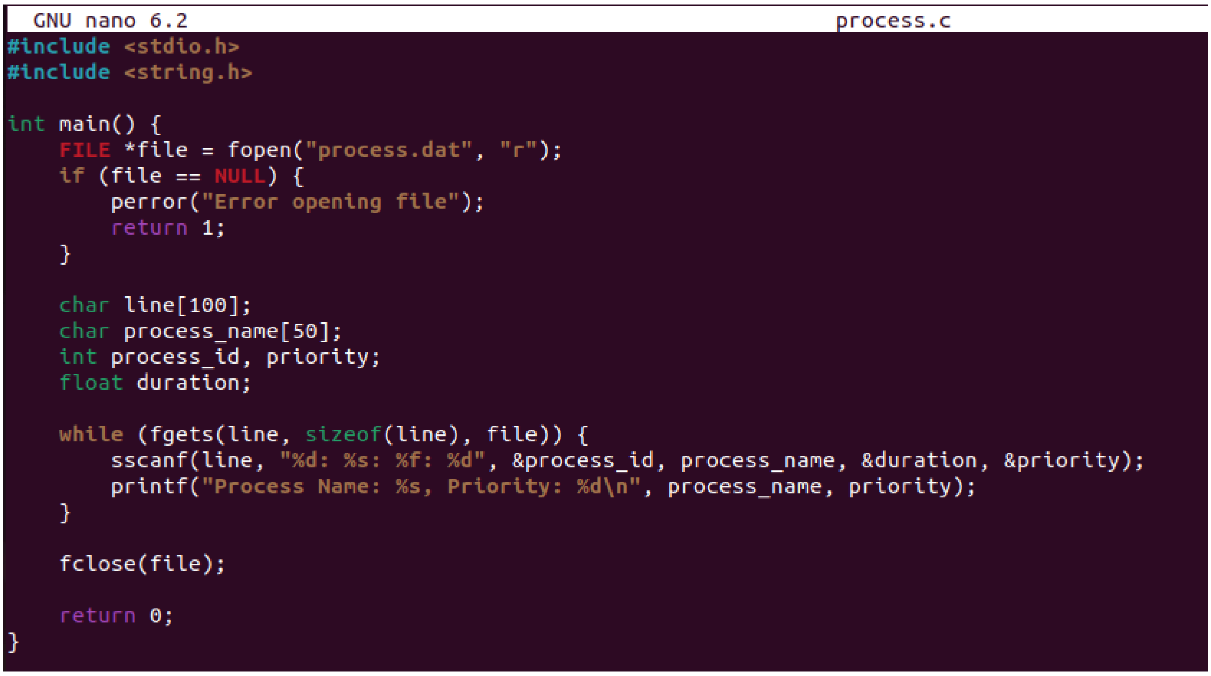
```
  GNU nano 6.2                                                                              cst
#include <stdio.h>
#include <string.h>

int main(int argc, char* argv[]) {
    const char *s1 = "hello world!\n";
    const char *s2 = "bye, bye!\n";
    char s3[20], s5[20];
    const char *s4 = "this is a string";
    int t = 1;

    // Get string length
    int len1 = strlen(s1);
    printf("The length of string \"hello world!\\n\" is: %d\n", len1);

    // Copy a string
    strcpy(s3, s1);
    if (strcmp(s1, s3) == 0)
        printf("Now s1 = %s and s3 = %s\n", s1, s3);
    else
        printf("Oops!! something went wrong copying s1 to s3!\n");

    // Tokenize a string
    strcpy(s5, s4);
    char *token = strtok(s5, " ");
    while (token != NULL) {
        printf("Token is = %s\n", token);
        token = strtok(NULL, " ");
    }

    return 0;
}
```



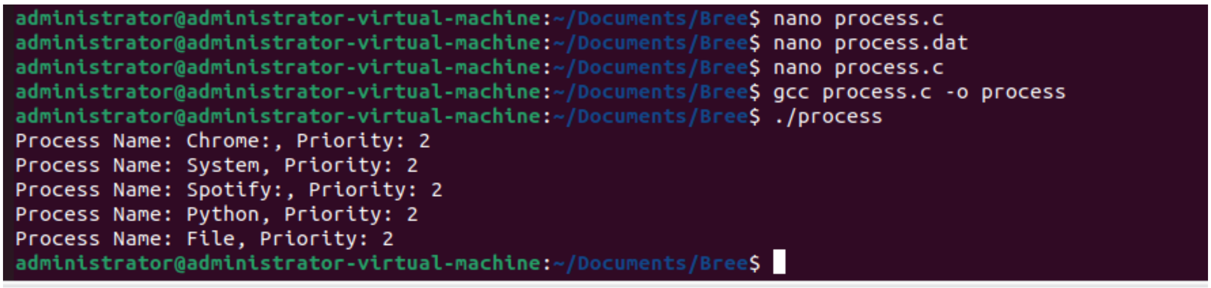
```
administrator@administrator-virtual-machine:~/Documents/Bree$ nano cstrings.c
administrator@administrator-virtual-machine:~/Documents/Bree$ gcc cstrings.c -o cstrings
administrator@administrator-virtual-machine:~/Documents/Bree$ ./cstrings
The length of string "hello world!\n" is: 13
Now s1 = hello world!
