Fondamenti di Programmazione - parte A: Esercitazione di laboratorio 1

Esercizio 1:

Scrivere un programma che crei due vettori di numeri interi a e b di lunghezza N=5. Il vettore b va inizializzato generando nel codice la sequenza di numeri 0, -3, 6, -9, 12. Il vettore a va creato con dei numeri a scelta. Il programma deve calcolare la somma incrociata degli elementi: a[0]+b[N-1], a[1]+b[N-2], …, a[N-1]+b[0] e memorizzarla nel vettore c. Il programma deve inoltre creare un vettore d della stessa lunghezza con valore 1 se a[i] > b[i], 0 se a[i]=b[i] e -1 altrimenti. Si visualizzino i contenuti di a, b, c, d.

Esercizio 2:

Due colleghi intendono fissare una riunione, pertanto devono identificare dei giorni nei quali sono entrambi liberi da impegni. A tale scopo, si realizzi un programma che permetta a ciascuno di immettere i propri giorni di disponibilità da tastiera, e che identifichi tutti i giorni nei quali entrambi sono liberi. Il programma deve chiedere i giorni di disponibilità ad entrambi i colleghi in successione (ciascuna persona può inserire un numero arbitrario di giorni di disponibilità, utilizzare il valore 0 per indicare la fine della sequenza dei giorni in cui ciascuna persona è libera da impegni).
Si consideri che la riunione sia nel mese corrente, quindi non interessa acquisire mese e anno, ma solo i giorni. Si memorizzi la disponibilità di ciascuna persona in un array di interi positivi di 31 elementi in cui il valore 1 rappresenta un giorno disponibile e un valore 0 rappresenta un giorno impegnato.
E' necessario verificare che i giorni siano numeri interi compresi tra 1 e 31.
Il programma deve alla fine visualizzare tutti i giorni in cui entrambe le persone sono libere da impegni.

Esercizio 3:

E' noto che esiste una definizione matematica della radice quadrata che si basa sulla seguente sequenza numerica: $x_1 = 1,\ x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right)$ dove a è un numero reale positivo.

Si può dimostrare che: $\lim_{i\to\infty} x_i = \sqrt{a}$

Si scriva un programma in cui si calcola la radice quadrata di tre numeri memorizzati in un array.
Il ciclo di calcolo della radice può fermarsi quando la differenza in valore assoluto tra il valore effettivo della radice (sqrt(a)) e x_{n+1} diventa inferiore ad un valore di soglia (es: 0.001). Utilizzare la funzione abs() per calcolare la differenza in valore assoluto.

Esercizio 4:

Definire una struct "tipolibro" con gli attributi: titolo (string), autore (string), anno di edizione (int) e prezzo (int), dove "titolo" e "autore" sono stringhe senza spazi. Scrivere un programma che legga da tastiera i dati di N (costante definita nel programma) libri memorizzandoli in un array di strutture tipolibro. Il prezzo di ciascun libro deve essere generato in modo casuale tra 0 e 200 all'interno del programma (per generare un numero casuale intero compreso tra 0 e n-1 si utilizzi l'espressione rand()%n). Il programma deve quindi calcolare il prezzo medio dei libri inseriti, determinare il libro con prezzo maggiore e il libro con l'anno di edizione più vecchio.

Esercizio 5:

Scrivere un programma che legga da tastiera il raggio (variabile float) di un cerchio ne stampi l'area utilizzando una funzione che restituisca il valore dell'area attraverso un passaggio per riferimento.

Esercizio 6:

Scrivere un programma in cui dato il prezzo netto di un prodotto (variabile float inserita in ingresso da tastiera) lo sconti del 35% se tale prezzo è > 100 e poi aggiunga l'IVA del 22% stampando a video il risultato finale.

