

The left side of the slide features a decorative design consisting of several vertical stripes in various shades of blue and teal. Overlaid on these stripes are several circles of different sizes, also in shades of blue and teal, creating a modern, abstract background element.

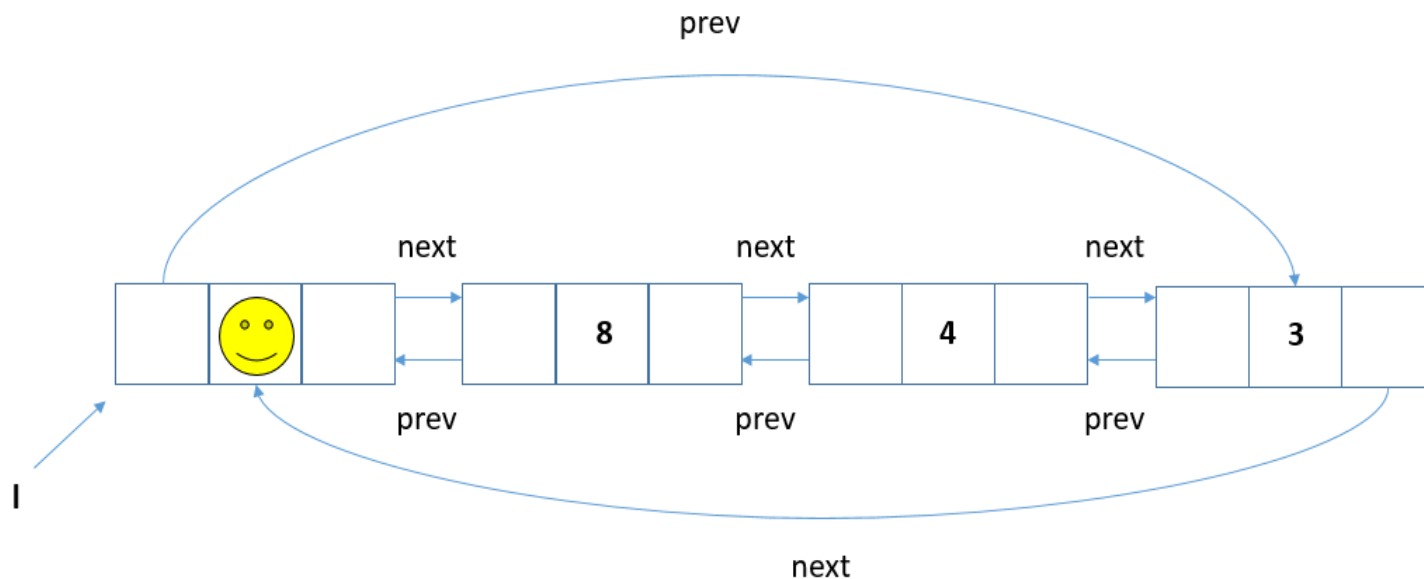
# **LABORATORIO 1**

**21-03-2023**

**Algoritmi e strutture dati 2022-2023**

# LISTE

- Liste doppiamente collegate con sentinella non ordinate



```
struct list::node {  
    Elem info;  
    node *prev;  
    node *next;  
};
```

# File .zip (TRACCIA)

Composta da **3** file + file di input

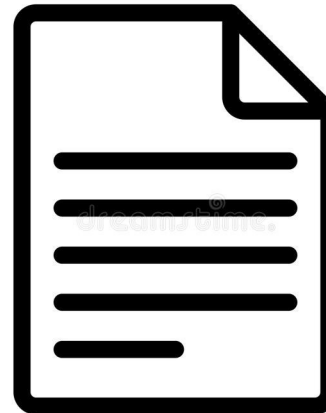
- **ASD-main.cpp**
  - il main contiene il menu che permette di eseguire le varie operazioni sulle liste - **NON MODIFICARE**
- **ASD-doubly-linked-list.h**
  - Contiene i prototipi delle funzioni che andranno implementate nel file **ASD-doubly-linked-list.cpp** e richiamate in **ASD-main.cpp** + altri dettagli implementativi utili
  - Header - **NON MODIFICARE**
- **ASD-doubly-linked-list.cpp**
  - Implementazione delle liste doppiamente collegate con sentinella
    - **Implementare le 11 funzioni richieste!**
- **File .txt**
  - Questi file contengono sequenze di numeri interi e possono essere utilizzati come file di input

