Esercizi

1. Stampa “T” (per True) o “F” (per False) a seconda che il valore rappresenti un file o cartella esistente

```
# DATA è una variabile locale
if [[ -f $DATA ]]; then echo True; else echo False; fi
```

2. Stampa “file”, “cartella” o “?” a seconda che il valore rappresenti un file (esistente), una cartella (esistente) o una voce non presente nel file-system

```
# DATA è una variabile locale
if [[ -f $DATA ]]; then echo file; elif [[ -d $DATA ]]; then echo cartella;
else echo ?; fi
```

3. Stampa il risultato di una semplice operazione aritmetica (es: ‘1 < 2’) contenuta nel file indicato dal valore di DATA, oppure “?” se il file non esiste

```
# DATA è una variabile locale
if [[ -f $DATA ]]; then if [[ $(cat $DATA) ]]; then echo True; else echo
False; fi; else echo ?; fi
```

4. Scrivere uno script che dato un qualunque numero di argomenti li restituisca in output in ordine inverso.

```
# bash ./es4.sh 1 2 3 4
lista = ()

while [[ $1 != "" ]]; do
    lista[${#lista[@]}]=$1
    shift
done

for((i=${#lista[@]}-1; i>=0; i--)); do
    echo ${lista[$i]}
done
```

5. Scrivere uno script che mostri il contenuto della cartella corrente in ordine inverso rispetto all’output generato da “ls” (che si può usare ma senza opzioni)

```
# bash ./es5.sh
lista=($(ls))

for((i=${#lista[@]}-1; i>=0; i--)); do
    echo ${lista[$i]}
done
```

6. Creare un makefile con una regola help di default che mostri una nota informativa, una regola backup che crei un backup di una cartella appendendo ".bak" al nome e una restore che ripristini il contenuto originale. Per definire la cartella sorgente passarne il nome come variabile, ad esempio: make -f mf-backup FOLDER=... (la variabile FOLDER è disponibile dentro il makefile)

```
# make help
# make FOLDER="test" backup
# make FOLDER="test" restore

help:
    @echo "Nota informativa"

backup:
    @cp -r $FOLDER "$(basename ${FOLDER}).back"

restore:
    @rm -r $FOLDER
    @mv "$(basename ${FOLDER}).back" $FOLDER
```

7. creare un "alias" in bash denominato "feedback" che se invocato attende dall'utente un input proponendo il prompt "Come ti chiami?" e rispondendo con "Ciao !" (dove è l'input immesso) senza però sovrascrivere o impostare alcuna nuova variabile nella shell attiva.

```
alias feedback="read -p \"Come ti chiami? \" var && echo Ciao $var"
```

8. creare un "alias" in bash denominato "somma" che legge un numero (intero con segno) alla volta (numero+INVIO, numero+INVIO, ...) e alla fine (immissione vuota premendo solo INVIO) stampa la somma dei numeri inseriti.

```
alias somma='sum=0 && while true; do read var; if [[ -z ${var} ]]; then echo
$sum; break; else sum="$sum + $var"; fi; done'
```

9. Fare un programmino in C "charcount" che accetta come parametro un carattere e ne conta il numero di occorrenze nel flusso in ingresso. Ad esempio con ls -alh /tmp | ./charcount "x" devono essere contate le "x" presenti. OPZIONALMENTE: aggiungere la ristampa dei dati in ingresso mostrando il carattere colorato.

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

int main(int argc, char** argv)
{
        if(argc!=2)
        return -1;

        char c1 = argv[1][0];
        char c2;

        int count = 0;
```

```
        while((c2 = getchar()) != EOF)
        {
                if(c1==c2)
                {
                        printf("\033[0;32m%c\033[0;0m",c2);
                        count++;
                }
                else
                printf("%c",c2);
        }

        printf("\nIl carattere %c compare %d volte\n",c1,count);
}
```

10. Scrivere un'applicazione che data una stringa come argomento ne stampa a video la lunghezza, ad esempio: ./lengthof "Questa frase ha 28 caratteri" deve restituire a video il numero 28.

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

int main(int argc, char **argv)
{
    if(argc==1)
    return -1;

    int len=0;

    while(argv[1][len] != '\0')
    len++;

    printf("Questa stringa ha %d caratteri\n",len);
    printf("Questa stringa ha %d caratteri\n",strlen(argv[1]));
}
```

11. Scrivere un'applicazione che definisce una lista di argomenti validi e legge quelli passati alla chiamata verificandoli e memorizzando le opzioni corrette, restituendo un errore in caso di un'opzione non valida.

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

#define TRUE 1
#define FALSE 0
#define MAX_PARAMS 4

typedef char bool;
const char * valid[] = {
    "-h",
    "-m",
    "-n",
    "--help"
};
```

```
//Return TRUE is parameter is valid, FALSE otherwise
bool isValid(char * arg)
{
    for (int i = 0; i < 4; i++) { // Cycle through all of the valid
parameters
        //Compare the i-th element of the valid list with the parameter
        if (!strcmp(arg, valid[i])) {
            return TRUE;
        }
    }
    return FALSE;
}

int main(int argc, char ** argv)
{
    char * paramList[MAX_PARAMS]; //Define list of stored parameters
    int ind = 0;
    for (int i = 1; i < argc; i++) { //Cycle through parameters
        if (isValid(argv[i])) { //Check if parameter is valid
            paramList[ind++] = argv[i]; //Save valid parameter
        } else {
            fprintf(stdout, "'%s' is a invalid parameter\n", argv[i]);
            exit(2);
        }
    }

