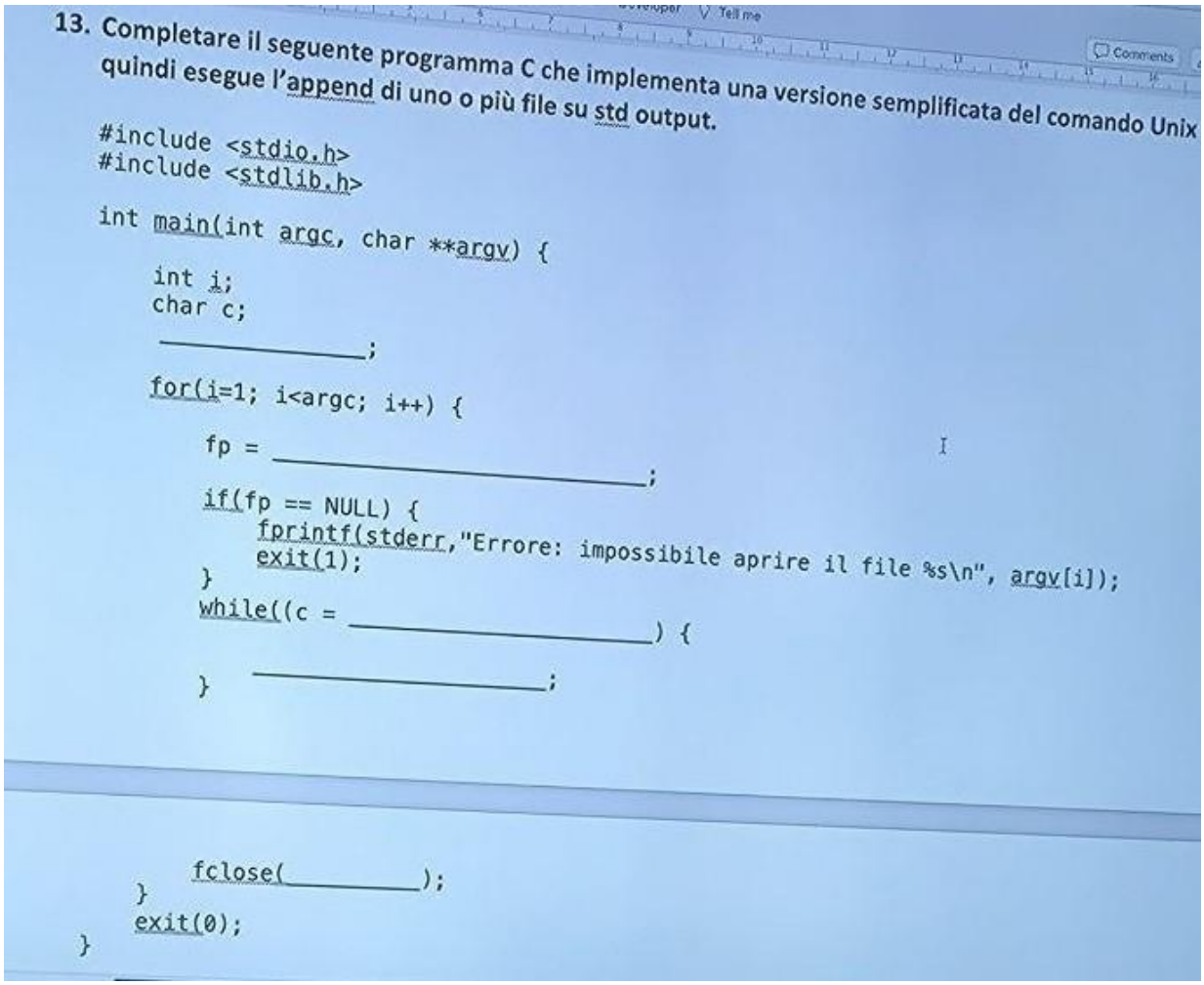
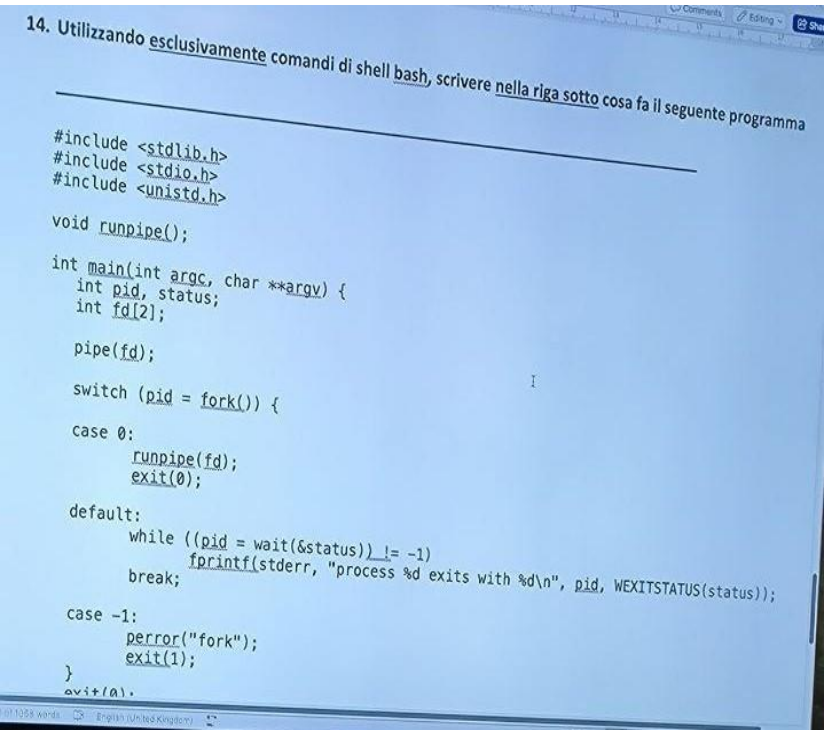
Completamento



