

Esercizio 1

Data la seguente porzione di programma rispondere alle domande corrispondenti:

```
int main() {  
    //scrivi sul foglio la tua matricola  
    int* matricola = new int[6]{..la tua matricola..};  
  
    // A: La seguente istruzione è corretta? Se sì, cosa stampa?  
    cout << matricola[matricola[0]%6] << endl;  
  
    // B: La seguente istruzione è corretta? Se sì, cosa stampa?  
    cout << 9 - *matricola << endl;  
  
    // C: La seguente lista di istruzioni è corretta? Se sì, cosa stampa?  
    int & a=matricola[0];    matricola[0]=10; a=15; cout<<matricola[0];  
  
    // D: Qual è il modo opportuno di gestire la memoria dinamica aggiungendo la seguente  
    istruzione al programma?  
    int* p = new int[matricola[0]];  
  
    // 1: Dobbiamo deallocare sia p che matricola  
    // 2: Non dobbiamo deallocare né p né matricola  
    // 3: Dobbiamo deallocare p, ma non matricola  
    // 4: Dobbiamo deallocare matricola, ma non p  
}
```

Esercizio 2

Definire una classe **BufferCircolare** che modelli una struttura dati che consenta il seguente funzionamento:

```
BufferCircolare bc;  
bc.aggiungi(8); bc.aggiungi(4); bc.aggiungi(2);  
for (int i = 0; i < 10; ++i) {  
    cout << bc.elemento_corrente()<<" ";  
    bc.avanza();  
} // Il for deve stampare: 8 4 2 8 4 2 8 4 2 8
```

```
bc.rimuovi(4);  
for (int i = 0; i < 10; ++i) {  
    cout << bc.elemento_corrente()<<" ";  
    bc.avanza();  
} // Il for deve stampare: 2 8 2 8 2 8 2 8 2 8 perchè il precedente for si era fermato sul 4
```

La classe dovrà implementare, oltre al costruttore, i seguenti metodi mostrati nell'esempio:

- `void aggiungi(int x)`: aggiunge l'elemento `x` al BufferCircolare
- `void rimuovi(int x)`: rimuove il primo elemento di valore `x` dal BufferCircolare
- `int elemento_corrente() const`: restituisce l'elemento corrente del BufferCircolare
- `void avanza()`: assegna l'elemento corrente del BufferCircolare al successivo

Non è nota a priori la dimensione massima della struttura dati. Non è possibile utilizzare per l'implementazione metodi o classi della libreria standard. L'implementazione dovrà fare uso (e gestire opportunamente) la memoria dinamica attraverso la gestione di un campo privato `int* buffer`.

Esercizio 3

Sia G un grafo diretto. Scrivere una funzione che presi in input un grafo G , un insieme di nodi X , un insieme di nodi Y e un nodo di partenza z , restituisca *true* se e solo se dal nodo z è possibile raggiungere tutti i nodi in X e nessuno dei nodi in Y .

La funzione dovrà avere segnatura:

bool funzione(const Grafo& G, const vector<unsigned>& X, const vector<unsigned>& Y, unsigned z);

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 .

N.B. non è possibile assumere già implementati gli algoritmi di visita di un grafo.

Esercizio 4

Siamo stati incaricati di organizzare una partita di pallamano. Per rendere la partita più interessante e spettacolare, dobbiamo assicurarci che le squadre possano essere bilanciate, cioè che sia possibile costruire due squadre la cui differenza di abilità sia minore di k . L'abilità di una squadra è la somma dell'abilità dei suoi giocatori, dove l'abilità di un giocatore è un intero positivo che determina la sua competenza nel gioco.

Scrivere una funzione che, preso in input un *vector<unsigned>* il cui contenuto rappresenta l'abilità dei giocatori disponibili e un intero positivo k che rappresenta la massima differenza di abilità tra le squadre, determini se è possibile distribuire i giocatori in due squadre in modo tale che l'abilità delle squadre risultanti differisca di al più k .

- Ogni giocatore disponibile dovrà essere assegnato esattamente ad una delle due squadre
- Non ci sono vincoli sul numero minimo di giocatori per squadra
- Le due squadre possono avere un diverso numero di giocatori

NB: La tecnica risolutiva adottata influenza la valutazione dell'esercizio. Verrà assegnato un bonus a chi utilizzerà la tecnica di programmazione dinamica.