Question 2/

```
ftestu@lovelace:~/1A/Système d'exploitation/Projet/fournitures$ ./test
>>> ls
>>> LisezMoi.html  minishell5.c  readcmd.c      test
LisezMoi.md    readcmd1.c    readcmd.h  visu_readcmd.c
ls
>>> LisezMoi.html  minishell5.c  readcmd.c      test
LisezMoi.md    readcmd1.c    readcmd.h  visu_readcmd.c
ls
>>> LisezMoi.html  minishell5.c  readcmd.c      test
LisezMoi.md    readcmd1.c    readcmd.h  visu_readcmd.c
pwd
>>> /home/ftestu/1A/Système d'exploitation/Projet/fournitures

Salut
```

Sans le wait, on voit que l'exécution se mélange avec l'affichage. En effet, on voit les «>>>» de la boucle suivante avant le résultat de la commande ls. L e wait permet d'attendre la fin du fils et donc la fin de l'exécution pour faire les choses dans le bon ordre.

(Il faut donc enlever les commentaires ligne 35 à 45 dans le code)

```
15      do {
16          printf(">>> ");
17          commande = readcmd();
18
19          sortie = (commande == NULL || commande-> seq[0] == NULL);
20
21          if (!sortie) {
22
23          pidFils = fork();
24          /* bonne pratique : tester systématiquement le retour des appels système */
25          if (pidFils == -1) {
26          NULL;
27          }
28          if (pidFils == 0) { /* fils */
29          execvp(commande->seq[0][0], commande->seq[0]);
30
31          printf ("ECHEC \n");
32          exit(2);
33          }
34
35          //else { /* père */
36          //idFils = wait(&codeTerm);
37          //if ( idFils == -1) {
38          //perror ("wait");
39          //    exit (8);
40          //}
41          //}
42
43          //if (WEXITSTATUS(codeTerm) != 2) {
44          //printf ("SUCCES \n");
45          //}
46          }
47      } while (!sortie);
48
49
50      //Affichage Salut à la sortie de la boucle
51      if (ret !=  1){
52          printf("\n");
53      }
54
55      printf("Salut \n");
56
57      return EXIT_SUCCESS; /* -> exit(EXIT_SUCCESS); pour le père */
58  }
```