    //Print list of recognised parameters
    for (int i = 0; i < ind; i++) {
        printf("%s\n", paramList[i]);
    }
    return 0;
}
```

12. Realizzare funzioni per stringhe char *stringrev(*char str) (inverte ordine caratteri) e int stringpos(*char str, char chr) (cerca chr in str e restituisce la posizione)

```
#include <stdio.h>

char* stringrev(char *str)
{
    int len=0;

    while(str[len]!='\0')
    len++;

    char nuovo_array[len];

    for(int i=len-1; i>=0; i--)
    nuovo_array[len-1-i] = str[i];

    for(int i=0; i<len; i++)
    str[i] = nuovo_array[i];

    return str;
```

```
}

int stringpos(char *str, char chr)
{
    int len=0;

    while(str[len]!='\0')
    len++;

    for(int i=0; i<len; i++)
    {
        if(str[i] == chr)
        return i;
    }

    return -1;
}

int main(int argc, char** argv)
{
    switch(argc)
    {
        case 0:
        return -1;
        break;

        case 1:
        printf("%s\n",stringrev(argv[0]));
        break;

        case 2:
        printf("%s\n",stringrev(argv[1]));
        break;

        case 3:
        printf("%s\n",stringpos(argv[1],argv[2][0]));
        break;
    }
}
```

13. Avendo come argomenti dei “binari”, si eseguono con exec ciascuno in un sottoprocesso

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

int main(int argc, char** argv)
{
    for(int i=1; i<argc; i++)
    {
        if(!fork())
        {
            char *args[] = {NULL};
            execvp(argv[i],args);
        }
```

```
    }
}
```

14. idem punto 1 ma in più salvando i flussi di stdout e stderr in un unico file

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

int main(int argc, char** argv)
{
    for(int i=1; i<argc; i++)
    {
        if(!fork())
        {
             int outfile = open("output.txt", O_RDWR | O_CREAT, S_IRUSR |
S_IWUSR);
             dup2(outfile,1);
             dup2(outfile,2);
             close(outfile);

            char *args[] = {NULL};
            execvp(argv[i],args);
        }
    }
}
```

15. Dati due eseguibili come argomenti del tipo ls e wc si eseguono in due processi distinti: il primo deve generare uno stdout redirezionato su un file temporaneo, mentre il secondo deve essere lanciato solo quando il primo ha finito leggendo lo stesso file come stdin. Ad esempio ./main ls wc deve avere lo stesso effetto di ls | wc.

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

#define MAX 1000

int main(int argc, char** argv)
{
    if(argc!=3)
    return -1;

    char working_dir[MAX];
    getcwd(working_dir,MAX);

    if(!fork())
    {
        char *args[] = {working_dir,NULL};

        int outfile = open("output.txt", O_RDWR | O_CREAT, S_IRUSR |
S_IWUSR);
```

```
            dup2(outfile,1);
            close(outfile);

            execvp("ls",args);
            return 0;
        }

        while(wait(NULL)>0);

        if(!fork())
        {
            int outfile = open("output.txt", O_RDWR | O_CREAT, S_IRUSR |
S_IWUSR);
            dup2(outfile,0);
            close(outfile);

            char c;
            char buf[MAX];

            int index = 0;

            while((c = getchar()) != EOF)
            {
               buf[index] = c;

               index++;

               if(index == MAX)
               {
                   index--;
                   buf[index] = '\0';
                   break;
               }
            }

            if(index == MAX)
            {
                index--;
                buf[index] = '\0';
            }

            printf("%s\n",buf);

            char *args[] = {buf,NULL};

            execvp("wc",args);
            return 0;
            /* DA FINIRE */
        }
    }
```

16. Impostare una comunicazione bidirezionale tra due processi con due livelli di complessità: ○ Alternando almeno due scambi (P1 → P2, P2 → P1, P1 → P2, P2 → P1). Estendendo il caso a mo' di "ping-pong", fino a un messaggio convenzionale di "fine comunicazione"

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <string.h>

#define READ 0
#define WRITE 1

#define MAX 100

int main()
{
        int pipe1[2];
        int pipe2[2];

        pipe(pipe1);
        pipe(pipe2);

        char buf[MAX];

        if(fork())
        {
                close(pipe1[READ]);
                close(pipe2[WRITE]);

                while(1)
                {

                        printf("Process1: ");
                        fgets(buf,MAX,stdin);
                        fflush(stdin);

                        write(pipe1[WRITE],buf,MAX);

                        if(buf[0]=='\n')
                        break;

                        read(pipe2[READ],&buf,MAX);

                        if(buf[0]=='\n')
                        break;

                        printf("-> Process2: %s\n",buf);
                }

                close(pipe1[WRITE]);
                close(pipe2[READ]);
        }
        else
        {
                close(pipe1[WRITE]);
                close(pipe2[READ]);

                while(1)
                {
```

```
                        read(pipe1[READ],&buf,MAX);

                        if(buf[0]=='\n')
                        break;

                        printf("-> Process1: %s\n",buf);

                        printf("Process2: ");
                        fgets(buf,MAX,stdin);
                        fflush(stdin);

                        write(pipe2[WRITE],buf,MAX);

                        if(buf[0]=='\n')
                        break;
                }

                close(pipe1[READ]);
                close(pipe2[WRITE]);
        }

        while(wait(NULL)>0);
}
```