

Esercizio 1

Data la seguente porzione di programma rispondere alle domande corrispondenti:

```
int main()
{
    int* matricola = new int[6]{..la tua matricola..}; //scrivi sul foglio la tua matricola
    //1. La seguente istruzione è corretta? Se sì, cosa stampa?
        cout << *(matricola + 3) << *(matricola + 2) << endl;
    //2. La seguente istruzione è corretta? Se sì, cosa stampa?
        cout << *(matricola[0]) << endl;
    //3. Cosa viene stampato dalla seguente porzione di codice?
        int& a = matricola[4];
        int b = matricola[5];
        --a;
        b += 1;
        cout << matricola[4] << " " << matricola[5] << endl;
    //4. Come andrebbe deallocata la memoria dinamica allocata inizialmente?
    //A
        for(int i = 0; i < 6; i++)
            delete matricola[i];
    //B
        for(int i = 0; i < 6; i++)
            delete *matricola[i];
    //C
        Nel main non serve deallocare la memoria dinamica.
    //D
        delete [] matricola;
}
```

Esercizio 2

Progettare una classe `GestoreValori` che sia in grado di gestire una sequenza di coppie (int-char). In particolare, la classe `GestoreValori` deve implementare almeno i seguenti metodi:

1. `inserisciCoppia(int a, char b)` che salvi il nuovo intero `a` e il nuovo char `b`; si noti che in generale ad uno stesso carattere possono essere associati più interi
2. `numCoppie()` che restituisca il numero di coppie (int-char) attualmente presenti;
3. `getRisultato()` che restituisca -1.

Implementare, successivamente, altre due classi: `GestoreValoriA` e `GestoreValoriB`. In particolare, le due classi devono estendere opportunamente `GestoreValori` e reimplementare, facendo uso del **polimorfismo**, il metodo `getRisultato()`.

In particolare:

1. Il metodo `getRisultato()` in `GestoreValoriA` deve restituire la somma di tutti gli interi inseriti accoppiati al carattere 'a'. Se non è presente alcun valore associato al carattere 'a', restituire 0.
2. Il metodo `getRisultato()` in `GestoreValoriB` deve calcolare la media (intera) degli interi inseriti accoppiati al carattere 'b'. Se non è presente alcun valore associato al carattere 'b', restituire 0.

La scelta delle classi, dei campi e dei metodi da utilizzare ed implementare, oltre a quelli richiesti dalla traccia, è libera. Verrà valutata, oltre che la correttezza del programma, anche la capacità di progettazione e l'efficienza/chiarità della soluzione.

Esercizio 3

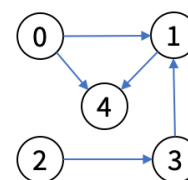
Scrivere una funzione **esercizio3** che prenda in input un grafo orientato G e restituisca il nodo che non è raggiunto dalla maggior parte degli altri nodi. Ovvero, sia $f(u)$ una funzione che restituisce il numero di nodi in G che non raggiungono il nodo u (cioè il numero dei nodi per il quale non esiste un cammino verso u). Dati tutti i nodi u in G , si deve restituire il nodo con valore $f(u)$ massimo ed etichetta (numero nodo) minima.

Il grafo è rappresentato da una classe Grafo con la seguente interfaccia (con g un'istanza della classe):

- $g.n()$ restituisce il numero di nodi del grafo,
- $g.m()$ restituisce il numero di archi del grafo,
- $g(i,j)$ restituisce *true* se esiste l'arco diretto tra il nodo i e il nodo j .

I nodi sono etichettati da 0 a $g.n()-1$. Se esistono più nodi con lo stesso valore $f(u)$ massimo, restituire quello con l'etichetta minore.

Esempio: il nodo 0 non è raggiunto da 4 nodi (quindi $f(0)=4$), il nodo 1 non è raggiunto da un nodo (non è raggiunto solo dal nodo 4, quindi $f(1)=1$), il nodo 2 non è raggiunto da 4 nodi ($f(2)=4$), il nodo 3 non è raggiunto da 3 nodi ($f(3)=3$), il nodo 4 è raggiunto da tutti i nodi ($f(4)=0$). I nodi con il numero più alto di nodi che non li raggiungono sono i nodi 0 e 2. La funzione restituisce 0 (il nodo con etichetta minore tra quelli non raggiunti dalla maggior parte dei nodi).



Esercizio 4

Scrivere una funzione **esercizio4** che prenda in input un grafo diretto $G(V,E)$, e due interi $1 \leq k_1 \leq k_2 \leq |V|$, e restituisca *true* se esiste un sottoinsieme U di V tale che le seguenti condizioni sono soddisfatte:

- U contiene almeno k_1 ed al più k_2 nodi, ovvero $k_1 \leq |U| \leq k_2$ e ,
- per ogni arco (i,j) in E se i è in U allora j non è in U ,

Se tale sottoinsieme non esiste, la funzione restituisce *false*.

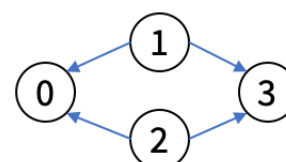
Il grafo è rappresentato da una classe Grafo con la seguente interfaccia (con g un'istanza della classe):

- $g.n()$ restituisce il numero di nodi del grafo,
- $g.m()$ restituisce il numero di archi del grafo,
- $g(i,j)$ restituisce *true* se esiste l'arco diretto tra il nodo i e il nodo j .

I nodi sono etichettati da 0 a $g.n()-1$.

Esempio: in questo caso la funzione restituirà *true* poiché è possibile creare un sottoinsieme U che rispetti le condizioni di cui sopra. Una possibile soluzione è data dal sottoinsieme $U = \{1,2\}$ il quale

- contiene almeno k_1 nodi e non più di k_2 ,
- per ogni arco (i,j) : se i è in U j non è in U ,



$k_1 = 2$ $k_2 = 3$