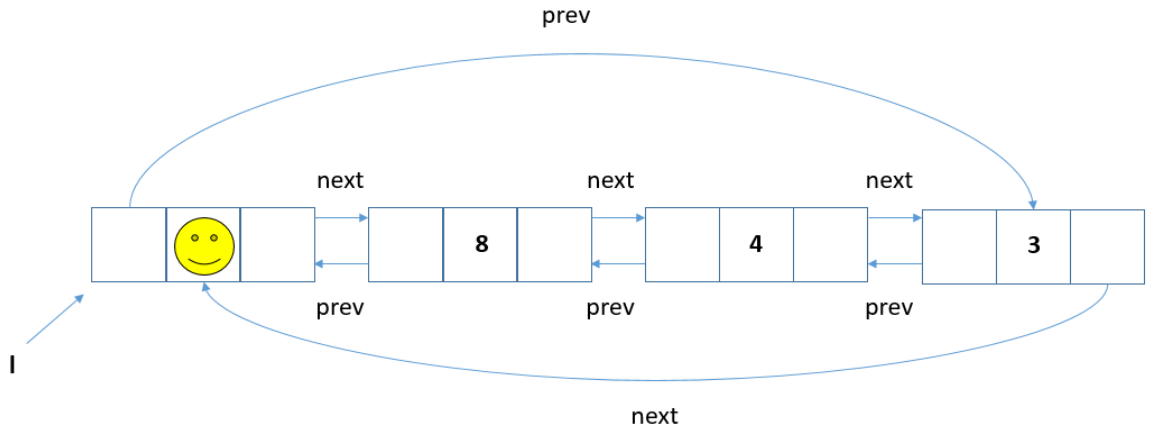
Laboratorio 1

ASD 2022/2023

In questo laboratorio viene richiesto di implementare il **tipo di dato liste** realizzate con strutture doppiamente collegate, circolari, con sentinella e non ordinate.

Di fatto si tratta di una variante di quanto avete già visto a lezione. Voi avete già visto il codice relativo alle liste semplici, semplici circolari, semplici circolari e con sentinella e alle liste doppiamente collegate (non circolari e senza sentinella).

Una rappresentazione grafica di una lista doppiamente collegata, con elementi interi immagazzinati, viene mostrata nella seguente immagine.



Come primo passo per svolgere il laboratorio dovete scaricare il file *.zip* che trovate su AulaWeb nella sezione relativa al **Laboratorio 1**. All'interno di questo file .zip (da 'unzippare', usando il comando *unzip* se state usando un sistema operativo linux[1]) troverete diversi file:

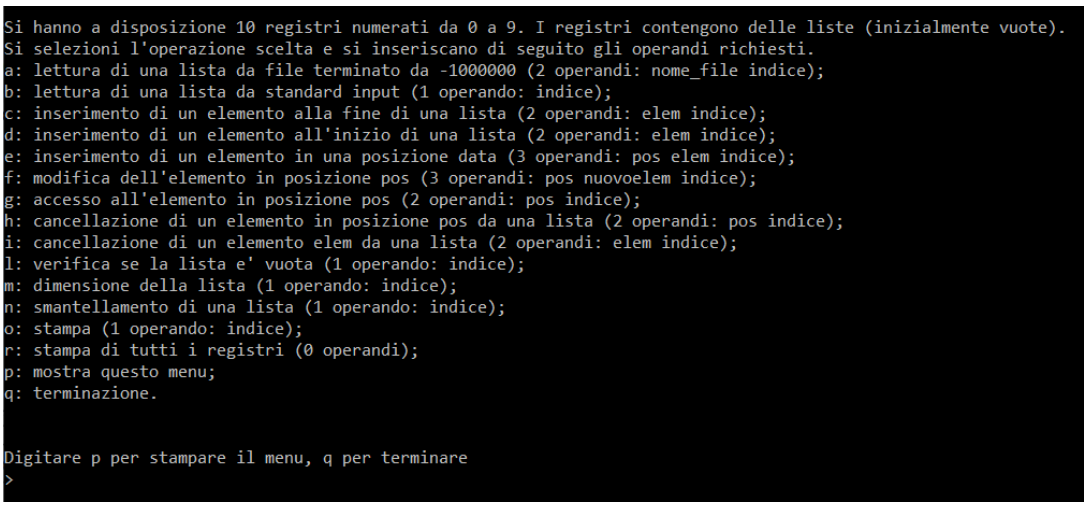
- **ASD-main.cpp**. Contiene il main del nostro programma. Questo file contiene un menù con diverse opzioni (dalla 'a' alla 'q') (vedere l'immagine mostrata sotto che rappresenta un esecuzione del codice che vi è stato fornito) che andranno a richiamare le funzioni implementate nel file **ASD-doubly-linked-list.cpp**. Come potete vedere dall'immagine il main inizializza 10 registri ognuno contenente una lista. Questo è utile perchè è possibile caricare diverse liste nei vari registri e poi compiere su queste diverse operazioni (ad esempio inserimento di un elemento, cancellazione, calcolo della dimensione, etc). Questo approccio verrà utilizzato spesso durante il corso di ASD anche con altri tipi di dato (ad esempio gli insiemi) e quindi va capito bene. All'interno del file **ASD-main.cpp** trovate le seguenti righe di codice che creano i dieci registri (precisamente come Vector di list) e li inizializzano in modo da avere all'interno di ogni registro una lista vuota:

```
const int maxreg = 10;
vector<list::List> v(maxreg); // creo i registri
for (int j=0; j<maxreg; ++j)    // inizializzo i registri in modo che tutti contengano
all'inizio
                                // la lista vuota
```

[1] https://linuxize.com/post/how-to-unzip-files-in-linux/

```
list::createEmpty(v[j]);
```

Dopo la fase inizializzazione è possibile utilizzare all'interno del file **ASD-main.cpp** la notazione v[i] per indicare la lista contenuta nel registro i-esimo del nostro Vector. Ad esempio, la linea di codice `list::addFront(5, v[2]);` una volta eseguita inserirà il numero intero 5 all'interno della lista contenuta nel registro numero 2 (attenzione che il primo registro è lo zero e non l'uno). Quindi quando nel menu raffigurato sotto viene indicato l'operando 'indice' si intende il numero del registro che contiene la lista sulla quale verrà svolta l'operazione. Come esempio, l'opzione d, richiede due operandi: l'elemento da inserire e l'indice. Con 'd 5 2' si ottiene l'inserimento in testa del numero 5 nel registro 2.