13. Completare il seguente programma C che implementa una versione semplificata del comando Unix quindi esegue l'append di uno o più file su std output.

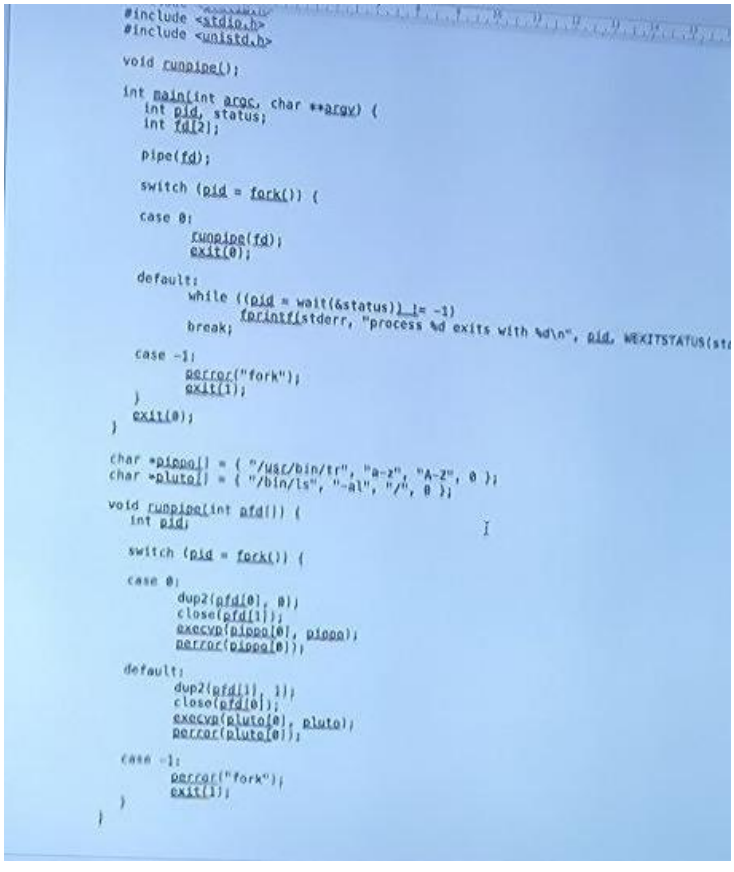
```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv) {

    int i;
    char c;

    _____;

    for(i=1; i<argc; i++) {

        fp = _____;

        if(fp == NULL) {
            fprintf(stderr,"Errore: impossibile aprire il file %s\n", argv[i]);
            exit(1);
        }
        while((c = _____) {

        }  _____;

        fclose(_____);
    }
    exit(0);
}
```