Fondamenti di Programmazione - parte A: Esercitazione di laboratorio 2

Esercizio 1:

Scrivere un programma che definisca una classe Libro. La classe deve contenere:

- 2 dati membro privati: titolo (stringa), pagine (numero int)
- Un costruttore che consenta di inizializzare i dati membro, e che contenga anche dei valori di inizializzazione di default (stringa vuota per titolo, 0 per numero di pagine)
- Metodi "set" e "get" per leggere e modificare ciascun dato membro

Scrivere nel "main" un codice di test che crei alcuni oggetti di tipo Libro, modifichi alcuni dati membro e stampi i dati membro di ciascun oggetto.

Esercizio 2:

Scrivere un programma che definisca una classe Persona. La classe deve contenere:

- 3 dati membro privati: nome (stringa), cognome (stringa), anni (numero intero)
- Un costruttore che consenta di inizializzare i dati membro
- Un metodo pubblico "`string getInformation();`" per stampare le informazioni di una persona nel formato: <prima lettera del nome>. <cognome>, <anni>
  (esempio: "M. Rossi, 37"). Per estrarre il primo carattere di una variabile string s, in formato string, è possibile utilizzare il metodo s.substr(0, 1)
- Un metodo pubblico "`void setcognome(string c)`" per modificare il cognome di una persona con un valore passato come argomento

Creare nel "main" tre oggetti Persona con valori a scelta e stampare le informazioni di ciascuna persona utilizzando il metodo "`getInformation()`". Creare un array di 3 persone e salvare nell'array i tre oggetti creati precedentemente. Scrivere un algoritmo (utilizzando un ciclo "for") per modificare il cognome di tutte le persone contenute nell'array con il valore "Bianchi" e stampare le informazioni di ciascuna persona dopo la modifica del cognome.

Esercizio 3:

Scrivere un programma che definisca due classi: Money e CreditCard. La classe Money deve contenere due dati membri privati interi (euros e cents). La classe Money deve anche contenere:

- Costruttore che inizializza a 0 entrambi i dati membri.
- metodi pubblici per impostare e leggere il valori dei dati membri: get_euros(), set_euros(int e), get_cents(), set_cents(int c)
- Un operatore '+' definito come funzione non-membro: `Money operator+(Money m1, Money m2)` che esegue la somma di due oggetti Money sommando euro e centesimi (i centesimi se eccedono il valore di 100 vanno convertiti in euro: es 10.50 euro+ 5.70 euro= 16.20 euro)

- Un operatore '<<' definito come funzione non-membro: `ostream& operator<<(ostream& os, Money m)` per stampare a video i dati membri euros e cents

La classe CreditCard contiene tre dati membri privati: il nome del proprietario (string), il numero della carta di credito (string), il totale dei pagamenti effettuati (oggetto della classe Money). La classe CreditCard contiene anche:

- Costruttore per creare una carta dato il nome del proprietario e il numero della carta.
- Una funzione membro: print() per stampare il nome del proprietario e il numero
- Una funzione membro: `Money get_totalcharges()` che restituisce il totale dei pagamenti
- Una funzione membro: `charge(string item, int e, int c)` che aggiorna il totale dei pagamenti a seguito di una singola spesa (con causale "item") di euro 'e' e di centesimi 'c'

Il programma principale crea un oggetto di tipo CreditCard, legge da un file di testo un elenco di spese e aggiorna il totale dei pagamenti. Il file di testo delle spese contiene per ogni riga le informazioni di una singola spesa su tre colonne: <causale della spesa> <euro> <centesimi>
es:
book 90 60
pizza 20 50

Esercizio 4:

Modificare la classe Persona dell'esercizio 2 aggiungendo un operatore binario "!="
`bool operator != (Persona p2)`
come funzione membro, per determinare se due oggetti Persona sono diversi. Due oggetti Persona sono diversi se almeno uno dei tre dati membro è diverso (nome, cognome, anni). Creare due oggetti Persona nel "main" e creare un file "output.txt" (aperto in scrittura). Se i due oggetti Persona sono diversi scrivere sul file la stringa "DIVERSI", altrimenti scrivere nel file la stringa "UGUALI".