- **ASD-doubly-linked-list.h.** Contiene i prototipi delle funzioni che andranno implementate da voi nel file **ASD-doubly-linked-list.cpp** e richiamate in **ASD-main.cpp**. Questi prototipi costituiscono l'interfaccia delle nostre liste e come potete vedere, visionando il codice, sono racchiusi all'interno del namespce *list*, così come vi è stato spiegato a lezione. Per comprendere meglio il concetto di information hiding visto a lezione e realizzato tramite i namespace, sono state inserite delle linee di codice commentate nel file **ASD-main.cpp** che vi consigliamo di de-commentare per verificare l'effetto.
- **ASD-doubly-linked-list.cpp.** Contiene la traccia delle funzioni che dovrete implementare al fine di avere un codice funzionante. Questo file costituisce una traccia di codice, dove le funzioni sono state 'lasciate in bianco' (ad eccezione delle funzioni di lettura da file e di stampa che sono già disponibili). Lo stesso approccio verrà utilizzato durante le prove di laboratorio di esame.
- **Diversi file .txt**, che contengono sequenze di numeri interi e possono essere utilizzati come file di input.

E' richiesto di implementare le funzioni seguenti (solo ed esclusivamente), contenute nel file **ASD-doubly-linked-list.cpp**.). **NOTA:** Gli altri file non devono essere modificati (salvo che per scopi di testing, ma poi devono essere riportati come da originale)

```
/************************************************/
/* Implementazione delle operazioni nel namespace */
/************************************************/

/* crea la lista vuota */
void list::createEmpty(List& l){}
```

```
/* "smantella" la lista svuotandola */
void list::clear(const List& l){}

/* restituisce l'elemento in posizione pos se presente; restituisce un elemento
che per convenzione si decide che rappresenta l'elemento vuoto altrimenti*/
Elem list::get(int pos, const List& l){}

/* modifica l'elemento in posizione pos, se la posizione e' ammissibile */
void list::set(int pos, Elem e, const List& l){}

/* inserisce l'elemento in posizione pos, shiftando a destra gli altri elementi
*/
void list::add(int pos, Elem e, const List& l){}

/* inserisce l'elemento alla fine della lista */
void list::addRear(Elem e,  const List& l){}

/* inserisce l'elemento all'inizio della lista */
void list::addFront(Elem e, const List& l){}

/* cancella l'elemento in posizione pos dalla lista */
void list::removePos(int pos, const List& l){}

/* cancella tutte le occorrenze dell'elemento elem, se presente, dalla lista */
void list::removeEl(Elem e, const List& l){}

/* restituisce true se la lista e' vuota (ed e' vuota se il next di l, e' l
stessa */
bool list::isEmpty(const List& l){}

/* restituisce la dimensione, ovvero il numero di elementi, della lista */
int list::size(const List& l){}
```

E’ importante, durante la fase di implementazione, seguire tutti i consigli che vi sono stati spiegati e che sono contenuti nelle slide “*Sarebbe bello che …*”. In particolare, è importante compilare spesso il codice (meglio compilare una volta in più che una in meno …) e iniziare la codifica delle funzioni che possono essere testate, seguendo un ordine che vi permetta sempre di verificare la correttezza di quello che state implementando (ad esempio è inutile implementare la funzione size() che restituisce la dimensione di una lista se prima non avete implementato la createEmpty() e la Insert()). Durante la preparazione della traccia i docenti hanno cercato di seguire i consigli di “buona programmazione”: come vedete, il codice che vi è stato fornito è stato annotato con commenti utili (in particolare quelli che spiegano la semantica delle operazioni da implementare) ed è stato indentato[2] in modo che sia facilmente comprensibile. In linux è possibile anche utilizzare il comando *indent*[3] che effettua l’indentazione di un programma C/C++ in automatico. Il suo utilizzo è molto semplice: il comando **indent -linux sorgente.cpp** indenta il codice contenuto nel file sorgente.cpp (attenzione però che per poter applicare questo comando è necessario che il file sia corretto sintatticamente).

E’ richiesto inoltre di testare in maniera approfondita il codice prodotto cercando di esercitare tutte le funzionalità offerte dal main. Per questo vi potranno essere utili i file di input forniti. Vedremo nei laboratori successivi che alcune funzioni di test verranno inserite direttamente nel

[2] https://it.wikipedia.org/wiki/Stile_d%27indentazione
[3] https://linux.die.net/man/1/indent

menu’ per facilitare la vostra fase di implementazione (questo approccio verrrà utilizzato anche durante la prova di esame di laboratorio).

Questa “struttura” del codice (ricorrerà spesso durante il corso di ASD: sia la suddivisione in file main, file .h e file aux.cpp) così come è stata presentata in questo laboratorio (anche se i file potrebbero essere anche più di tre), sia l’utilizzo dei dieci registri utilizzati come contenitori.