Pipe



14. Utilizzando esclusivamente comandi di shell bash, scrivere nella riga sotto cosa fa il seguente programma

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

void runpipe();

int main(int argc, char **argv) {
   int pid, status;
   int fd[2];

   pipe(fd);

   switch (pid = fork()) {

   case 0:
         runpipe(fd);
         exit(0);

   default:
         while ((pid = wait(&status)) != -1)
               fprintf(stderr, "process %d exits with %d\n", pid, WEXITSTATUS(status));
         break;

   case -1:
         perror("fork");
         exit(1);
   }
   exit(0);
```



```
#include <stdio.h>
#include <unistd.h>

void runpipe();

int main(int argc, char **argv) {
   int pid, status;
   int fd[2];

   pipe(fd);

   switch (pid = fork()) {

   case 0:
         runpipe(fd);
         exit(0);

   default:
         while ((pid = wait(&status)) != -1)
               fprintf(stderr, "process %d exits with %d\n", pid, WEXITSTATUS(st
         break;

   case -1:
         perror("fork");
         exit(1);
   }
   exit(0);
}

char *pippo[] = { "/usr/bin/tr", "a-z", "A-Z", 0 };
char *pluto[] = { "/bin/ls", "-al", "/", 0 };

void runpipe(int pfd[]) {
   int pid;

   switch (pid = fork()) {

   case 0:
         dup2(pfd[0], 0)
         close(pfd[1]);
         execvp(pippo[0], pippo);
         perror(pippo[0]);

   default:
         dup2(pfd[1], 1);
         close(pfd[0]);
         execvp(pluto[0], pluto);
         perror(pluto[0]);

   case -1:
         perror("fork");
         exit(1);
   }
}
```

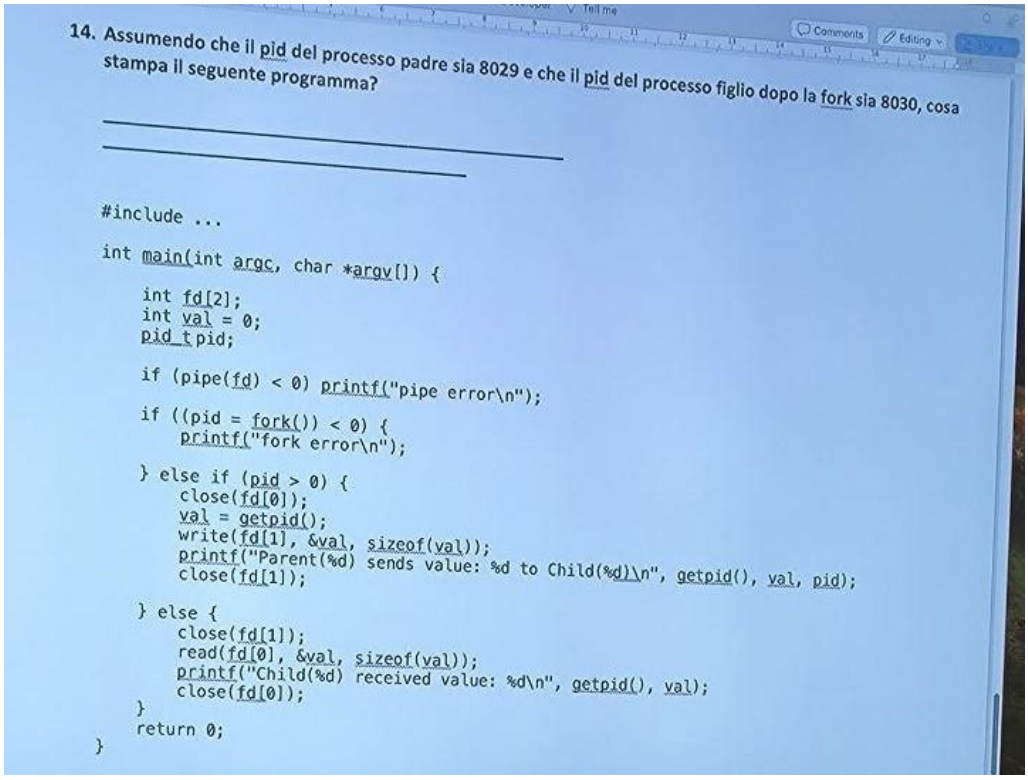
Completamento

Comando ls

12. Completare il seguente programma C che implementa una versione semplificata del comando Unix "ls -la <directory_name>".