 and s3 = hello world!

Token is = this
Token is = is
Token is = a
Token is = string
```



```
  GNU nano 6.2
1010: Chrome: 125.3: 10
2021: System Idle: 523.5: 1
3300: Spotify: 45.2: 8
4505: Python Script: 180.0: 6
5678: File Explorer: 95.7: 5
```

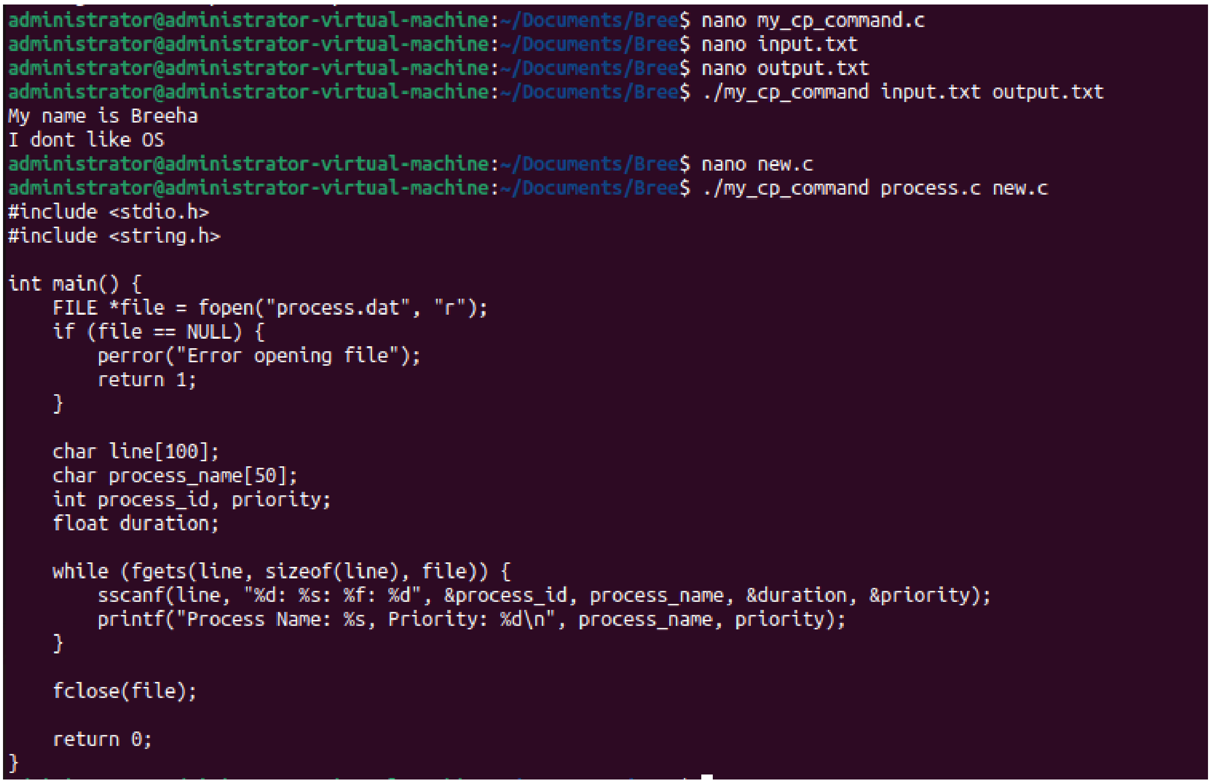


```
  GNU nano 6.2                                                      process.c
#include <stdio.h>
#include <string.h>

int main() {
    FILE *file = fopen("process.dat", "r");
    if (file == NULL) {
        perror("Error opening file");
        return 1;
    }

    char line[100];
    char process_name[50];
    int process_id, priority;
    float duration;

    while (fgets(line, sizeof(line), file)) {
        sscanf(line, "%d: %s: %f: %d", &process_id, process_name, &duration, &priority);
        printf("Process Name: %s, Priority: %d\n", process_name, priority);
    }

    fclose(file);

    return 0;
}
```

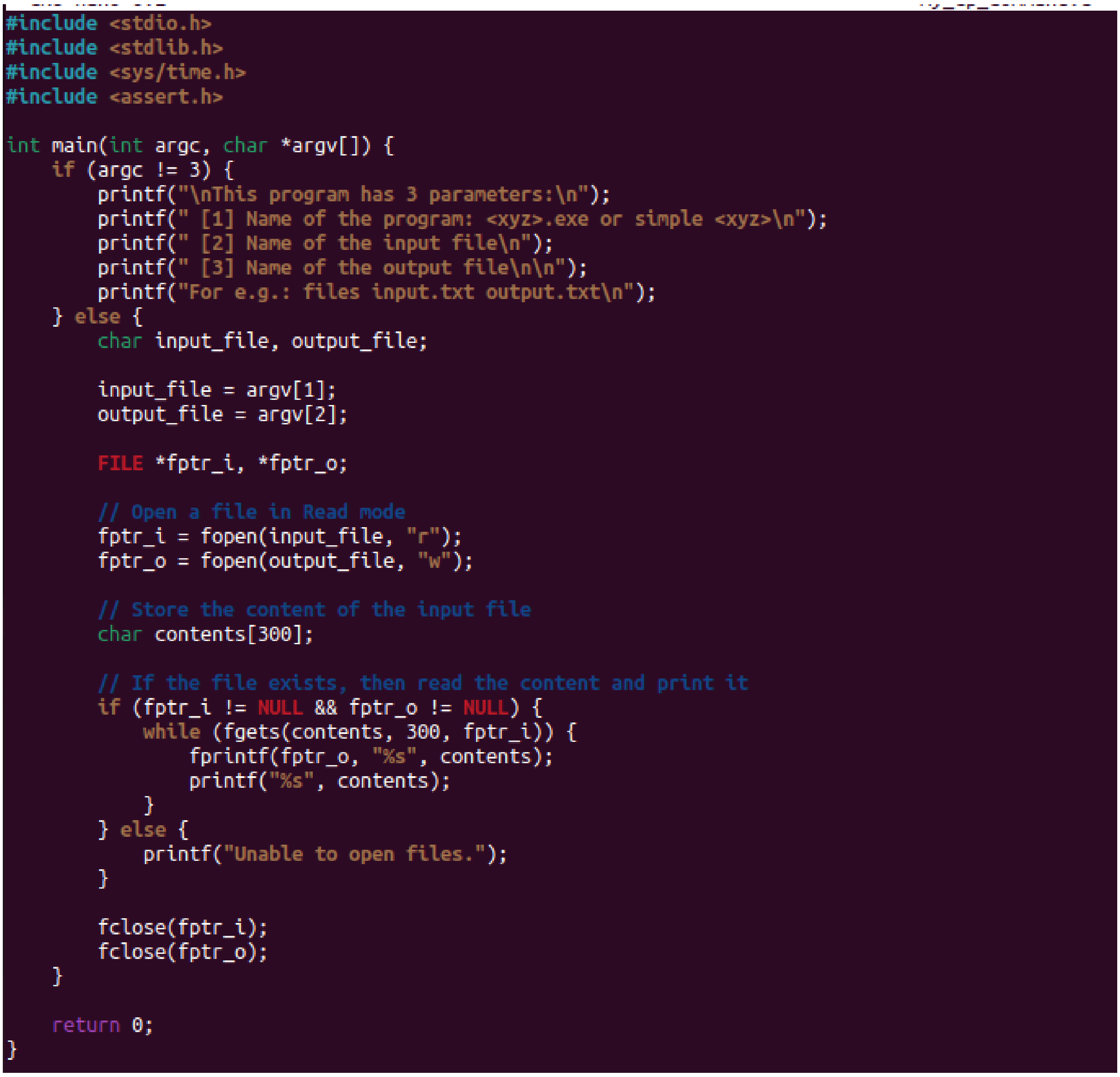


```
administrator@administrator-virtual-machine:~/Documents/Bree$ nano process.c
administrator@administrator-virtual-machine:~/Documents/Bree$ nano process.dat
administrator@administrator-virtual-machine:~/Documents/Bree$ nano process.c
administrator@administrator-virtual-machine:~/Documents/Bree$ gcc process.c -o process
administrator@administrator-virtual-machine:~/Documents/Bree$ ./process
Process Name: Chrome:, Priority: 2
Process Name: System, Priority: 2
Process Name: Spotify:, Priority: 2
Process Name: Python, Priority: 2
Process Name: File, Priority: 2
