Scritto 15 Gennaio 2026

ESERCIZIO 1

Scrivere una funzione

```
bool only_primes(int A[], int length)
```

che prende in input un array `A` di interi di lunghezza `length` e ritorna vero se l'array contiene soltanto numeri primi, falso altrimenti.

È possibile usare funzioni ausiliarie, purchè definite.

Per esempio, se `A` è {5, 7, 3, 11, 3, 13} la funzione restituisce `true`, invece per un l'array {3, 7, 9, 11, 13} la funzione restituisce `false`.

ESERCIZIO 2

Un **albero ternario di interi** (albero i cui nodi contengono interi e possono avere al massimo tre figli) è definito dal seguente tipo:

```
struct tree {
  int val;
  tree *ltree;
  tree *ctree;
  tree *rtree;
};

typedef tree *ptree ;
```

(A) Scrivere una funzione

```
ptree ammuina(ptree T)
```

che prende un albero ternario `T` e lo modifica come segue:

1. per ogni nodo che contiene un intero pari, il sottoalbero di sinistra diventa quello di mezzo, il sottoalbero di mezzo diventa quello di destra e quello di destra diventa il sottoalbero di sinistra (si spostano tutti verso sinistra);
2. per ogni nodo che contiene un intero dispari, il sottoalbero di sinistra diventa quello di destra, il sottoalbero di mezzo diventa quello di sinistra e quello di destra diventa il sottoalbero di mezzo (si spostano tutti verso destra).

Per esempio, l'albero in figura 1 diventa quello in figura 2.

(B) Scrivere una funzione

```
ptree upgrade10(ptree T)
```

che prende un albero ternario `T` e:

1. rimuove da `T` tutte le foglie il cui intero è minore di 10;
2. le foglie in `T` con un valore maggiore a uguale a 10, vengono aggiornate in modo che il loro valore sia incrementato di 10.

Per esempio, l'albero in figura 2 diventa l'albero in figura 3.

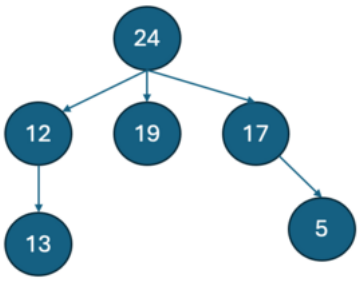

Figure 1: Albero iniziale

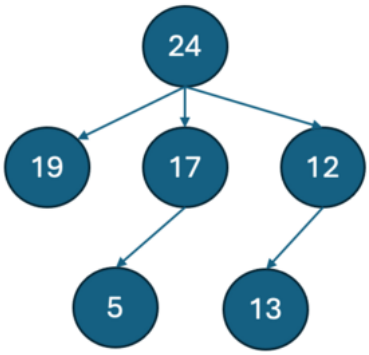

Figure 2: Albero dopo ammuina

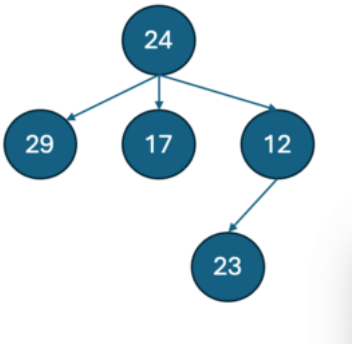

Figure 3: Albero della Figura 2 dopo upgrade10

ESERCIZIO 3

Definire una classe

```
class Abitazione;
```

con i seguenti metodi:

- un costruttore Abitazione(int n, double m) che prende in input numero di stanze n e la metratura m;
- double costo_abitazione(double prezzo) che prende il prezzo al metro-quadro e ritorna il costo dell'abitazione. Per esempio, se abbiamo un'abitazione con metratura 80, costo_abitazione(1000) restituisce 80000.

Definire quindi due sottoclassi di Abitazione

```
class Appartamento;
class Villetta;
```

- con i relativi costruttori Appartamento(int n, double m, int p) e Villetta(int n, double m) (nel caso della classe appartamento, il costruttore prende in input anche il piano p);
- la classe Appartamento ha un metodo int num_piano() che ritorna il piano dell'appartamento;
- il metodo costo_abitazione, nel caso di villetta, ritorna il costo maggiorato del 20% (con metratura 80, costo_abitazione(1000) restituisce 96000); nel caso di Appartamento, quando il piano è superiore al 5^, il costo è maggiorato del 10% (con metratura 80 costo_abitazione(1000) sarebbe 88000).

```
class Abitazione
{
public:
  Abitazione(int n, double m);
  double costo_abitazione(double p);
};

class Villetta: public Abitazione
{
public:
  Villetta(int n, double m);
  double costo_abitazione(double p);
};

class Appartamento: public Abitazione
{
public:
  Appartamento(int n, double m, int p);
  double costo_abitazione(double p);
  int num_piano();
};
```