## Esercizio 1

Data la seguente porzione di programma rispondere alle domande corrispondenti:

```
void func(char& a, char b){
   char c = a;
   a = b;
   b = c;
}
int main()
{
   char* nome = new char[3]{..i primi tre caratteri diversi del tuo nome..};
  //1. La seguente istruzione è corretta? Se sì, cosa stampa?
         cout << *(nome[1]) << endl;
  //2. La seguente istruzione è corretta? Se sì, cosa stampa?
         cout << *(nome + 2) << endl;
  //3. La seguente istruzione è corretta? Se sì, cosa stampa?
         char* a = &nome[2];
         cout << *(a - 1) << endl;
  //4. Cosa viene stampato dalla seguente porzione di codice?
         char* nome_prof = new char[3]{'m','a','t'};
         func (nome_prof[0], nome_prof[1]);
         cout << nome_prof[0] << " " << nome_prof[1] << endl;
}
```

## Esercizio 2

Definire una classe FakeList che erediti opportunamente da list<int> e implementi i seguenti metodi:

1. int getFakeSize() const: che restituisce la dimensione della lista raddoppiata;
2. void insert(int el): che inserisce el in coda se el è già presente nella lista, in testa altrimenti;
3. void fakeSort(bool s): che ordina la lista in ordine crescente se s è true, in ordine decrescente altrimenti;
4. void fakeClear(bool  c): che svuota la lista se c è true e non fa nulla altrimenti.

La traccia richiede esplicitamente che dal main non sia possibile invocare i metodi propri della classe list su oggetti di tipo FakeList.
Ad esempio, nel main non deve essere possibile eseguire:

```
FakeList l;
cout << l.size();
```

Nell'implementazione fare in modo che, se la classe FakeList viene ulteriormente estesa tramite ereditarietà, le classi figlie di FakeList possano comunque accedere ai metodi di list<int>.

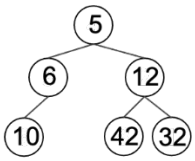
## Esercizio 3

Scrivere una funzione **esercizio3** che prenda in input un albero binario di interi A e un intero x, e restituisca in output il cammino dal nodo radice dell'albero binario A al nodo foglia contenente il valore informativo x. Se non esiste un nodo foglia contenente il valore informativo x, la funzione deve restituire in output -1.

L'albero è rappresentato da una classe AlberoB. Sia `a` una istanza della classe, l'interfaccia è la seguente:

- `a.figlio(SIN/DES)` restituisce il figlio sinistro o destro dell'albero `a` se esiste, altrimenti restituisce la costante `NULLO`;
- `a.padre()` restituisce il padre dell'albero `a` se esiste, altrimenti restituisce la costante `NULLO`;
- `a.radice()` restituisce il valore informativo associato all'albero `a` (il valore intero contenuto all'interno del nodo);
- `a.foglia()` restituisce `true` se l'albero `a` è una foglia, altrimenti `false`.
- `a.nullo()` restituisce `true` se l'albero `a` è nullo, altrimenti `false`.

Si può assumere che nell'albero non vi siano due nodi con lo stesso valore informativo.

| | |
|---|---|
| *Esempio*: dati in input l'albero qui d'esempio e x = 42, la funzione deve restituire il cammino 5, 12, 42. Se avessimo x = 37, la funzione deve restituire -1 poiché non esiste un nodo contenente il valore informativo 37. Se avessimo x = 12, la funzione deve restituire -1 poiché esiste un nodo con il valore informativo 12 ma esso non è un nodo foglia. |  |

## Esercizio 4

Scrivere una funzione **esercizio4** che prenda in input un insieme di stringhe A (di cardinalità n) e un insieme di triple ordinate C dove ogni tripla contiene elementi distinti di A. La funzione deve restituire `YES` se è possibile associare ad ogni elemento $a \in A$ un numero da 1 ad n, di seguito indicato con f(a), tale che le seguenti condizioni siano vere:

- non ci sono due o più elementi con lo stesso numero, quindi f(a) != f(b) per ogni $a, b \in A$, e
- per ogni tripla $(x, y, z) \in C$, è vero che $f(x) < f(y) < f(z)$ oppure $f(z) < f(y) < f(x)$.

Se non è possibile creare una associazione che renda vere queste condizioni, la funzione deve restituire `NO`.

Si può assumere che:

- A sia rappresentato come un `vector<string>`
- C sia rappresentato come un `vector<Triple>` dove Triple è una classe già definita che ha tre campi pubblici `x`, `y`, `z` rappresentanti rispettivamente gli elementi (stringhe) della tripla,
- Non esistano due o più triple con gli elementi nello stesso ordine.

*Esempio utilizzo Triple:* sia `t` una istanza di `Triple` con gli elementi (`abc, def, ghi`), allora `t.x == abc`, `t.y == def`, e `t.z == ghi`.

| | |
|---|---|
| *Esempio*: in questo caso la funzione restituirà `YES` poiché è possibile associare ad ogni elemento di A un numero da 1 a 5 tale che le condizioni di cui sopra siano rispettate.<br>In particolare, supponiamo di aver associato i seguenti numeri: a → 2, b → 1, c → 3, d → 5, e → 4. La prima condizione è soddisfatta (non ci sono due elementi con lo stesso numero); la seconda condizione è anch'essa soddisfatta, poiché:<br>• per la tupla (a, e, d) vale f(a) < f(e) < f(d),<br>• per la tupla (b, c, d) vale f(b) < f(c) < f(d),<br>• per la tupla (c, a, b) vale f(b) < f(a) < f(c),<br>• per la tupla (d, e, c) vale f(c) < f(e) < f(d). | n = 5<br>A = { a, b, c, d, e }<br>C = { (a, e, d), (b, c, d),<br>(c, a, b), (d, e, c) } |