administrator@administrator-virtual-machine:~/Documents/Bree$ 
```

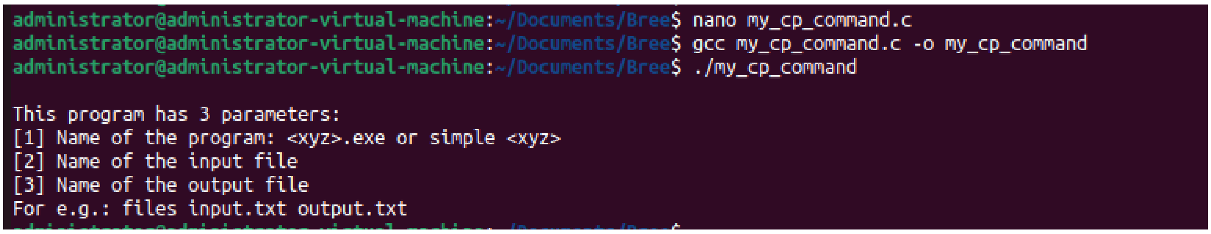
Q4



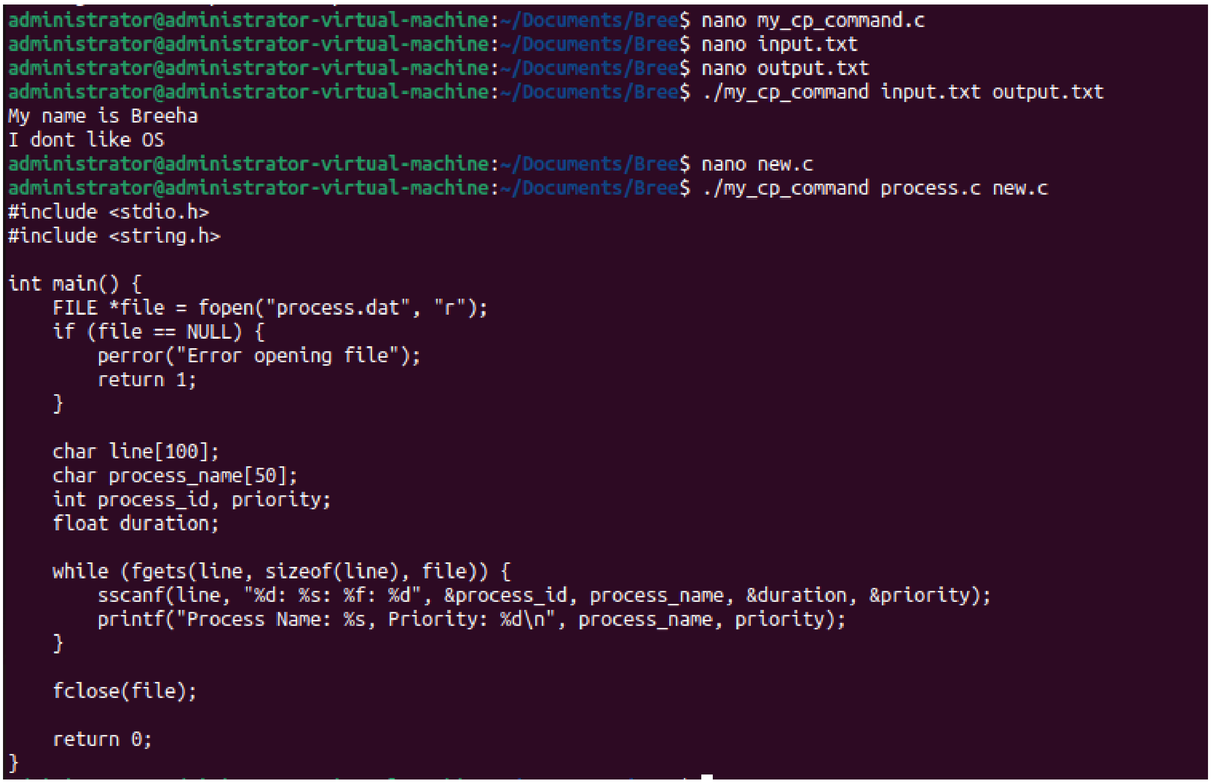
```
administrator@administrator-virtual-machine:~/Documents/Bree$ nano my_cp_command.c
administrator@administrator-virtual-machine:~/Documents/Bree$ nano input.txt
administrator@administrator-virtual-machine:~/Documents/Bree$ nano output.txt
administrator@administrator-virtual-machine:~/Documents/Bree$ ./my_cp_command input.txt output.txt
My name is Breeha
I dont like OS
administrator@administrator-virtual-machine:~/Documents/Bree$ nano new.c
administrator@administrator-virtual-machine:~/Documents/Bree$ ./my_cp_command process.c new.c
#include <stdio.h>
#include <string.h>

int main() {
    FILE *file = fopen("process.dat", "r");
    if (file == NULL) {
        perror("Error opening file");
        return 1;
    }

    char line[100];
    char process_name[50];
    int process_id, priority;
    float duration;

    while (fgets(line, sizeof(line), file)) {
        sscanf(line, "%d: %s: %f: %d", &process_id, process_name, &duration, &priority);
        printf("Process Name: %s, Priority: %d\n", process_name, priority);
    }

    fclose(file);

    return 0;
}
```



```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <assert.h>

int main(int argc, char *argv[]) {
    if (argc != 3) {
        printf("\nThis program has 3 parameters:\n");
        printf(" [1] Name of the program: <xyz>.exe or simple <xyz>\n");
        printf(" [2] Name of the input file\n");
