# Laboratorio di Sistemi Operativi
# 30 Gennaio 2019
### Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (4 punti) Si scriva una pipeline che, dato un file di testo con nome `test.txt`, stampi le sue linee, evitando le ripetizioni di linee che occorrano più volte, e (prima di ogni linea) il conteggio delle sue occorrenze nel file. Ad esempio, supponendo che il file `test.txt` sia formato dalle seguenti linee:

   ```
   abc
   abc
   def
   abc
   ghi
   ghi
   ghi
   abc
   ```

   la pipeline dovrà produrre quanto segue:

   ```
   4 abc
   1 def
   3 ghi
   ```

   ```
   sort test.txt | uniq -c
   ```

2. (6 punti) si scriva uno script `dim.sh` della shell che legga delle linee di testo dallo standard input fintanto che l'utente non digiti una linea di terminazione composta dalla sola parola `fine`. Per ogni linea diversa da quella di terminazione, lo script deve stampare a video la dimensione in byte della linea ed una serie di asterischi di lunghezza pari alla dimensione.

   Esempio:

   ```
   > ./dim.sh
   abc
   3 byte
   ***
   ciao, mondo!
   12 byte
   ************
   fine
   > _
   ```

   ```
   1  while true
   2  do
   3    read linea
   4
   5    if test $linea = 'fine'
   6    then
   7      exit 0
   8    else
   9      dim=`echo $linea | wc -c`
   10     dim=$[$dim-1] # escludo dal conteggio il newline
   11     i=0;
   12
   13     while test $i -lt $dim
   14     do
   ```

```
15        echo -n '*'
16        i=$[$i+1]
17      done
18
19      if test $dim -gt 0
20      then
21        echo
22      fi
23
24    fi
25
26  done
```

3. (8 punti) Data la seguente struttura dati ricorsiva `Node`:

```
struct node_t {
  int data;
  struct node_t *next;
};

typedef struct node_t Node;
```

si scriva un programma C che allochi spazio in memoria ed inizializzi una lista finita di naturali con i seguenti valori:

`11, 6, 30`

Si scriva infine una procedura ricorsiva per stampare a video i valori dei nodi di una lista di naturali, partendo dal puntatore alla testa della lista.

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

struct node_t {
  int data;
  struct node_t *next;
};

typedef struct node_t Node;

void print_list(Node *h);

int main() {
  Node *head=NULL;
  head=(Node*)malloc(sizeof(Node));
  head->data=11;
  head->next=(Node*)malloc(sizeof(Node));
  head->next->data=6;
  head->next->next=(Node*)malloc(sizeof(Node));
  head->next->next->data=30;
  head->next->next->next=NULL;
  print_list(head);
  return 0;
```

```
}

void print_list(Node *h) {
  if(h!=NULL) {
    printf(" %d ",h->data);
    print_list(h->next);
  }
}
```

4. (6 punti) Scrivere l'output del seguente programma C.

```c
#include <stdio.h>

int main()
{
    int levels=5;

    for (int i = 0; i < levels; i++) {
        for (int j = 0; j < levels - i; j++){
            printf("-");
        }
        for (int k = 0; k < (2 * i + 1); k++){
            if(i>0 && i<levels-1) {
                    if(k>0 && k < (2 * i + 1) - 1)
                            printf(" ");
                    else
                            printf("*");
            }
            else
                printf("*");
        }
        printf("\n");
    }
    return 0;
}
```

```
-----*
----* *
---*   *
--*     *
-*********
```

5. (8 punti) La procedura `logger` seguente è la start routine di una famiglia di POSIX thread che accedono concorrentemente in scrittura al file `log.txt`.

```c
struct channel {
  int fd;  /* file descriptor del canale di comunicazione */
  int num; /* numero intero identificante il thread */
};

void *logger(void *c) {
  char inputline[LINESIZE];
  char buffer[2*LINESIZE];
```

```
9   int  len;
10  int fd=((struct channel *)c)->fd;
11  int num=((struct channel *)c)->num;
12  FILE *logfd;
13   /* finché ricevo linee da fd... */
14  while ((len = recv(fd, inputline, LINESIZE-1, 0)) > 0) {
15    inputline[len]='\0';          /* imposto il terminatore */
16    sprintf(buffer,"Thread n. %d: ",num); /* scrivo il n. del thread in buffer */
17    strcat(buffer,inputline);     /* concateno in buffer quanto ricevuto da fd */
18    logfd=fopen("log.txt","a");   /* apro il file log.txt in append */
19    fputs(buffer,logfd);          /* scrivo nel file */
20    fclose(logfd);                /* chiudo il file log.txt */
21  }
22
23  close(fd);
24 }
```

(a) C'è il pericolo che si verifichino delle race condition, con un possibile accavallamento delle linee di un thread con quelle di altri thread nel file `log.txt`?

(b) In caso di risposta affermativa alla domanda precedente, specificare come modificare la routine in modo da garantire l'integrità del file condiviso, aggiungendo le dichiarazioni ed i comandi necessari.

(a) Sì, c'è il pericolo che si verifichino delle race condition, con la conseguente "corruzione" del contenuto di `log.txt` in quanto tra le scritture nel file da parte di un thread potrebbero intervenire altre operazioni di scrittura da parte di altri thread della stessa famiglia, accavallando le linee.

(b) Bisogna introdurre un mutex globale:

```
pthread_mutex_t mutex=PTHREAD_MUTEX_INITIALIZER;
```

Dopodiché bisogna proteggere il ciclo `while` nel modo seguente:

```
void *logger(void *c) {
  char inputline[LINESIZE];
  char buffer[2*LINESIZE];
  int  len;
  int fd=((struct channel *)c)->fd;
  int num=((struct channel *)c)->num;
  FILE *logfd;

  pthread_mutex_lock(&mutex);       /* inizio sezione critica */

  while ((len = recv(fd, inputline, LINESIZE-1, 0)) > 0) {
    inputline[len]='\0';
    sprintf(buffer,"Thread n. %d: ",num);
    strcat(buffer,inputline);
    logfd=fopen("log.txt","a");
    fputs(buffer,logfd);
    fclose(logfd);
  }

  pthread_mutex_unlock(&mutex);     /* fine sezione critica */

  close(fd);
}
```