LABORATORIO DI PROGRAMMAZIONE
SIMULAZIONE DI TEMA DI ESAME IN `C`
CORSO DI LAUREA IN SICUREZZA INFORMATICA
UNIVERSITÀ DEGLI STUDI DI MILANO
2024–2025

Parte 1. Allocazione dinamica della memoria

La prima parte, in linguaggio `C`, servirà a valutare le vostre competenze di base di programmazione. La seconda parte, in linguaggio `C`, e la terza, in linguaggio `Java`, serviranno a valutare le competenze specifiche sui due linguaggi. Al fine di passare l'esame, dovrete risultare sufficienti in ciascuna parte, oltre che ottenere un voto combinato $\geqslant 18$. In questa lezione proveremo a risolvere la parte del tema relativa al linguaggio `C`. Vi proponiamo esercizi tratti da due temi di esame di anni precedenti.

Parte 1

Funzioni di base per gestire una lista.

Tempo: 60 min.

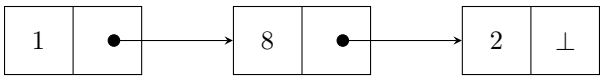

Figura 1. Esempio di una lista contenente tre valori. Le freccie indicano che un nodo contiene un riferimento ad un altro elemento della lista, mentre il simbolo $\perp$ rappresenta il valore `NULL`.

Una ***lista*** è una struttura dati *dinamica* composta da un insieme di elementi detti ***nodi*** collegati linearmente tra loro. Ogni nodo non è altro che un contenitore di dati. Nel nostro caso ogni nodo conterrà un valore numerico intero (`int`).

Avrete a disposizione il materiale caricato durante le lezioni sui siti del corso. Potrete consultarlo, e usarlo come riferimento durante l'esame. Inoltre se ne avete, potete portare dei libri di testo. Questo vuol dire che non siete sicurissimi su di un argomento, o volete essere sicuri sulla sintassi di qualche costrutto, potete fare riferimento alla documentazione. Usatela.

Ultima revisione: 21 novembre 2024.

All'interno di una lista, ogni nodo contiene un riferimento (un *puntatore*) al nodo successivo (`next`), mentre la lista terrà un riferimento al primo nodo (`header`). Una lista vuota è un puntatore a `NULL`.

In questo esercizio dovrete implementare una **lista** e le funzioni che gestiscono come vengono inseriti e rimossi dei dati da essa. Vi chiediamo di implementare una lista che supporti diverse modalità di inserimento e di rimozione degli elementi. La lista dovrà permettere all'utente di effettuare le seguenti operazioni:

- `inserisciInTesta()`: inserisce un nuovo elemento nella lista in prima posizione, in testa alla lista;
- `inserisciInCoda()`: inserisce un nuovo elemento nella lista in ultima posizione, in coda alla lista;
- `rimuoviInTesta()`: rimuove il primo elemento della lista, la testa;
- `rimuoviInCoda()`: rimuove l'ultimo elemento della testa, la coda.

Per inserire un nuovo elemento nella lista, dovrete creare un nuovo *nodo* in maniera dinamica. Un esempio di una lista contenente 3 valori è visibile in Figura 1. Alla fine della Parte 2 trovate un esempio di utilizzo della lista.

I primi esercizi serviranno per implementare i metodi e le strutture dati che utilizzerete durante la restante parte dell'esame, per la prova di C. Testate con dei casi semplici e con un main (anche senza usare le `printf` e `scanf` le funzioni ogni volta che ne finite una. Per testare una funzione, bisogna valutarla

- su di un caso normale;
- su casi limite (ad esempio, lista vuota);
- su casi di errore (ad esempio, se l'utente prova ad eliminare un elemento in una lista vuota);

Potete utilizzare le funzioni che avete già scritto e di cui avete verificato la correttezza per testare le funzioni implementate successivamente.

Esercizio 1.1
Definite una struttura `Nodo` che rappresenta un nodo della lista. La struttura contiene un campo per il *valore* dell'elemento, e un campo che si riferisce al nodo successivo. Il valore contenuto da un nodo è un numero intero.

Esercizio 1.2
Scrivete una funzione `inserisciInTesta()` che prende in input un valore intero e lo inserisce in testa alla lista. La funzione restituisce il valore dell'elemento inserito, -1 in caso di errore.

Esercizio 1.3
Scrivete una funzione `inserisciInCoda()` che prende in input un valore intero e lo inserisce in coda alla lista. La funzione restituisce il valore dell'elemento inserito, -1 in caso di errore.

Esercizio 1.4
Scrivete una funzione `rimuoviInTesta()` che elimina il primo valore della lista. Restituisce -1 se la lista è vuota.

Esercizio 1.5
Scrivete una funzione `rimuoviInCoda()` che elimina l'ultimo elemento della lista. Restituisce -1 se la lista è vuota.

Esercizio 1.6
Scrivete una funzione `cancella()` che, ricevuto una lista, ne cancelli ogni suo elemento, svuotandola.

Attenzione alla gestione della memoria allocata dinamicamente.

