Fondamenti di Programmazione 2
(Corso di Laurea in Informatica)
Esame del 5/09/2024

UNIVERSITÀ DELLA CALABRIA

## Esercizio 1

Data la seguente porzione di programma rispondere alle domande corrispondenti:

```
int main() {
int* matricola = new int[6]{...inserisci la tua matricola…};

// 1. La seguente istruzione è corretta? Se sì, cosa stampa? Altrimenti, perché?
cout << matricola->back() << endl;

// 2. La seguente istruzione è corretta? Se sì, cosa stampa? Altrimenti, perché?
int x = matricola[3];
matricola[3] = 8;
cout << x << " " << matricola[3] << endl;

int& p = matricola[3];
// 3. La seguente istruzione è corretta? Se sì, cosa stampa? Altrimenti, perché?
cout << *p << endl;

int* nuova = matricola;
// 4. Qual è il modo corretto di gestire la memoria dinamica istanziata nel main?
Argomentare.
// A: delete nuova; delete[] matricola;
// B: delete nuova;
// C: delete[] matricola;
// D: La risposta corretta non è tra le precedenti
}
```

## Esercizio 2

Data una classe **Alimento** con la seguente interfaccia pubblica (i metodi dichiarati nell'interfaccia seguente si possono considerare già implementati):

```
class Alimento {
   string nome() const; // (es. "carote","latte","bistecca","cioccolato")
   string tipo() const; // (es. "frutta","verdura","carne","latticini")
   unsigned kcal() const;
   bool sano() const;
}
```

Si richiede di modellare e implementare opportunamente una classe **astratta** **Frigorifero**, che funzioni secondo i seguenti principi:

- Un frigorifero contiene dei ripiani, ciascuno dei quali ha il nome del tipo di Alimento che conterrà (es. frutta,verdura,carne,latticini).
- È possibile aggiungere alimenti al frigorifero, mediante un metodo *void aggiungiAlimento(const Alimento& a)*. La funzione dovrà aggiungere *a* nel ripiano corretto in base al *tipo* di alimento.
- È possibile *estrarre* un alimento da un determinato ripiano tramite un metodo *Alimento estrai (string ripiano).* La funzione per la classe Frigorifero **non** deve avere un'implementazione di default, rendendo la classe Frigorifero astratta.

Implementare quindi una classe **FrigoriferoSalutista**, che estende opportunamente la classe astratta **Frigorifero**, dove il metodo *estrai* restituisce l'alimento nel ripiano con il più basso apporto calorico e, a parità, preferisce un cibo "sano". L'alimento restituito va rimosso dal ripiano.
È possibile aggiungere ad **Alimento** i metodi che si ritiene opportuno/utile.

**Suggerimento**: Usare la struttura dati `map` semplifica l'implementazione.

UNIVERSITÀ DELLA CALABRIA

## Esercizio 3

Il gioco del *Campo Minato (o Minesweeper o Prato Fiorito)*, può essere rappresentato tramite una matrice in cui in alcune celle sono presenti "bombe". Si richiede di scrivere una funzione

```
... popola_campo(unsigned N,vector<pair<int,int>> bombe) { ... }
```

il cui scopo è quello di costruire e restituire un *Campo Minato* secondo i seguenti criteri:

1) Il "campo" da gioco ha dimensione NxN
1) Ogni coordinata ricevuta nel vector *bombe* sarà una bomba nel campo (il primo elemento della coppia è l'indice di riga, mentre il secondo è l'indice di colonna)
2) Se (i,j) contiene una bomba, allora `campo[i][j] = -1`
3) Se (i,j) non contiene una bomba, `campo[i][j]` contiene il numero di bombe ad essa adiacenti (in orizzontale, verticale o diagonale).

Si rende necessario, ai fini dell'esercizio, scegliere un tipo di ritorno appropriato (in base all'implementazione fornita) per la funzione popola_campo. Infine, quando si accede ad un indice, è necessario controllare che esso sia valido.

## Esercizio 4

È ben noto come in Italia, durante il periodo estivo, ci sia un picco dei furti in appartamento, con conseguente incremento nell'installazione di antifurto sempre più all'avanguardia. Questo rende il mestiere del ladro sempre più complesso, e la pianificazione dei furti diventa cruciale per rimanere competitivi in questo business.

Sia *G* un grafo diretto, dove ciascun nodo *x* rappresenta il possibile bersaglio di un furto con associato potenziale bottino *V(x)*. Scrivere una funzione che ricevuto in input il grafo *G*, un vettore *V* (dove *V[i]* è il bottino associato al nodo *i*) e tre interi *k, s, W* determini se è possibile eseguire almeno *k* furti tale che:

- Il primo furto avviene sempre nel nodo *s*;
- Il bottino totale (cioè, la somma dei bottini) è esattamente *W*;
- Non è possibile derubare un nodo *i* più di una volta;
- È possibile derubare il nodo *j* dopo il nodo *i* se e solo se *(i,j)* è un arco di *G*