Fondamenti di Programmazione - parte A: Esercitazione di laboratorio 3

Esercizio 0:

Scrivere una classe “mobile” con due dati membro privati di tipo int: “altezza” e “larghezza”. La classe mobile deve inoltre definire un costruttore per inizializzare a “0” i valori dei due dati membro, e deve contenere dei metodi pubblici “set” e “get” per impostare e leggere il valore dei due dati membro. Scrivere una classe “armadio” derivata da “mobile” con ereditarietà di tipo pubblico. La classe “armadio” deve aggiungere un dato membro privato di tipo int “numero_ante”, un costruttore per inizializzare a “0” il numero_ante, e i rispettivi metodi pubblici “get” e “set” per impostare e leggere il valore di numero_ante. Scrivere un programma di test per verificare il comportamento delle classi.

Esercizio 1:

Scrivere una funzione template `template <class T> T max2(T primo, T secondo)`

per calcolare il maggiore tra due elementi.

Scrivere una funzione template `template <class T> T max3(T primo, T secondo, T terzo)`

per calcolare il maggiore tra tre elementi, che utilizzi la funzione max2. Scrivere un programma di test per verificare il funzionamento delle funzioni realizzate su elementi di tipo int e su elementi di tipo float.

Esercizio 2:

Scrivere una funzione template: `template <typename T> T most_common(T *A, int size)`

che accetti in ingresso un array di elementi di tipo generico T e la sua dimensione “size” e fornisca in uscita l’elemento dell’array ripetuto più volte. Se vi sono due o più elementi con lo stesso numero di ripetizioni massimo la funzione restituisce il primo elemento trovato.

Esercizio 3:

Scrivere una classe template `template<class T> class Pair` i cui oggetti rappresentino coppie di elementi (“first” e “second”) dello stesso tipo generico T. La classe oltre ai costruttori deve contenere i seguenti metodi pubblici:

- `void set_element(int position, T value); // imposta value in posizione 1 o 2`
- `T get_element(int position) const; // restituisce l’elemento in posizione 1 o 2`
- `void add_up(const Pair<T>& the_pair); // somma gli elementi corrispondenti di un oggetto Pair con quelli contenuti nell’argomento the_pair`

Esercizio 4:

Scrivere una classe template `template <class T> class Matrix2x2` per rappresentare array a due dimensioni generici di dimensione 2x2 (matrici). La classe oltre ai costruttori deve contenere un metodo pubblico per sommare ad un oggetto `Matrix2x2` una matrice passata come argomento

```
Matrix2x2<T> Add(Matrix2x2 x);
```

(la somma di due matrici è la matrice i cui elementi sono la somma di elementi in posizione corrispondente)

Fondamenti di Programmazione - parte A: Esercitazione di laboratorio 4

Esercizio 1:

Scrivere un programma in grado di fondere due liste singolarmente concatenate di interi L1, L2 (non ordinate) in una singola lista L3 ordinata in modo decrescente.

Suggerimento: scrivere una funzione "fondiliste" che utilizzi al suo interno la funzione `insert_ordered` vista a lezione e che accetti come parametri in ingresso i riferimenti alle liste L1, L2, L3.

Esempio: L1= {5, 1, 3}, L2= {10, 12, 7, 2}, L3= {12, 10, 7, 5, 3, 2, 1}

Esercizio 2:

Definire una lista concatenata composta da oggetti Item che contengono ciascuno due stringhe: una stringa abbreviata di due lettere (ad esempio TO, MI, PR) e un stringa che contiene la parola estesa (ad esempio "Torino", "Milano") corrispondente.
Scrivere una funzione che ricevendo come parametri un riferimento alla lista, una abbreviazione e la corrispondente parola estesa cerchi l'abbreviazione nella lista e

a) restituisca 0 se esiste un elemento che possiede la parola estesa e l'abbreviazione

b) restituisca 1 se non esiste un elemento che possiede la parola estesa e l'abbreviazione e inserisca un nuovo Item in coda alla lista che contenga la parola estesa e l'abbreviazione

c) restituisca 2 se l'abbreviazione è presente ma ad essa corrisponde una parola differente. La parola estesa dell'elemento della lista deve essere sostituita da quella passata come parametro.

