Задача 4. В текущия каталог се намира текстов файл file.txt със следното съдържание

```
abcdef
0123456789
ABCD
```

Изпълнимият файл, полу§ен след компилация на зададения по-долу програмен фрагмент, се стартира с командния ред:

```
./a.out ff
```

Напишете какво ще бъде изведено на стандартния изход и какво ще бъде съдържанието на двата файла след приключване на успешното изпълнение

```
1     #include <stdio.h>
2     #include <fcntl.h>
3     main(int argc, char* argv[]){
4           int fdi, fdo, k, broi, i=0, status;
5           char buff[40], c;
6           if((fdi = open("file.txt", O_RDWR)) == -1){
7                 printf("\n Cannot open \n"); exit(1);
8           }
9           if((fdo = open(argv[1], O_CREAT|O_TRUNC|O_RDWR, 0666)) == -1){
10                printf("\n Cannot open \n"); exit(1);
11          }
12          if(fork() == 0){
13                k = dup(1); close(1); dup(fdi);
14                broi = read(fdi, buff, 40);
15                c = buff[i++];
16                if(c <= '0' || c >= '9'){
17                      while(buff[i++]!='\n' && i < broi) write(1,"*",1);
18                      write(1, "\n", 1);
19                      close(1); dup(k);
20                      write(1, buff, 3);
21                      write(fdo, buff, 10);
22                } else {
23                      write(1, buff, broi); close(1); dup(k);
24                      write(1, "*\n", 2);
25                }
26                lseek(fdo, 0, 0);
27                write(fdo, "*\n", 2);
28                close(fdi); close(fdo);
29          } else {
30                wait(&status);
31                close(1); dup(fdi);
32                execlp("grep", "grep", "c", argv[1], 0);
33                execlp("wc", "wc", "-l", "ff", 0);
34          }
35    }
```

Решение.

Компилираме файла "a.c" с командата "gcc -Wall -o a.out a.c". Подаваме на gcc компилатора следните аргументи "-Wall" за да ни каже дали някъде имаме предупреждения за грешки, след което (след -o, като output) задаваме името на output изпълнимия файл ("a.out") и името на файла, който ще компилираме ("a.c").

- На ред 4 дефинираме променливите от целочислен тип fdi (file descriptor in), fdo (file descriptor out), k, broi, i и status, като само променливата i я инициализираме с 1.
- На ред 5 създаваме масив buff от 40 символа, както и символ c.
- На ред 6 отваряме файл с име "file.txt" за четене и писане и запаметяваме файловия дескриптор в променливата fdi. След това проверяваме дали този файлов дескриптор fdi е равен на -1. Ако е равен на -1, това ще означава че отварянето на файл file.txt не е било успешно и в if блока на следващия 7-ми ред принтираме подходящо съобщение и завършваме програмата с код за грешка.
- На ред 9 отваряме файла подаден като първи аргумент на програмата с O_CREAT, O_TRUNC и O_RDWR флагове.

  O_CREAT - създава файла с име стойността на аргумента argv[1] ("ff"), ако такъв файл не съществува. При създаване се задават правата му за owner, group и others на разрешени за четене и писане: -rw-rw-rw- (0666).
  O_TRUNC - ако вече съществува файл с такова име, то съдържанието му се изчиства.
  O_RDWR - файлът който се е създал или изчистил се отваря за четене и писане.

- Ред 10 е аналогичен на ред 10.
- На ред 12 системното извикване fork( ) създава детински процес и PID на детинския процес ще бъде върнат в родителския процес, а 0 ще бъде върната в детинския процес. По този начин чрез проверката fork( ) == 0 ще знаем, че if блока ще се изпълни в детинския процес, а else блока в родителския процес.
- На ред 13 се намираме в детинския процес. Правим дубликат на файловия дескриптор 1 в променливата k. Сега k сочи съм стандартния изход (0 - stdin, 1 - stdout, 2 - stderr).

  close(1). Затваряме файловия дескриптор 1 (за стандартния изход, който доблирахме), за да го освободим.

