IP a.a. 2021/22 - Simulazione esame lab

Prima di cominciare lo svolgimento leggete attentamente tutto il testo. Questa prova è organizzata in due sezioni, ciascuna delle quali contiene un esercizio *di riscaldamento*, ovvero per cui è dato l'algoritmo come sequenza di commenti (come nell'eserciziario di IP), e uno di cui dovete progettare l'algoritmo di soluzione.

Per ciascuna sezione nel file zip del testo trovate una cartella contenente

- un file `.h` contenente le intestazioni delle due funzioni che dovete implementare ed eventualmente di altre già implementate.
- un file `main.cpp` contenente un main da usare per fare testing delle vostre implementazioni e la realizzazione (corpo) delle funzioni fornite da noi. Questo file **NON** deve essere consegnato, quindi eventuali modifiche **NON** sono utili ai fini della consegna.

Voi dovrete creare nella stessa cartella il file `<VostroCognome><NumeroDellaSezione>.cpp` con le vostre implementazioni. Potete inserirvi tutti gli `#include` che vi servono oltre a quello relativo all'header con le funzioni da implementare.

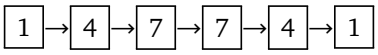
Suggerimento – Procedere ordinatamente:

- create il file `<VostroCognome><NumeroDellaSezione>.cpp` nella cartella `Sezione<NumeroDellaSezione>.cpp`.
- definite in questo file **entrambe** le funzioni richieste con un corpo minimo (ad esempio vuoto se il tipo di ritorno è `void`, o `return nullptr;` se la funzione restituisce un tipo puntatore).
- implementate una funzione alla volta, testatela e salvate il lavoro fatto **sul server per la consegna** (se incontrate difficoltà nello sviluppo, salvate anche versioni incomplete o scorrette, **purché compilanti**, per evitare di perdere il lavoro fatto fino a quel punto in caso la vostra macchina si spegnesse).

1 Lista palindroma

Per questa parte lavorate nella cartella `Sezione1`, nel file `<VostroCognome>1.cpp`.

Il tipo `PList` è una lista linkata palindroma il cui contenuto è del tipo



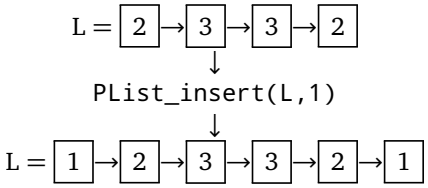
"Palindroma" = la sequenza degli elementi contenuti è la stessa scorrendo la lista dalla testa alla coda o dalla coda alla testa. L'ultimo elemento punta a `nullptr`. In aggiunta, ogni valore compare solo due (o zero) volte.

Ogni operazione sulla lista palindroma deve mantenere queste tre proprietà, ovvero: la lista deve restare palindroma, quindi inserimento e cancellazione devono operare su due elementi simmetrici; la lista deve essere sempre terminata da `nullptr`; l'inserimento avviene solo se l'elemento non è già presente. (Nota: una lista non vuota ha sempre un numero pari di elementi.)

La lista è, come al solito, realizzata come catena di elementi di un tipo `struct` che incapsula assieme un contenuto (qui un intero) e un puntatore all'elemento successivo. Il tipo di ogni elemento è il tipo struct `Elem` e l'intera lista ha tipo `PList` che equivale a `Elem *`. Queste definizioni sono date nel file `plist.h` insieme ai prototipi delle funzioni.