> Quando trovate una indicazione che dovete prestare attenzione a qualcosa, significa che dovete farlo. In questo caso, la nota vi suggerisce che dovreste ricordarvi sempre che una `malloc` abbia avuto successo, e dovrete fare una `free` ogni volta che una porzione di memoria non è più necessaria.

Il main che utilizza la struttura dati così definita è oggetto della Parte 2. Testate le singole funzioni, e considerate i possibili errori che potrebbero verificarsi durante l'esecuzione.

Parte 2

Classe ***`main`*** *e esercizi semplici.*

Tempo: 60 min.

> La seconda parte dell'esame vi chiede di implementare delle funzioni semplici su alcuni argomenti trattati durante il corso. Leggete bene il tema di esame perché spesso alcune funzioni sono utili anche a voi per debuggare il vostro codice; partite da sviluppare quelle funzioni, e poi potrete riutilizzarle. Ad esempio, in questo caso, la funzione `stampa` vi è molto utile.

Esercizio 2.1 Implementa una funzione `salva()`, che riceve come input una lista, definita come in Esercizio 1, e un nome di file. La funzione scrive i valori della lista nel file, uno per volta.

Esercizio 2.2 Implementa una funzione `carica()`, che riceve come input un nome di file e legge da esso un numero, non noto a priori, di interi. Tali valori verranno inseriti in testa in una lista definita come nell'Esercizio 1. In caso il file esista, assumete che sia strutturato in maniera corretta.

Esercizio 2.3 Implementa una funzione `stampa()` che stampa a schermo il contenuto della lista, in ordine.

Esercizio 2.4 Scrivete una funzione `rimuovi()` che elimina un elemento dalla lista, dato il suo valore. In caso l'elemento non sia presente nella lista, la funzione ritornerà -1. Nel caso vi siano più occorrenze dell'elemento nella lista, solo una verrà eliminata. Nel caso l'elemento sia presente, la funzione restituisce il valore dell'elemento eliminato.

Gestire le eccezioni.

Esercizio 2.5 Implementa una funzione ricorsiva `stampaRicorsiva` che stampi a schermo gli elementi della lista in ordine inverso. Utilizzate la **ricorsione** per ottenere questo risultato. Ad esempio, se la lista contiene 19, 12, 37, 22 a schermo dovrà essere stampato 22, 37, 12, 19.

Esercizio 2.6

> Il `main` è una collezione delle funzioni che avete sviluppato. Potete farlo mano a mano che sviluppate le singole funzioni.

Scrivere un `main()` che permetta all'utente di utilizzare una lista. Il `main()` chiederà un valore numerico all'utente e si comporterà nel seguente modo:

1 Aggiungi un elemento in testa (inserito dall'utente)
2 Rimuovi un elemento dalla testa
3 Aggiungi un elemento in coda (inserito dall'utente)
4 Rimuovi un elemento dalla coda
5 Rimuovi un elemento in lista (inserito dall'utente)
6 Carica lista da file
7 Salva lista su file
8 Stampa
9 Stampa la lista in ordine inverso
10 Cancella gli elementi di una lista.
0 Esci dal programma

Il programma continuerà a chiedere una nuova istruzione **valida** all'utente in maniera continuata, fino all'inserimento del valore di uscita.

ESEMPIO:

> Leggete tutto il testo dell'esame, perché l'esempio che viene fornito in fondo può aiutarvi a capire meglio il testo.

Lista: `NULL`
`inserisciInTesta(1)`
Lista: `1 → NULL`
`inserisciInTesta(3)`
Lista: `3 → 1 → NULL`
`inserisciInCoda(8)`
Lista: `3 → 1 → 8 → NULL`
`inserisciInCoda(2)`
Lista: `3 → 1 → 8 → 2 → NULL`
`inserisciInTesta(10)`
Lista: `10 → 3 → 1 → 8 → 2 → NULL`
`rimuoviInTesta()`
Lista: `3 → 1 → 8 → 2 → NULL`
`rimuovi(3)`
Lista: `1 → 8 → 2 → NULL`

Parte 3

Un altro esempio di esercizio.

Tempo: 60 min.

Questo è un altro esempio di un esercizio tratto da un tema di esame, in questo caso una variante di quello implementato in precedenza

Un ***array*** è una struttura dati *sequenziale* e di dimensione prefissata composta da un insieme di organizzato di oggetti sequenziali tra loro. Nel nostro caso ogni elemento dell'array conterrà una *stringa*; ogni stringa sarà di una singola parola (senza spazi); la dimensione del singolo elemento dell'array, della singola stringa, non è nota a priori.

In questo esercizio dovrete implementare un **array** di stringhe e le funzioni che gestiscono come vengono inseriti e rimossi dei dati da esso. Gli array conterranno al massimo **100** elementi (parole); tale dimensione è definita a compile time. Inizialmente saranno vuoti, e l'inserimento di un nuovo dato sarà fatto nella prima posizione libera dell'array. Dovrete implementare le funzioni per effettuare le seguenti operazioni:

- `inserisci`: inserisce un nuovo elemento nell'array nella prima posizione libera;
- `rimuovi`: rimuove l'ultimo elemento inserito nell'array;
- `unisci`: riceve due array; da questi ne crea un terzo; in questo array sarà contenuto prima l'array 1, poi l'array 2.