```
#include ...
int main(int argc, char* argv[])
{
    DIR *mydir;
    struct dirent *myfile;
    struct stat mystat;

    if (argc != 2)
        err_quit("usage: filepermission <directory_name>");

    if ((mydir = _____) == NULL)
        err_sys("can't open directory %s", argv[1]);

    printf("\n");

    char path[512];
    while((myfile = _____) != NULL)
    // while((myfile = readdir(mydir)))
    {
        sprintf(path, "%s/%s", argv[1], myfile->d_name);
        stat(path, &mystat);

        // file permissions
        mode_t val;
        val = (mystat.st_mode & ~S_IFMT);
        (val & S_IRUSR) ? printf(_____) : printf("-");
        (val & S_IWUSR) ? printf(_____) : printf("-");
        (val & S_IXUSR) ? printf(_____) : printf("-");
        (val & S_IRGRP) ? printf(_____) : printf("-");
        (val & S_IWGRP) ? printf(_____) : printf("-");
        (val & S_IXGRP) ? printf(_____) : printf("-");
        (_____) ? printf("r") : printf("-");
        (_____) ? printf("w") : printf("-");
        (_____) ? printf("x") : printf("-");

        // number of hard links
        printf("\t%d", _____.st_nlink);
        // user ID of owner
        printf("\t%d", _____.st_uid);
        // group ID of owner
        printf("\t%d", _____.st_gid);
        // size in byte (for regular files)
        printf("\t%d", (int)_____.st_size);

        // last modification time
        struct tm *time_stamp = localtime(&_____.st_mtime);
        char buffer[80];

        strftime(buffer,10, "%b", time_stamp);

        printf("\t%4d %s %2d ", time_stamp->tm_year+1900, buffer, time_stamp->tm_mday);
        printf(" %s\n", myfile->d_name);
    }

    printf("\n");
    closedir(mydir);
}
```

Pipe



14. Assumendo che il pid del processo padre sia 8029 e che il pid del processo figlio dopo la fork sia 8030, cosa stampa il seguente programma?

```
#include ...

int main(int argc, char *argv[]) {

    int fd[2];
    int val = 0;
    pid_t pid;

    if (pipe(fd) < 0) printf("pipe error\n");

    if ((pid = fork()) < 0) {
        printf("fork error\n");

    } else if (pid > 0) {
        close(fd[0]);
        val = getpid();
        write(fd[1], &val, sizeof(val));
        printf("Parent(%d) sends value: %d to Child(%d)\n", getpid(), val, pid);
        close(fd[1]);

    } else {
        close(fd[1]);
        read(fd[0], &val, sizeof(val));
        printf("Child(%d) received value: %d\n", getpid(), val);
        close(fd[0]);
    }
    return 0;
}
```

Comando

20. Il seguente programma implementa una versione semplificata di quale comando di shell?
Scrivere la risposta nella riga tratteggiata qui sotto (NON DI PIU').