1.1 Riscaldamento: `PList_insert`

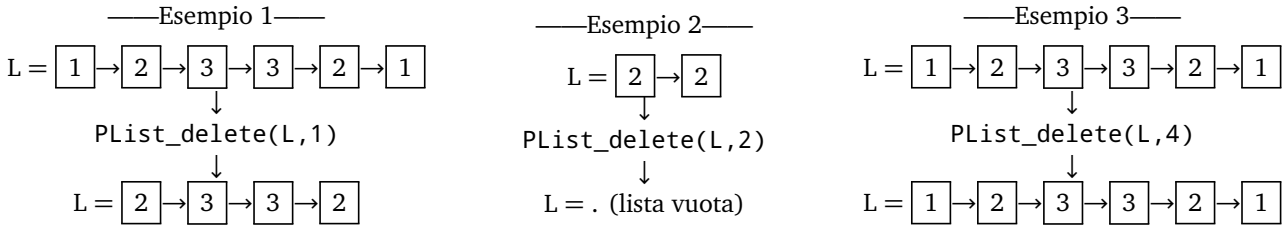
Questa funzione inserisce un elemento agli estremi della lista, ovvero in testa **e anche** in coda. Ad esempio



```
void PList_insert(PList &L, int val) {
// dichiara una variabile cur di tipo puntatore a Elem e inizializzala a L
// se L non è il puntatore nullo:
//    /* verifica che val sia presente: */
//    fintanto che il campo next di cur non è il puntatore  nullo
//        se il campo cont di cur è uguale a val
//              esci dalla funzione /* val già presente */
//        aggiorna cur assegnandogli il suo campo next
// dichiara una variabile aux di tipo puntatore a Elem e allocala
// assegna val al campo cont di aux
// assegna L al campo next di aux
// assegna aux a L
// alloca (nuovamente) aux
// assegna (nuovamente) val al campo cont di aux
// assegna il puntatore nullo al campo next di aux
// se cur è nullo assegna L a cur
// assegna aux al campo next di cur
}
```

1.2 `PListDelete`

Questa funzione rimuove un elemento dato da una lista palindroma. Sappiamo che ogni elemento, se compare, compare esattamente due volte. In tal caso la funzione deve cancellare le due occorrenze; altrimenti non fa niente.



Si suggerisce di tenere conto dei seguenti casi:

- Cancellazione di un elemento da lista vuota
- Cancellazione di un elemento che si trova all'inizio (e quindi anche alla fine)...
  - ...in lista di soli due elementi
  - ...in lista di quattro o più elementi
- Cancellazione di un elemento che si trova in posizione generica
- Cancellazione di un elemento che non è presente nella lista

2 Ordinamento ed inserimento in un std::vector

Per questa parte lavorate nella cartella `Sezione2`, nel file `<VostroCognome>2.cpp`. Per verificare il corretto funzionamento delle vostre funzioni potete utilizzare i file di esempio `AL_TEST.txt` e `AL_TEST_VUOTO.txt`, o, se preferite, produrne uno voi seguendo lo stesso schema dei txt forniti.

Il tipo di dato su cui lavorare è un `std::vector` di un tipo `struct` che memorizza le informazioni relative ad album musicali:

```
struct Album {
    std::string title;
    std::string performer;
    unsigned int year;
};
typedef std::vector<Album> AlbumLibrary;
```

2.1 Riscaldamento: ordinamento usando l'algoritmo Selection Sort

Dato un argomento di tipo `AlbumLibrary`, la funzione `void sortAlbumLibrary(AlbumLibrary& AL)` realizza l'ordinamento del contenuto del `std::vector` in ordine temporale rispetto al campo `year`. A parità di valore per tale campo, si considera l'ordine alfabetico rispetto al campo `title`, ossia al titolo dell'album. L'ordinamento viene implementato secondo il metodo *Selection Sort*. Dovete quindi implementare il seguente algoritmo:

```
void sortAlbumLibrary(AlbumLibrary& AL) {
// dichiara una variabile greatestIndex di tipo unsigned int
/* per tutti gli elementi di AL
    memorizza in greatestIndex la posizione corrente (sia i)
    /* per gli elementi di AL dalla posizione successiva a quella corrente, ossia da i+1
        se il campo year alla posizione corrente (sia j) è minore di quello alla...
        ... alla posizione greatestIndex OPPURE il valore di year è lo stesso ma...
        ... il titolo alla posizione j precede alfabeticamente quello alla...
        ... posizione greatestIndex
            memorizza j in greatestIndex
    */
    // se i è diverso da greatestIndex, scambia il contenuto alla posizione i...
    // ... con quello alla posizione greatestIndex
*/
}
```

2.2 insertAlbum

Questa funzione inserisce le informazioni relative ad un nuovo album in un `AlbumLibrary` **già ordinato**, senza verificare se sia già presente o no (duplicati ammessi). Nell'implementare la funzione, occorre quindi prestare attenzione ad inserire il nuovo elemento nella posizione giusta.