Fondamenti di Programmazione 2
(Corso di Laurea in Informatica)
Esame del 25/1/2024

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] {...};

  // 1. Le seguenti istruzioni sono corrette? Se sì, cosa stampano?
  int i = 5 - *matricola;
  if (i < 0) { i = -i; }
  cout << matricola[i] << endl;

  // 2. La seguente istruzione è corretta? Se sì, cosa stampa?
  i=5;
  int j = matricola[i];
  matricola[i] = 0;
  cout << j << endl;

  // 3. La seguente istruzione è corretta? Se sì, cosa stampa?
  i=0; j=1;
  int& k = matricola[i];
  matricola[i] = j;
  cout << k << endl;

  int* matricola_2 = matricola;

  // 4. Tra le seguenti opzioni, qual è il modo corretto di deallocare la
memoria dinamica allocata inizialmente?
  // A: delete matricola;
  // B: delete[] matricola_2;
  // C: Sono necessarie entrambe le istruzioni in (A) e (B)
  // D: Nessuna delle risposte precedenti è corretta.

  return 0;
}
```

Esercizio 2

Implementare una classe CalcolatoreFrequenze per calcolare le frequenze di occorrenza di ciascun carattere in dei testi. Per semplicità, si può assumere che tutti i testi contengano esclusivamente caratteri minuscoli dell'alfabeto latino.
La classe dovrà implementare (almeno) i seguenti metodi:

- void leggi_testo(const string& s) - Esecuzioni successive di leggi_testo accodano il nuovo testo agli eventuali testi letti precedentemente.
- vector<char> top(int k) const - Restituisce un vettore contenente i k caratteri più frequenti nel testo letto (mediante leggi_testo) fino a questo momento
- double frequenza(char c) const - Restituisce il numero di occorrenze del carattere c nel testo letto (mediante leggi_testo) fino a questo momento. *NB: Per questo esercizio, il cast implicito da int a double non causa nessun problema.*

Utilizzando opportunamente l'ereditarietà e il polimorfismo, implementare una classe CalcolatoreFrequenzeRelative, il cui metodo frequenza dovrà restituire la *frequenza relativa* del carattere c, cioè il numero totale delle sue occorrenze diviso la lunghezza totale del testo letto fino a quel momento mediante leggi_testo.

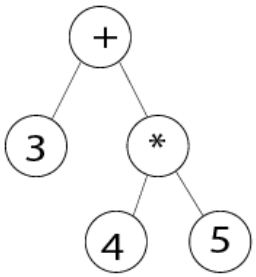
UNIVERSITÀ DELLA CALABRIA

Esercizio 3

Ogni espressione aritmetica può essere codificata da (almeno) un albero binario dove le foglie hanno come valore informativo numero intero e i nodi interni un operatore aritmetico tra "+" (addizione), "-" (sottrazione), "/" (divisione intera), e "*" (moltiplicazione).

Scrivere una funzione che presa in input un'espressione aritmetica, modellata da un'istanza di AlberoB<std::string>, restituisca una sua rappresentazione in forma di stringa inserendo correttamente anche le opportune parentesi (vedi esempio).

Si può assumere l'input rappresenti espressioni aritmetiche sintatticamente corrette.

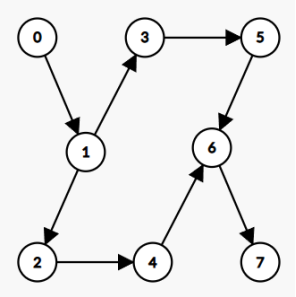
L'albero in figura codifica l'espressione aritmetica 3+4*5. La funzione dovrà restituire la stringa (3+(4*5)). NB: Nell'albero in input, i valori informativi dei nodi sono rispettivamente le stringhe "+", "3", "*", "4", "5".	

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.foglia() - Restituisce true se t è una foglia
- 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

Scrivere una funzione che presi in input un grafo orientato $G(V, E)$, una sequenza di *coppie proibite* $S = (x_1, y_1), \ldots, (x_n, y_n)$ con $x_i, y_i \in V$ per ogni $1 \leq i \leq n$ e una coppia di nodi *(s,t)* restituisca *true* se e solo se esiste un cammino da *s* a *t* tale che, per ogni coppia *(x,y)* in *S*, al più uno dei nodi tra *x* e *y* sia incluso nel cammino tra *s* e *t*.
In particolare, modellare il grafo *G* mediante un'istanza della classe *Grafo* vista a lezione, e *S* mediante un *vector<pair<int,int>>*.



Esempio Sia *G(V,E)* il grafo in figura, con *s=0* e *t=7*. Per *S_1 = (2,3), (4,5)* la funzione dovrebbe restituire *true*, in quanto un possibile cammino valido è: *0-1-2-4-6-7*, dove compare il vertice 2 (ma non 3) e il vertice 4 (ma non 5). Consideriamo il caso *S_2 = (1, 6).* L'unico nodo adiacente a 0 (il nodo di partenza del cammino) è 1, mentre l'unico nodo adiacente a 7 è 6 (il nodo di arrivo del cammino). Pertanto, ogni cammino da 0 a 7 dovrà necessariamente includere i nodi 1 e 6. In questo caso, la funzione dovrebbe restituire *false.*