Esercizio 3:

Scrivere un programma che crei un array di liste di dimensione N. Le liste contengono Item con valori interi positivi. Il programma deve calcolare l'indice dell'array contenente la lista in cui compare l'elemento di valore massimo tra gli elementi di tutte le liste.

Esercizio 4:

Scrivere un programma che crei una lista contenente oggetti Item con valori interi.
Si scriva una funzione che ricevendo in ingresso un riferimento alla lista la modifichi, memorizzando nell'ultimo nodo il prodotto del penultimo ed ultimo nodo, nel penultimo il prodotto del terzultimo e del penultimo e così via. Il primo elemento della lista non deve essere modificato.
Ad esempio, una lista contenente la sequenza di interi 4 6 2 3 9 verrà modificata dalla funzione nella list a 4 24 12 6 27.

Fondamenti di Programmazione - parte A: Esercitazione di laboratorio 5

Esercizio 1:

Scrivere una funzione template
```
template <typename E> void Reverse(LStack<E>& S)
```
per invertire una pila (stack) sfruttando la funzione di copia di una pila vista a lezione
```
LstackTransfer(LStack<E>& S1, LStack<E>& S2)
```

Esercizio 2:

Scrivere una funzione membro `void reverse()`
della classe LQueue per invertire una coda che utilizzi internamente uno stack
(aggiungere `#include "lstack.h"` nel file "lqueue.h")

Esercizio 3:

Salvare i numeri interi da 1 a N in uno stack e scrivere un algoritmo che utilizzi i dati nello stack per calcolare il fattoriale di N

Esercizio 4:

Scrivere un programma per il calcolo della sequenza dei primi N numeri della sequenza di Fibonacci (1 1 2 3 5 8 13 21 34 55 89 144, …) che utilizzi uno stack

Esercizio 5:

Riscrivere la funzione
```
int equal_queues(LQueue<Item>& Q1, LQueue<Item>& Q2)
```
facendola diventare una funzione template
```
template <typename E>
int equal_queues(LQueue<E>& Q1, LQueue<E>& Q2)
```
aggiungendo l'operatore != alla classe Item
```
bool operator!=(const Item &other) const
```

Esercizio 6:

Scrivere un programma che chieda all'utente un numero intero N di valori di tipo "float" da inserire. A questo punto il programma deve chiedere all'utente di inserire gli N valori. Il programma deve memorizzare i valori in una coda. Successivamente il programma deve iterativamente dimezzare la dimensione del numero di elementi della coda (arrotondando per eccesso) producendo una coda che sostituisca i valori con, alternativamente, la somma, sottrazione, moltiplicazione e divisione di coppie di elementi consecutivi. L'iterazione si arresta quando rimane un solo elemento nella coda. Esempio:
l'utente inserisce N=9 e quindi inserisce i seguenti valori interi: 4 8 -2 17 -3 2 3 21 -9
Dopo la prima iterazione la coda deve contenere i seguenti 5 (9/2 arrotondato per eccesso) valori:
12 (4+8, inizio con la somma) 15 (-2+17) -1 (-3+2) 24 (3+21) -9
Dopo la seconda iterazione la coda deve contenere i seguenti 3 (5/2 arrotondato per eccesso) valori:
-3 (12-15, questa volta tocca alla sottrazione) -25 (-1-24) -9
Dopo la terza iterazione la coda deve contenere i 2 valori (3/2 arrotondato per eccesso) seguenti:
75 (-3 * (-25), tocca alla moltiplicazione) -9
Dopo la quarta e ultima iterazione rimane un solo valore che è 75 / (-9) = -8.3333

