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

const char * usage = ""
"Usage:\n"
"    cat_grep file-name word outfile\n"
"\n"
"    It does something similar to the shell command:\n"
"        $> cat file | grep word > outfile\n"
"\n"
"Example:\n"
"    $./a.out cat_grep.c printf outputfile\n"
"    then run \n"
"    $cat outputfile\n\n";

const char *cat = "cat";
const char *grep = "grep";

int main(int argc, char **argv)
{
        if (argc < 4) {
                fprintf(stderr, "%s", usage );
                exit(1);
        }

        // Save default input, output, and error because we will
        // change them during redirection and we will need to restore
them
        // at the end.

        int defaultin = dup( 0 );
        int defaultout = dup( 1 );
        int defaulterr = dup( 2 );

        //////////////////  cat ////////////////////////

        // Input:    defaultin
        // Output:   pipe
        // Error:    defaulterr

        // Create new pipe

        int fdpipe[2];
        if ( pipe(fdpipe) == -1) {
                perror( "cat_grep: pipe");
                exit( 2 );
        }



        // Redirect output to pipe
```

```
        dup2( fdpipe[ 1 ], 1 );

        // Redirect err

        // Create new process for "cat"
        int pid = fork();
        if ( pid == -1 ) {
                perror( "cat_grep: fork\n");
                exit( 2 );
        }

        if (pid == 0) {
                //Child

                // close file descriptors that are not needed
                close(fdpipe[0]);
                close(fdpipe[1]);
                close( defaultin );
                close( defaultout );
                close( defaulterr );

                // You can use execvp() instead if the arguments are
stored in an array
                execlp(cat, cat, argv[1], (char *) 0);

                // exec() is not suppose to return, something went
wrong
                perror( "cat_grep: exec cat");
                exit( 2 );
        }

        //////////////////  grep ///////////////////////////

        // Input:    pipe
        // Output:   outfile
        // Error:    defaulterr

        // Redirect input.
        dup2( fdpipe[0], 0);

        // Redirect output to utfile
        int outfd = creat( argv[ 3 ], 0666 );

        if ( outfd < 0 ) {
                perror( "cat_grep: creat outfile" );
                exit( 2 );
        }

        dup2( outfd, 1 );
        close( outfd );

        // Redirect err
        dup2( defaulterr, 2 );

        pid = fork();
        if (pid == -1 ) {
                perror( "cat_grep: fork");
```

```
                exit( 2 );
        }

        if (pid == 0) {
                //Child

                // close file descriptors that are not needed
                close(fdpipe[0]);
                close(fdpipe[1]);
                close( defaultin );
                close( defaultout );
                close( defaulterr );

                // You can use execvp() instead if the arguments are
stored in an array
                execlp(grep, cat, argv[2], (char *) 0);

                // exec() is not suppose to return, something went
wrong
                perror( "cat_grep: exec grep");
                exit( 2 );

        }

        // Restore input, output, and error

        dup2( defaultin, 0 );
        dup2( defaultout, 1 );
        dup2( defaulterr, 2 );

        // Close file descriptors that are not needed
        close(fdpipe[0]);
        close(fdpipe[1]);
        close( defaultin );
        close( defaultout );
        close( defaulterr );

        // Wait for last process in the pipe line
        waitpid( pid, 0, 0 );

        exit( 2 );
}
```