ASD-main.cpp



**include**

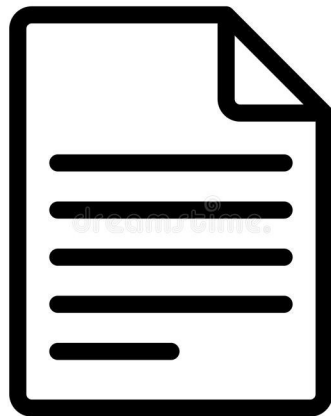
ASD-doubly-linked-list.h



**include**

**Librerie varie  
(es, <iostream>)**

ASD-doubly-linked-list.cpp



**include**

# FUNZIONI DA IMPLEMENTARE

```
/* 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) {

}
```

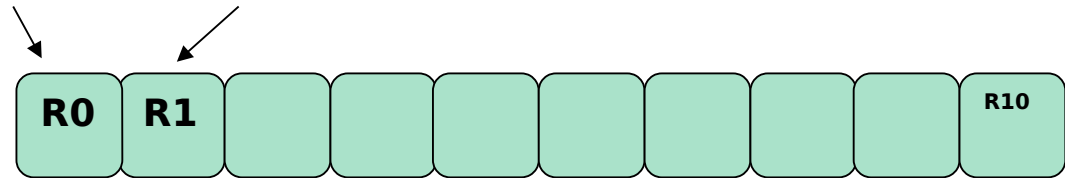
**ASD-doubly-linked-list.cpp**

# CONSIGLI 'di buona programmazione'

- **Selezionare un ordine 'buono' per implementare le funzioni**
  - Create, add, e altre operazioni
- **Compilare spesso**
  - Meglio una volta in più che una in meno ...
- **Testare bene il codice**
  - Casi particolari
  - Vedere slide dopo
- **Dare nomi significativi alle variabili**
  - x,y,pippo ... NO!
  - next, prev ... OK!
- **Aggiungere i commenti quando opportuno**
- **Indentare il codice**

# MENU

[3, 5, 8][1, 1, 2, 7]



- Il file **ASD-main.cpp** contiene un menù con diverse opzioni che andranno a richiamare le funzioni implementate nel file **ASD-doubly-linked-list.cpp**

```
Si hanno a disposizione 10 registri numerati da 0 a 9. I registri contengono delle liste (inizialmente vuote).  
Si selezioni l'operazione scelta e si inseriscano di seguito gli operandi richiesti.  
a: lettura di una lista da file terminato da -1000000 (2 operandi: nome_file indice);  
b: lettura di una lista da standard input (1 operando: indice);  
c: inserimento di un elemento alla fine di una lista (2 operandi: elem indice);  
d: inserimento di un elemento all'inizio di una lista (2 operandi: elem indice);  
e: inserimento di un elemento in una posizione data (3 operandi: pos elem indice);  
f: modifica dell'elemento in posizione pos (3 operandi: pos nuovoelem indice);  
g: accesso all'elemento in posizione pos (2 operandi: pos indice);  
h: cancellazione di un elemento in posizione pos da una lista (2 operandi: pos indice);  
i: cancellazione di un elemento elem da una lista (2 operandi: elem indice);  
l: verifica se la lista e' vuota (1 operando: indice);  
m: dimensione della lista (1 operando: indice);  
n: smantellamento di una lista (1 operando: indice);  
o: stampa (1 operando: indice);  
r: stampa di tutti i registri (0 operandi);  
p: mostra questo menu;  
q: terminazione.
```

```
Digitare p per stampare il menu, q per terminare  
>
```