Fondamenti di Programmazione - parte A: Esercitazione di laboratorio 6

Esercizio 1: Scrivere un programma che inizializzi due liste non ordinate L1 ed L2 di N numeri interi casuali compresi tra MINVAL e MAXVAL. Il programma deve inserire tutti gli elementi di L2 in L1 e, successivamente, ordinare in modo crescente la lista L1 e stamparla a video. Utilizzare uno qualsiasi degli algoritmi di ordinamento per liste.

Esercizio 2:Scrivere un programma per gestire una agenda telefonica contenente massimo 100 nominativi utilizzando una lista con array (classe AList). Ogni nominativo (classe Item) è composto da una struttura dati contenente: Nome (string in caratteri minuscoli), Cognome (string in caratteri minuscoli) e data di nascita (tre campi “giorno”, “mese”, “anno” in valori interi positivi). L’agenda va letta da un file di testo contenente su ogni riga <nome> <cognome> <data di nascita> di una persona. La data di nascita è espressa nella forma GG/MM/ANNO (es: 02/11/1990). L’agenda va stampata a video ordinata in modo crescente per Cognome e, a parità di Cognome, per Nome, e a parità di Cognome e Nome per data di nascita con uno qualsiasi degli algoritmi di ordinamento visti a lezione (modificando le condizioni di confronto tra Item). (Soluzione alternativa: utilizzare un operatore “<” per la classe Item)

Esercizio 3: Scrivere una funzione di ordinamento decrescente per liste utilizzando l’algoritmo “insertion sort” `void insertionsort(List<Item>& L)`

Esercizio 4: Scrivere un programma che legga da file un numero N (memorizzato nella prima riga) che rappresenta il numero di studenti di una classe. Ogni riga successiva del file contiene il cognome (string), l’età (int) e la media (float) dei voti di ciascuno studente nel formato: <cognome> <età> <media voti>
Esempio:

Rossi 24 24.56
Bianchi 23 18.2
Verdi 30 29.65
Blu 22 28.30
Rosa 19 27.3
Gialli 25 27.3
Neri 24 25.56
Viola 21 30.0

Il programma chiede all’utente due interi positivi (num_scelti ed eta_max). L’elenco degli studenti viene salvato in un array di dimensione N allocato dinamicamente. Gli elementi dell’array sono oggetti di una classe Item che contiene una struttura dati “studente” (con tre campi cognome, età e media voti). Dopo aver ordinato l’array in modo decrescente secondo la media dei voti (utilizzando un qualsiasi algoritmo) viene creata una lista concatenata che contiene i num_scelti studenti con la media più alta che hanno un’età strettamente inferiore a eta_max e la stampa a video. Nell’esempio, con num_scelti uguale=4 ed eta_max=25 la lista deve contenere gli studenti: Viola 30, Blu 28.3, Rosa 27.3 e Neri 25.56.

Esercizio 5: Scrivere una funzione ricorsiva per invertire una stringa.

Fondamenti di Programmazione - parte B: Esercitazione di laboratorio 7

Esercizio 1: Scrivere una classe Item da utilizzare come elemento di un `BinaryTree` che contenga un campo “nome” (stringa) e l’operatore “<”. Creare un `BinaryTree` nel programma principale inserendo alcuni elementi e verificare il corretto attraversamento dell’albero secondo una delle modalità viste a lezione.

