# IP a.a. 2021/22 - Esame del 4 febbraio 2022

**Prima di cominciare lo svolgimento leggete attentamente tutto il testo.**

Questa prova è organizzata in due sezioni, in cui sono dati alcuni elementi e voi dovete progettare ex novo tutto quello che manca per arrivare a soddisfare le richieste del testo. Per ciascuna sezione nel file zip del testo trovate una cartella contenente i file da cui dovete partire. Dovete lavorare solo sui file Cognome1.cpp e Cognome2.cpp, rinominandoli rispettivamente con il vostro cognome seguito dal numero 1 o 2. Modificare gli altri file è sbagliato (ovviamente a meno di errata corrige indicata dai docenti).

In questi file dovete realizzare le funzioni richieste, esattamente con la *segnatura* con cui sono indicate: nome, tipo restituito, tipo degli argomenti nell'ordine in cui sono dati. Potete inoltre realizzare altre funzioni in tutti i casi in cui lo ritenete appropriato. Potete inserirvi tutti gli `#include` che vi servono oltre a quello relativo allo header con le funzioni da implementare.

## 1 Sottostringhe

**Analizzare le sostringhe in comune tra due stringhe.**

Vogliamo analizzare la presenza di sottostringhe in comune tra due messaggi (stringhe). Per gli scopi di questo esercizio, una sottostringa è definita come una sequenza di caratteri contenuta in una stringa e aperta da un certo carattere e chiusa da un altro carattere.

Questi caratteri sono dati come argomenti a una funzione, quindi sono personalizzabili; hanno la funzione di *delimitatori* e quindi non fanno parte della sottostringa.

Per esempio, un delimitatore sia d'apertura che di chiusura potrebbe essere le virgolette doppie; due delimitatori distinti potrebbero essere parentesi aperta e parentesi chiusa.

### Funzione `substrings` (PSEUDOCODICE DATO)

`vector<string> substrings(string s, char opening, char closing)`

Usando l'algoritmo indicato nel seguito, scrivere la funzione `substrings` che riceve come argomento in ingresso una stringa `s` e due caratteri `opening` e `closing`.

La funzione cerca tutte le sotto-stringhe che iniziano con il carattere `opening` (escluso) e terminano con il carattere `closing` (escluso), **oppure** fino alla fine della stringa `s`. Restituisce un `std::vector` di `std::string` che contiene le sottostringhe trovate.

Esempi:

| Contenuto di s | opening | closing | risultato |
|---|---|---|---|
| `"{ atleta } vince { gara }"` | `'{'` | `'}'` | `atleta` \| `gara` |
| `"#wearethechampions We won the match! #cup"` | `'#'` | `' '` | `wearethechampions` \| `cup` |

```
   vector<string> substrings(string s, char opening, char closing)
   // dichiara una var. vector di stringhe substringlist
   // dichiara una var. stringa substring
   // dichiara una var. booleana copysubstring;
   // per ogni carattere nella stringa:
   //    se il carattere coincide con opening:
   //         assegna true a copysubstring; svuota substring
   //    altrimenti se copysubstring e' true:
   //         se il carattere coincide con closing
   //               assegna false a copysubstring
   //               aggiungi substring a substringlist
   //         altrimenti aggiungi il carattere a substring
   // se all'uscita dal ciclo copysubstring e' ancora true:
   //    aggiungi substring a substringlist
   // restituisci substringlist
}
```

SPIEGAZIONE DELL'ALGORITMO: La funzione scorre tutti i caratteri; quando trova il carattere di inizio sottostringa, attiva (mette a `true`) un flag `copysubstring` che si interpreta come "da ora in avanti copia tutti i caratteri nella sottostringa corrente". Quando trova il carattere di fine sottostringa, oltre a "spegnere" il flag (`false`), aggiunge la sottostringa alla lista da restituire. Se all'uscita dal ciclo era in corso la copia dei caratteri, ovvero era iniziata una sottostringa che alla fine non era ancora terminata, tale sottostringa corrente viene comunque aggiunta alla lista.

### Funzione `match` (DA FARE)

```
int match(vector<string> l1, vector<string> l2);
```

Riceve in ingresso due vector di stringhe e restituisce il numero di stringhe in comune fra i due

### Funzione `nomatch` (DA FARE)

```
int nomatch(vector<string> l1, vector<string> l2);
```

Riceve in ingresso due vector di stringhe e restituisce il numero totale di stringhe che non sono in comune fra i due.

### Esempio

```
l1 = [ "a", "b", "c"];    l2 = [ "a", "c", "d", "e"]
```

`match(l1,l2)` = 2
`nomatch(l1,l2)` = 3

NOTA: Non tenere conto delle ripetizioni. Assumiamo che non ci siano elementi ripetuti.

## 2 Conta occorrenze nomi

**Leggere una lista di nomi di persona e per ciascuno elencare il numero di volte che compare.**

È dato un file `nomi.txt` contenente centinaia di nomi propri di persona. Su un campione di queste dimensioni molti nomi sono inevitabilmente ripetuti. Vogliamo sapere il numero di occorrenze di ciascun nome.

Vogliamo:

- Leggere il file in una lista
- Contare il numero di occorrenze di ciascun nome
- Creare una diversa lista che contiene tutti i nomi, ciascuno una sola volta, e ciascuno con il suo numero di occorrenze.

## Materiale dato

Nel file zip trovate

- un file `nomi.h` contenente le definizioni di tipo date e le intestazioni delle funzioni ⟵ NON MODIFICARE
- un file `main.cpp` contenente un main da usare per fare testing delle vostre implementazioni e la realizzazione (corpo) delle funzioni fornite da noi. ⟵ NON MODIFICARE
- un file `Cognome.cpp` contenente lo scheletro delle funzioni che dovete realizzare voi ⟵ DOVETE MODIFICARE SOLO QUESTO
- un file `nomi.txt` contenente dati di prova

Dovete rinominare il file `Cognome.cpp` con il vostro cognome. Potete inserirvi tutti gli `#include` che vi servono oltre a quello relativo a `nomi.h`. Non siete però autorizzati a usare `algorithms`; negli esami del primo anno dovete dimostrare di essere in grado di programmare voi gli algoritmi.

Nel seguito è descritto il codice, sia quello che si richiede di fare che quello dato.

### Struttura `UnNome` (FORNITA)

```
struct UnNome {
   std::string nome;
   UnNome *next;
};
```

Rappresenta un elemento della lista nomi.

### Tipo `ListaNomi` (FORNITO)

```
typedef UnNome * ListaNomi;
```

Rappresenta una lista di nomi, ovvero un puntatore a nome.

### Struttura `UnConteggioNome` (FORNITA)

```
struct UnConteggioNome {
   std::string nome;
   int frequenza;
   UnConteggioNome *next;
};
```

Rappresenta un elemento della lista di conteggi dei nomi.

### Tipo `ListaConteggiNomi` (FORNITO)

```
typedef UnConteggioNome * ListaConteggiNomi;
```

Rappresenta una lista di conteggi di nomi, ovvero un puntatore a ListaConteggiNomi.

### Funzione `unique` (DA FARE)

```
ListaConteggiNomi unique(ListaNomi l);
```

Data una lista di nomi ordinata, restituisce una nuova lista di conteggi nomi in cui ogni nome compare solo una volta, con annotato nel campo `frequenza` il numero di volte in cui tale nome appare nella lista in ingresso.

### Funzione `main` (FORNITA)

Il programma principale, dato, esegue le seguenti operazioni:

- Legge il file nella lista nomi
- Stampa la lista dopo la lettura
- Chiama la funzione unique
- Stampa la lista frequenze