```
#include ...

#define BUFF_SIZE 1024

int main(int argc, char* argv[]) {

    int fd1, fd2, numb1, numb2;
    char *buff[BUFF_SIZE];

    if( argc != 3 || strcmp(argv[1], "--help") == 0 ) {
        printf("\n Usage: ./es20 <file1> <file2> \n\n");
        exit(EXIT_FAILURE);
    }

    fd1 = open(argv[1], O_RDONLY);

    if(fd1 == -1) {
        printf("\n Error opening file %s errno = %d \n", argv[1], errno);
        exit(EXIT_FAILURE);
    }

    fd2 = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR |
                S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH);

    if(fd2 == -1) {
        printf("\n Error opening file %s errno = %d \n", argv[2], errno);
        exit(EXIT_FAILURE);
    }

    while( (numb1 = read(fd1, buff, BUFF_SIZE)) > 0 ) {
        if( write(fd2, buff, numb1) != numb1 )
            printf("\n Error in writing data to %s \n", argv[2]);
    }

    if(numb1 == -1) printf("\n Error in reading data from %s \n", argv[1]);

    close(fd1);
    close(fd2);
    exit(EXIT_SUCCESS);
}
```

Signal e alarm

20. Sapendo che la funzione alarm(3) invia al processo corrente il segnale SIGALRM dopo che siano trascorsi 3 secondi, cosa fa il seguente programma. Lo spazio sottostante è più che sufficiente per dare la risposta.

```
#include...

void handler() {
    printf("Questo segnale lo gestisco io!\n");
    alarm(3);
}

int
main() {
    signal(SIGALRM, handler);
    alarm(3);
    while(1){}
}
```

Completamento

17. Completare il seguente programma C che legge dallo standard input delle stringhe separate (da [SPAZIO] oppure da [INIVIO]) e stampa sullo standard output il numero di stringhe lette.
Inoltre, il programma crea in append un file di nome "output.txt" in cui scrive tutto lo storico delle stringhe inserite. Per esempio, a fronte della sequenza di stringhe: pluto1 Pippo1 [INVIO] minnie [INIVIO] [Ctrl-D], il file output.txt conterrà:
pluto1
Pippo1
minnie
-- FINE INSERIMENTO --

```
#include ...

int main(int argc, char *argv[]) {

    int count = 0;

    _____;

    _____;

    printf("Inserisci una sequenza di stringhe alfanumeriche separate da
            [SPAZIO] oppure da [INVIO]. Terminare la sequenza con
            [INVIO][Ctrl-D] \n");

    fp = _____;

    while(_____) {

        _____;

        count_____;
    }

    fprintf(_____, _____, "-- FINE INSERIMENTO --");

    _____(fp);

    /* Di seguito si stampa sullo standard output:
        "IL NUMERO DI STRINGHE INSERITE E': <valore di count>" */
```

Thread

15. Scrivere nelle righe sotto (non usare tutte le righe) cosa stampa il seguente programma?

```
#include "apue.h"
#include <pthread.h>

void *
thr_fn1(void *arg)
{
        printf("Thread 1 returning\n");
        return((void *)arg);
}

void *
thr_fn2(void *arg)
{
        printf("Thread 2 exiting\n");
        pthread_exit((void *)arg);
}

int
main(void)
{
        int             err;
        pthread_t       tid1, tid2;
        void            *tret1, *tret2;

        err = pthread_create(&tid1, NULL, thr_fn1, (void *)2);
        if (err != 0)
            err_exit(err, "can't create thread 1");

        err = pthread_create(&tid2, NULL, thr_fn2, (void *)4);
        if (err != 0)
            err_exit(err, "can't create thread 2");

        err = pthread_join(tid1, &tret1);
        if (err != 0)
            err_exit(err, "can't join with thread 1");

        err = pthread_join(tid2, &tret2);
        if (err != 0)
            err_exit(err, "can't join with thread 2");

        printf("Final result %ld\n", (long)tret1*(long)tret2);

        exit(0);
}
```