        printf(" [3] Name of the output file\n\n");
        printf("For e.g.: files input.txt output.txt\n");
    } else {
        char input_file, output_file;

        input_file = argv[1];
        output_file = argv[2];

        FILE *fptr_i, *fptr_o;

        // Open a file in Read mode
        fptr_i = fopen(input_file, "r");
        fptr_o = fopen(output_file, "w");

        // Store the content of the input file
        char contents[300];

        // If the file exists, then read the content and print it
        if (fptr_i != NULL && fptr_o != NULL) {
            while (fgets(contents, 300, fptr_i)) {
                fprintf(fptr_o, "%s", contents);
                printf("%s", contents);
            }
        } else {
            printf("Unable to open files.");
        }

        fclose(fptr_i);
        fclose(fptr_o);
    }

    return 0;
}
```



```
administrator@administrator-virtual-machine:~/Documents/Bree$ nano my_cp_command.c
administrator@administrator-virtual-machine:~/Documents/Bree$ gcc my_cp_command.c -o my_cp_command
administrator@administrator-virtual-machine:~/Documents/Bree$ ./my_cp_command

This program has 3 parameters:
[1] Name of the program: <xyz>.exe or simple <xyz>
[2] Name of the input file
[3] Name of the output file
For e.g.: files input.txt output.txt
```



```
administrator@administrator-virtual-machine:~/Documents/Bree$ nano my_cp_command.c
administrator@administrator-virtual-machine:~/Documents/Bree$ nano input.txt
administrator@administrator-virtual-machine:~/Documents/Bree$ nano output.txt