  dup(fdi). Правим дубликат на файловия дескриптор който сочи към файла "file.txt" на първата (най-малка) свободна позиция за дескриптор, която е току що освободената 1-ца. Сега файловият дескриптор 1 (stdout) сочи към файла "file.txt". Тоест стандартния т изход вече пише във файла "file.txt"
- На ред 14 прочитаме до 40 байта от файла "file.txt" и ги съхраняваме в масива от символи buff. Системното извикване read връща броя на прочетените байтове, които се съхраняват в променливата broi.
- На ред 15 взимаме първия символ от буфера buff и инкрементираме брояча i с единица.
- На ред 16 проверяваме дали първият символ от буфера е различен от цифра.
- На ред 17 стартираме while цикъл, който цикли докато в буфера не се срещне символа за нов ред '\n' или не се достигне края му. Инкрементирането на брояча i, с който четем от буфера става вътре в while цикъла след проверката. В тялото на while цикъла се извиква системната команда write(...), която записва във файловия дескриптор 1 (стандартния изход, сочещ към файла "file.txt") един байт от стринга "*", тоест само звездата без дефолтния нов ред на края на стринга (ще се запишат общо 5 зведички).
- На ред 18 след края на изпълнението на while цикъла се записва нов ред във файла "file.txt".
- На ред 19 затваряме файловият дескриптор 1, който до сега сочеше към файла "file.txt". След това правим дубликат на свободния файлов дескриптор 1 - файловият дескриптор запаметен в променливата k, тоест възстановяваме дефолтната наредба на файловите дескриптори.
- На ред 20 записваме първите 3 байта от буфера buff на стандартния изход (stdout), а именно "abc".

- На ред 21 записваме първите 10 байта от буфера buff във файла "ff", подаден като първи аргумент на програмата (файловият дескриптор fdo сочи към този файл). Тоест записваме във файла "ff" следното съдържание "abcdef\n012".
- На ред 22 никога няма да влезем в else блока на if-else конструкцията, тъй като първият символ от файла "file.txt" не е цифра и влизаме в if блока.
- Редовете 23 и 24 НЕ се изпълняват, но ако се изпълняваха щяха да запишат във файла "file.txt" броя прочетени байтове в буфера buff (тоест цялото съдържание на файла "file.txt"), да възстановят след това дефолтната наредба на файловите дескриптори и да запишат два байта от стринга "*\n" на стандартния изход (конзолата) - тоест да изпечатат на конзолата звездичка и нов ред.
- на ред 26 позиционираме отместването на четеца във файла "ff" да е 0 байта от началото му (0 = SEEK_SET).
- На ред 27 записваме два байта от стринга "*\n" на мястото на първите два байта от съдържанието му. Новото му съдържание вече е "*\ncdef\n012".
- На ред 28 затваряме файловите дескриптори сочещи към двата файла.
- На ред 29 else блока ще се изпълни от родителския процес.
- На ред 30 първата системна команда wait( ) ще изчака детинския процес да завърши и ще запамети статус кода от терминацията му.
- На ред 31 затваряме файловия дескриптор към стандартния изход и 1 става свободен, като след това дублираме на негово място файловия дескриптор към файла "file.txt". Сега стандартния изход stdout сочи към файла "file.txt".
- На ред 32 ще се изпълни командата grep с параметри "c" и "ff", която ще провери за съвпадения на символа "c" във всеки ред на файла и ще го запише във "file.txt" ако такова съвпадение има. Нулата на края на execlp командата ще се конвентира до (char*)NULL, с което ще се обозначи за край на подаваните аргументи на командата grep.
  Тоест ще се запише реда "cdef" във файла "file.txt", тъй като това е единственият ред от файла "ff", в който има сумвола "c".
- На ред 33 НЯМА да се изпълни командата wc с аргументи "-l" и "ff", която ще принтира броя редове във файла "ff" и името му с водеща табулация. Това е така, защото предходното извикване на командата execlp на ред 32 заменя текста, данните, паметта в хиип-а и стековата памет с нова програма от диска (но забележете, че файловите дескриптори се запазват, те се менажират от kernel-а и не са част от новия image). Ако искаме да се изпълнят няколко команди с execlp е необходимо да ги извикваме през детински процеси.

Окончателно:

На конзолата ще се принтира:

```
abc
```

Съдържание на файл "file.txt":

```
abcdef
0123456789
ABCD
*****
cdef
```

Съдържание на файл "ff":

```
*
cdef
012
```

□