Esercizio 2: Scrivere una funzione membro “equal_tree_structure” della classe BinaryTree per determinare se un albero ha la stessa struttura di un albero il cui puntatore al nodo radice viene passato come argomento (stesso numero nodi, stesso numero di livelli, stesso numero di figli in ogni sotto-albero – ma non necessariamente gli stessi valori nei nodi). E’ consentito rendere “root” membro pubblico della classe BinaryTree per avere un accesso diretto.

```
int equal_tree_structure(Node<T>* theRootB)
```

Esercizio 3: Scrivere una funzione membro “`equal_tree`” della classe BinaryTree per determinare se un albero è uguale ad un albero il cui puntatore viene passato come argomento (stesso numero nodi, stesso numero di livelli, stesso numero di figli in ogni sotto-albero, stessi valori nei nodi).

```
int equal_tree(BinaryTree<T>* treeB)
```

Esercizio 4: Scrivere il distruttore della classe `BinaryTree`

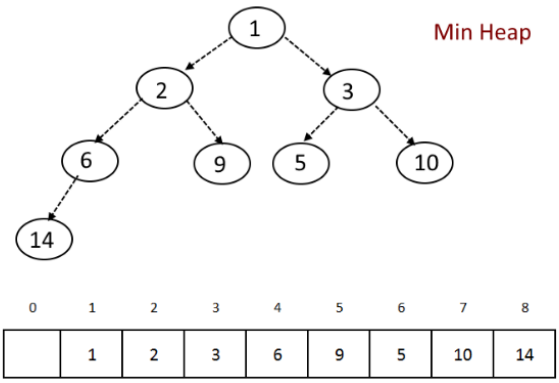
Fondamenti di Programmazione - parte B: Esercitazione di laboratorio 8

Esercizio 1:

Scrivere una classe MinHeapPQ per rappresentare una struttura dati min-heap. Un min-heap è una struttura dati che soddisfa le seguenti condizioni:

1. è un albero binario completo realizzato con un array a partire dall'elemento di indice 1
2. il valore di un nodo figlio è maggiore o uguale a quello del nodo padre

Un min-heap è una struttura dati duale rispetto alla struttura dati "heap" vista a lezione (che più propriamente è chiamata max-heap). Un esempio di min-heap è rappresentato nella seguente figura:



Scrivere il codice del costruttore, distruttore e delle seguenti funzioni membro:

- `int empty()`
- `void insert(Item item)`
- `Item getmin() per estrarre l'elemento più piccolo`

Scrivere anche il codice di un programma principale di esempio per verificare il funzionamento della classe MinHeapPQ nel caso in cui contenga elementi interi (MinHeapPQ<int>). Per risolvere l'esercizio partite dalla classe PQ vista a lezione e modificatela (è necessario modificare anche le funzioni fixUp e fixDown).

Esercizio 2:

Si vogliono attaccare assieme L funi di lunghezza diversa una dopo l'altra. Il costo dell'operazione di attaccare due funi una con l'altra è dato dalla somma della lunghezza delle due funi.

Scrivere un algoritmo che utilizzi un min-heap di numeri interi per calcolare il costo minimo possibile.

Per calcolare il costo minimo possibile si seguano i seguenti passi:

1. creare un min-heap e inserire le lunghezze di tutte le funi
2. inizializzare a 0 il costo totale
3. Ripetere i seguenti passaggi finchè il min-heap non contiene un solo elemento:
    a. Estrarre dal min-heap il minimo elemento e il secondo minimo elemento

   b. Sommare al costo totale i valori dei due elementi estratti
   c. Inserire nel min-heap la somma dei due elementi estratti
4. Stampare a video il costo totale

Per esempio avendo a disposizione 4 funi le cui lunghezze sono salvate in un array

```
int len[L] = { 4, 3, 2, 6 };
```

il costo totale minimo, secondo la procedura proposta è: 5+9+15=29

Esercizio 3:

Scrivere una procedura per creare un unico array di dimensione k*n ordinato in modo crescente partendo da k array (di n elementi ciascuno) già ordinati memorizzati in una matrice.

```
Input:
k = 3, n =  4
arr[k][n] = { {1, 3, 5, 7},
              {2, 4, 6, 8},
              {0, 9, 10, 11}} ;

Output: 0 1 2 3 4 5 6 7 8 9 10 11
```

