Fondamenti di Programmazione 2
(Corso di Laurea in Informatica)
Esame del 20/11/2023

UNIVERSITÀ DELLA CALABRIA

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] {...};

  int s = 0;
  for (int i = 0; i < 6; ++i) {
    s += *(matricola + i);
  }

  // 1. La seguente istruzione è corretta? Se sì, cosa stampa?
  cout << s << endl;

  // 2. La seguente istruzione è corretta? Se sì, cosa stampa?
  cout << *(matricola + ((matricola + 3) - (matricola + 1))) << endl;

  int& a = matricola[0];
  int& b = matricola[1];
  matricola[0] = matricola[2];
  b = matricola[3];

  // 3. La seguente istruzione è corretta? Se sì, cosa stampa?
  cout << a << matricola[1] << endl;

  // 4. Come andrebbe deallocata la memoria dinamica allocata inizialmente?
  // A: for (int i = 0; i < 6; ++i) delete matricola[i];
  // B: delete[6] matricola;
  // C: Sono necessarie entrambe le operazioni A, B.
  // D: Tutte le risposte precedenti sono errate.

  return 0;
}
```

Esercizio 2

Implementare una classe Ricetta che contenga (almeno) i campi privati nome, tempo_di_preparazione_in_minuti e ingredienti_richiesti, rispettivamente di tipo string, unsigned e vector<string>.

Successivamente, implementare una classe Ricettario, per la gestione di un insieme di ricette. La soluzione proposta dovrà implementare (almeno) i seguenti metodi:

- bool aggiungiRicetta(const Ricetta& r)
  - Aggiunge una ricetta. Non è possibile aggiungere due ricette con lo stesso nome. Il metodo restituisce true se la ricetta è stata aggiunta correttamente, altrimenti false.
- void svuotafrigo(const vector<string>& i, unsigned t)
  - Preso in input un insieme di ingredienti, la funzione dovrà stampare, con opportuna re-definizione di std::ostream::operator<< per Ricetta, tutte le ricette che si possono prepare in meno di t minuti utilizzando solamente ingredienti contenuti in i.

Nella segnatura di tutti i metodi di cui sopra, utilizzare const quando opportuno.

UNIVERSITÀ DELLA CALABRIA

Esercizio 3

Scrivere una funzione **esercizio3** che presi in input una stringa *s* e un albero binario di char restituisca true se e solo se esiste una foglia dell'albero tale che concatenando tutti i caratteri lungo il percorso radice-foglia si ottenga esattamente *s*.

La funzione dovrà avere la seguente segnatura:

```
bool esercizio3(const string& s, const AlberoB<char>& t);
```

NOTA: La classe template AlberoB<T> possiede la seguente interfaccia pubblica, dove `t` è un'istanza della classe:

- `t.radice()` - Restituisce il valore informativo della radice di `t`.
- `t.nullo()` - Restituisce true se `t` è l'albero vuoto, false altrimenti.
- `t.figlio(d)` - Restituisce il sottoalbero sinistro (se d == SIN) o destro (se d == DES).

Esercizio 4

Siano x_1, …, x_n delle variabili booleane, dove *n* è fissato e ricevuto in input. A partire da esse si possono costruire delle coppie del tipo (x_i,x_j), (not x_i, x_j), (x_i, not x_j), (not x_i, not x_j).

Una coppia si dice *soddisfatta* se è verificata almeno una di queste due condizioni:

- contiene una variabile non negata che assume il valore **true**;
- contiene una variabile negata che assume il valore **false**.

Scrivere una funzione **esercizio4** che prese in input delle coppie (modellate mediante un *vector<pair<int,int>>*) un numero intero positivo *n,* il numero di variabili utilizzate nelle coppie, e un intero positivo *k* restituisca **true** se e solo se esiste un assegnamento delle *n* variabili booleane (cioè, quale valore ciascuna delle variabili assume) tale per cui è possibile soddisfare contemporaneamente almeno *k* delle coppie in input. La funzione dovrà avere la seguente segnatura:

```
bool esercizio4(const vector<pair<int,int>>& coppie, unsigned n, unsigned k)
```

Ciascuna coppia è rappresentata da un pair<int,int>. Si può supporre che l'input sia corretto. Ad esempio, la sequenza di coppie [(1,2),(1,-2),(-1,2),(-1,-2)] va interpretata come la sequenza di coppie di variabili (x_1, x_2),(x_1, not x_2),(not x_1, x_2),(not x_1,not x_2).

Su questo input, per k = 3, la funzione dovrebbe restituire **true** in quanto, per esempio, assegnando **true** ad entrambe le variabili le prime tre coppie sono soddisfatte. Per k = 4 invece la funzione dovrebbe restituire **false** perché non è possibile che tutte le coppie siano contemporaneamente soddisfatte.
Si ricorda che, dato un *vector<pair<int,int>> V* per accere al primo (risp., al secondo) elemento di *V[i]* si può scrivere *V[i].first* (risp., *V[i].second*)