administrator@administrator-virtual-machine:~/Documents/Bree$ ./my_cp_command input.txt output.txt
My name is Breeha
I dont like OS
administrator@administrator-virtual-machine:~/Documents/Bree$ nano new.c
administrator@administrator-virtual-machine:~/Documents/Bree$ ./my_cp_command process.c new.c
#include <stdio.h>
#include <string.h>

int main() {
    FILE *file = fopen("process.dat", "r");
    if (file == NULL) {
        perror("Error opening file");
        return 1;
    }

    char line[100];
    char process_name[50];
    int process_id, priority;
    float duration;

    while (fgets(line, sizeof(line), file)) {
        sscanf(line, "%d: %s: %f: %d", &process_id, process_name, &duration, &priority);
        printf("Process Name: %s, Priority: %d\n", process_name, priority);
    }

    fclose(file);

    return 0;
}
```

Q5

The C compilation process starts with preprocessor, it processes the source program “hello.c” by handling preprocessor directives for e.g. #include and #define. During this process macros are changed and any header file contents are inserted to create a modified source program “hello.i”. After that, the compiler converts the altered source into assembly code, such as “hello.s”, which is more akin to machine code but still readable by humans. This assembly code is translated into machine language by assembler producing an object file “hello.o” which has binary format. Then finally linker combines object file with other necessary object files to produce executable file “hello”.