Per risolvere l'esercizio si seguano i seguenti passi:

1. Creare l'array di output di dimensione k*n
2. Creare un min-heap di numeri interi (di dimensione massima k) e inserire il primo elemento di ciascuno dei k array nel min-heap
3. Ripetere le seguenti istruzioni n*k volte:
   a. Estrarre l'elemento più piccolo dal min-heap e salvarlo nella prima posizione libera disponibile nell'array
   b. Inserire nel min-heap l'elemento successivo (se esiste) preso dallo stesso array a cui apparteneva l'elemento estratto dal min-heap
4. Stampare a video l'array di output ordinato

Suggerimento: definire un min-heap di oggetti di una classe Item che contiene tre membri:

1. Il valore (numero intero) dell'elemento
2. L'indice dell'array di provenienza dell'elemento
3. L'indice dell'elemento successivo nell'array di provenienza dell'elemento

Fondamenti di Programmazione - parte B: Esercitazione di laboratorio 9

Esercizio 1: Leggere un file “info.dat” che contiene riga per riga un numero (ID) e una stringa.

Sviluppare un programma in C++ che svolga le seguenti operazioni:

1. memorizzare le informazioni lette dal file in un albero binario di ricerca utilizzando l'ID come chiave
2. leggere da tastiera nuove coppie di ID e stringa che vengono memorizzate in un secondo albero binario di ricerca avente lo stesso tipo di nodo del primo (questa fase termina se viene inserito un ID negativo)
3. eseguire il bilanciamento dei due alberi binari di ricerca e stamparli in logica post-order
4. scrivere un metodo “merge” della classe bst per recuperare i dati inseriti nei due alberi e inserirli in una lista singolarmente concatenata ordinata per ID crescente; qualora nei due alberi siano presenti nodi con lo stesso ID questi devono dare origine ad un singolo nodo nella lista ottenuto concatenando le stringhe con lo stesso ID
5. stampi il contenuto della lista.

Esercizio 2: Il file telefoni.txt contiene riga per riga: 1 un identificativo ufficio (stringa), nome e cognome di una persona (stringa) e un numero di telefono tutti separati dal carattere “,”.

Esempio:

pal4,SAIA Francisco,905819

palA,ZENNA Michele,905751

pal1,BERTOZZI Massimo,905845

Scrivere un programma C++ che

1. legga il contenuto del file e lo memorizzi in due alberi binari di ricerca tali che:
    i. il primo albero contenga come chiave il nome-cognome della persona e come valore l’ufficio
    ii. il secondo albero contenga come chiave il numero di telefono e come valore il nome-cognome della persona
2. Stampi a video il contenuto dei due alberi in logica in-order dopo averli bilanciati
3. Stampi a video su ogni riga l’ufficio, il nome-cognome e il telefono di ogni persona (scrivere un algoritmo non efficiente basato su una visita del secondo albero utilizzando iterativamente la funzione “select”)
4. Iterativamente chieda all'utente una stringa e
    i. se la stringa è numerica cerchi il relativo numero di telefono e stampi, se la ricerca ha esito positivo, tutti i dati della persona corrispondente (altrimenti stampare un messaggio di errore)
    ii. se la stringa non è numerica cerchi il nome inserito e stampi, se la ricerca ha esito positivo, **tutti** i nomi delle persone che lavorano nello stesso ufficio della persona cercata (altrimenti stampare un messaggio di errore). Nella ricerca dei nomi tenere

conto anche delle differenze tra caratteri maiuscoli e minuscoli (es: “PAOLO FEDERICI” è un nome diverso da “paolo Federici”). E’ consentito l’uso della funzione select.