# MENU (Implementazione)

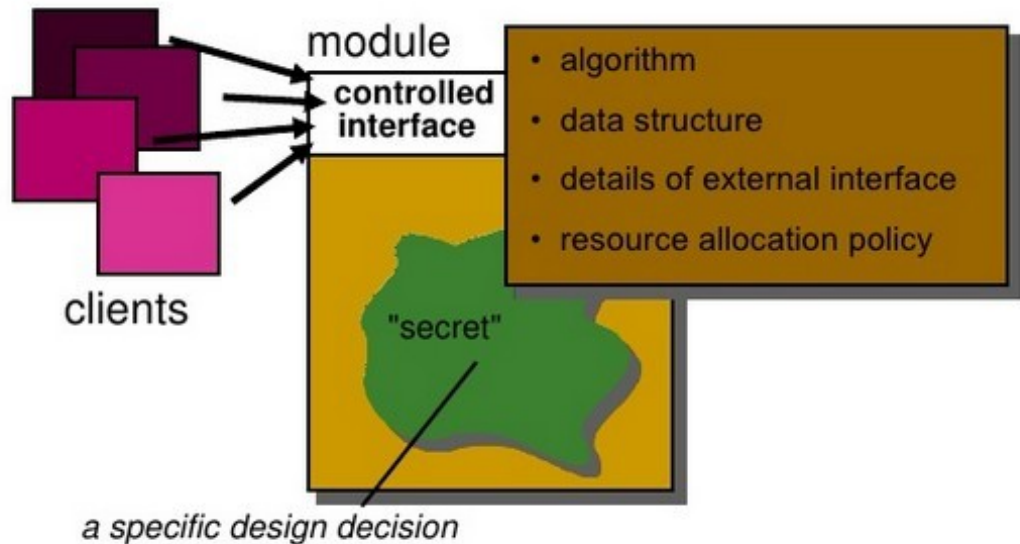
```
vector<list::List> v(maxreg);  
for (int j=0; j<maxreg; ++j)  
    list::createEmpty(v[j]);
```

- 
- 
- list::addFront(5, v[2]);**
- 
-



# INFORMATION HIDING

```
/*  
list::node s;  
s.elem=3;  
  
list::List lista;  
lista->elem = 8;  
*/
```



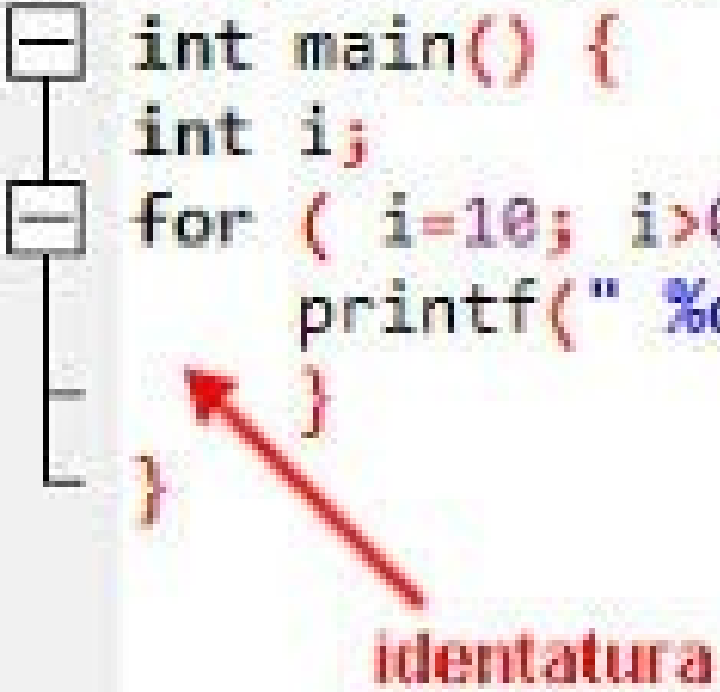
# TESTING

- Consiste nell'esecuzione del software da parte dello Sviluppatore o Tester e nel valutare se il comportamento del software rispetta le specifiche
  - Ovvero in questa fase che segue l'implementazione si cerca di capire se il software si comporta come atteso



# INDENTAZIONE

```
1  #include <stdio.h>
2  [ ] int main() {
3      | int i;
4      [ ] for ( i=10; i>0; i-- ) {
5          | printf(" %d ", i );
6          | }
7      | }
8
9
```



The diagram illustrates the indentation levels in the provided C code. Brackets on the left side of the code lines indicate the nesting of blocks: a bracket on line 2 groups lines 3 and 4, and another bracket on line 4 groups lines 5 and 6. A red arrow points from the word 'indentatura' (meaning indentation in Italian) to the space between the opening brace of the for loop and the first parameter of the printf function on line 5.

# COMPILAZIONE ED ESECUZIONE

- Compilazione: **g++ -Wall -std=c++14 \*.cpp**
- Esecuzione: **./a.out**

codice sorgente

```
...  
for (i=0;i<10;i++){  
    v[i]++;  
}  
...
```

prova.cpp



Compilatore C++

<https://gcc.gnu.org/>



codice eseguibile

```
...  
000101010101011  
101010101111010  
...
```

prova.exe

esiste anche il più recente **c++17** che può essere utilizzato ma le cui nuove feature molto probabilmente non serviranno nel corso di ASD