Per inserire un nuovo elemento nell'array, dovrete allocare una nuova *stringa* in maniera dinamica. Alla fine della Parte 2 trovate un esempio di utilizzo degli array.

Esercizio 3.1
Definite una struttura `ArrayParole`. La struttura contiene un campo con l'*array* di stringhe, di dimensione 100, e un contatore che tiene traccia di quante parole sono state effettivamente inserite nell'array. La lunghezza di ogni elemento dell'array, ovvero di ogni stringa contenente una parola, non è nota a priori; quando un nuovo elemento viene inserito, viene quindi allocata dinamicamente.

Esercizio 3.2
Scrivete una funzione `inserisci` che riceve come input una stringa e la inserisce nella prima posizione disponibile dell'array. La funzione restituisce quanti elementi sono stati inseriti nell'array in caso positivo, -1 in caso di errore.

Esercizio 3.3
Scrivete una funzione `rimuovi` che elimina l'ultimo elemento dell'array.

Esercizio 3.4
Scrivete una funzione `unisci` che, ricevute due ArrayParole, crea una terza struttura composta dalle parole della prima concatenate alle parole della seconda. Il nuovo array deve contenere delle copie delle stringhe. La grandezza del nuovo array è sempre 100; assumete che i due array da unire abbiano, complessivamente, meno di 100 elementi.

Ad esempio, se il primo array contiene le parole:
`A1[0]: "Corso"; A1[1]: "di"; A1[2]: "Informatica";`
e ila secondo array contiene le parole:
`A2[0]: "Esame"; A2[1]: "di"; A2[2]: "Programmazione";`
la struttura dati risultante risultante dalla loro unione sarà:
`A3[0]: "Corso"; A3[1]: "di"; A3[2]: "Informatica"; A3[3]: "Esame";`
`A3[4]: "di"; A3[5]: "Programmazione";`

Esercizio 3.5
Scrivete una funzione `cancella` che, ricevuto un array, ne cancelli ogni suo elemento, svuotandolo.

Attenzione alla gestione della memoria allocata dinamicamente.

Testate le singole funzioni, e considerate i possibili errori che potrebbero verificarsi durante l'esecuzione.

Anche in questo caso, la seconda parte del tema di esame vi chiedeva di implementare un `main` e le funzioni per stampare, salvare, caricare gli array. Gli esercizi sono simili a quelli della Parte 2, vi laciamo come riferimento l'esercizio sulla ricorsione.

Esercizio 3.6
Implementa una funzione ricorsiva `Ribalta` che inverta gli elementi dell'array, facendo diventare il primo elemento l'ultimo, il secondo il penultimo, e viceversa. Utilizzate la **ricorsione** per ottenere questo risultato. Ad esempio, se l'array contiene `piove, che, ore, sono`, l'array ribaltato diventa `sono ore che piove`. La modifica viene fatta sull'array stesso.

ESEMPIO:
Array 1: `lunghezza:0`;
Array 2: `lunghezza:0`;
Array 3: `lunghezza:0`;
Inserisci in Array 1: `"Lunedì"`
Array 1: `[0]:Lunedì; lunghezza:1`;
Array 2: `lunghezza:0`;
Array 3: `lunghezza:0`;
Inserisci in Array 2: `"Mercoledì"`;
Array 1: `[0]:"Lunedì"; lunghezza:1`;
Array 2: `[0]:"Mercoledì"; lunghezza:1`;
Array 3: `lunghezza:0`
Inserisci in Array 1: `Martedì`;
Array 1: `[0]:"Lunedì"; [1]:"Martedì"; lunghezza:2`;
Array 2: `[0]:"Mercoledì"; lunghezza:1`;
Array 3: `lunghezza:0`
Inserisci in Array 2: `"Giovedì"`;
Array 1: `[0]:"Lunedì"; [1]:"Martedì"; lunghezza:2`;
Array 2: `[0]:"Mercoledì" [1]:"Giovedì"; lunghezza:2`;
Array 3: `lunghezza:0`
Unisci le Array:

Array 1: `[0]:"Lunedì"; [1]:"Martedì"; lunghezza:2;`
Array 2: `[0]:"Mercoledì" [1]:"Giovedì"; lunghezza:2;`
Array 3: `[0]:"Lunedì"; [1]:"Martedì"; [2]:"Mercoledì" [3]:"Giovedì"; lunghezza:4;`
Rimuovi da Array 1;
Array 1: `[0]:"Lunedì"; lunghezza:1;`
Array 2: `[0]:"Mercoledì" [1]:"Giovedì"; lunghezza:2;`
Array 3: `[0]:"Lunedì"; [1]:"Martedì"; [2]:"Mercoledì" [3]:"Giovedì"; lunghezza:4;`

Dipartimento di Informatica, Università degli Studi di Milano,