Esercizio 3:

Nei file “parteA.txt” e “parteB.txt” sono riportati i risultati di due prove di laboratorio di fondamenti di programmazione. Riga per riga i file riportano i dati di uno studente: matricola, cognome nome, voto, email separati da virgole. Ad esempio:

150755,Aldreghetti Ivano,22,ivano.aldreghetti@studenti.unipr.it

“cognome nome” e mail sono da considerarsi stringhe, matricola e voto sono numeri interi. Una votazione insufficiente viene riportata nel file con la stringa “INSUFFICIENTE” e deve essere convertita in un valore intero pari a 0.

Scrivere un programma che:

1. legga entrambi i file e inserisca tutti i dati in due distinti alberi binari di ricerca la cui chiave deve essere la matricola, li bilanci e li stampi in logica pre-order
2. effettui un attraversamento di uno dei due alberi e per ogni nodo visitato:
   i. ricerchi la matricola corrispondente nel secondo albero

   ii. calcoli il risultato complessivo dei due esami dello studente come media algebrica dei due risultati assegnando a INSUFFICIENTE il valore 0

   iii. stampi il valore medio se sufficiente (>=18) oppure INS se insufficiente (<18)

3. stampi la media dei voti sufficienti per ciascuna delle due prove

Fondamenti di Programmazione-parte B: Esercitazione di laboratorio 10

Esercizio 1: Creare una lista di numeri interi come oggetto della classe STL list e inizializzare la lista con i seguenti valori: (1,22,4,31,4,13). Stampare il contenuto della lista. Utilizzando la funzione “insert” inserire il numero “5” in posizione 1 nella lista e, successivamente, inserire il numero “15” in posizione 2. Stampare il contenuto della lista.

Esercizio 2: Creare una lista singolarmente concatenata della classe LList, chiamata “lista_di_liste”, i cui elementi siano costituiti da liste della classe STL list contenenti oggetti di una classe Item (classe con un campo “key” di tipo intero). Inserire nella “lista_di_liste” 4 liste STL con elementi Item con valori “key” a scelta. Scrivere un algoritmo per determinare il valore “key” massimo tra tutti gli Item della “lista_di_liste”.

Esercizio 3: Creare una lista della classe STL list, chiamata “lista_di_code”, i cui elementi siano costituiti da code di numeri interi della classe STL queue. Inserire nella “lista_di_code” 4 code inizializzate con i seguenti valori: coda1: (30,12,1,203,10); coda2: (7); coda3: (14,29,45,100); coda4: (2,13,8)

Scrivere un algoritmo che per ciascuna coda della “lista_di_code”, partendo dalla prima coda fino alla penultima, estragga un elemento dalla coda e lo inserisca nella coda di indice successivo della “lista_di_code”. Stampare il contenuto di tutte le code nella “lista_di_code”.

Esercizio 4: Scrivere un programma che legga il contenuto del file “lotto.txt”. Ciascuna riga del file contiene il nome di una ruota del Lotto (singola parola) e cinque numeri interi estratti in quella ruota nel formato:

<nome_ruota> <numero1> <numero2> <numero3> <numero4> <numero5>

Salvare i dati contenuti nel file in un oggetto map di STL, chiamato “lotto”,che contenga come chiave di ogni elemento il <nome_ruota> e come dato un set di numeri interi:

```
map<string, set<int> > lotto;
```

Scrivere una funzione:

```
bool Find_terno(map<string, set<int> >& lotto, string ruota, int n1, int n2, int n3)
```

che accetti come argomenti la mappa, il nome di una ruota e tre numeri interi. La funzione Find_terno deve restituire 1 se i tre numeri passati come argomento costituiscono un “terno” sul nome della ruota passato come argomento, e 0 altrimenti. Chiamare la funzione Find_terno due volte per verificare se i tre numeri (11,12,13) costituisco un terno sulla ruota di “Firenze” e se i tre numeri (46,84,3) costituiscono un terno sulla ruota di Milano.