Laboratorio II

Corso A

Esercizio 1

Scrivere un programma in **ANSI C** che gestisca una lista concatenata di studenti.
Ogni studente è rappresentato da una struttura contenente:

- matricola (intero positivo)
- età (intero positivo)

Il programma deve:

- Aggiungere studenti alla lista dinamicamente, trattata come una coda FIFO
- Stampare la lista.
- Ordinare la lista per età (ordine crescente).
- Estrarre tutti gli studenti di una certa età (letta da tastiera).
- Liberare la memoria al termine.

Soluzione (parte 1)

```c
#include <stdio.h>
#include <stdlib.h>

// ========================
// Struttura Studente
// ========================

typedef struct Studente {
    int matricola;
    int eta;
    struct Studente *next;
} Studente;


// ========================
// Funzione per aggiungere uno studente in fondo alla lista
// ========================

Studente* aggiungiStudente(Studente *head, int matricola, int eta) {
    Studente *nuovo = (Studente*)malloc(sizeof(Studente));
    if (nuovo == NULL) {
        printf("Errore: allocazione di memoria fallita.\n");
        return head;
    }

    nuovo-> matricola = matricola;
    nuovo->eta = eta;
    nuovo->next = NULL;

    if (head == NULL) {
        return nuovo;  // primo elemento
    }

    Studente *temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = nuovo;

    return head;
}
```

```c
// ========================
// Funzione per stampare la lista
// ========================
void stampaLista(Studente *head) {
    if (head == NULL) {
        printf("La lista è vuota.\n");
        return;
    }

    Studente *temp = head;
    while (temp != NULL) {
        printf("ID: %d\tEtà: %d\n", temp-> matricola, temp->eta);
        temp = temp->next;
    }
}


// ========================
// Funzione per liberare la memoria della lista
// ========================
void liberaLista(Studente *head) {
    Studente *temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }
}
```

Soluzione (parte 2)

```
// ========================
// Funzione per ordinare la lista per età (ordine crescente)
// ========================
// Implementiamo un semplice algoritmo di bubble sort sulla lista collegata

void ordinaPerEta(Studente *head) {
    if (head == NULL) return;

    int scambiato;
    Studente *ptr;
    Studente *ultimo = NULL;

    do {
        scambiato = 0;
        ptr = head;

        while (ptr->next != ultimo) {
            if (ptr->eta > ptr->next->eta) {
                // Scambio dei valori (senza modificare i puntatori)
                int temp_eta = ptr->eta;
                int temp_ matricola = ptr->matricola;

                ptr->eta = ptr->next->eta;
                ptr-> matricola = ptr->next-> matricola;

                ptr->next->eta = temp_eta;
                ptr->next-> matricola = temp_ matricola;

                scambiato = 1;
            }
            ptr = ptr->next;
        }
        ultimo = ptr;
    } while (scambiato);
}

// ========================
// Funzione per estrarre tutti gli studenti di una certa età, sfruttando l'ordine crescente
// ========================
Studente* estraiPerEta(Studente *head, int eta) {
    Studente *nuovaLista = NULL;
    Studente *temp = head;

    while (temp != NULL) {
        if (temp->eta == eta) {
            nuovaLista = aggiungiStudente(nuovaLista, temp-> matricola, temp->eta);
        }
        temp = temp->next;
    }
    return nuovaLista;
}
```

```
// ========================
// Funzione di test. La testa è allocata nel main
// ========================
void testLista(Studente** lista) {
    // Aggiungiamo alcuni studenti
    *lista = aggiungiStudente(lista, 1, 23);
    *lista = aggiungiStudente(lista, 2, 21);
    *lista = aggiungiStudente(lista, 3, 22);
    *lista = aggiungiStudente(lista, 4, 21);

    printf("\nLista originale:\n");
    stampaLista(*lista);

    // Ordinamento per età
    ordinaPerEta(*lista);
    printf("\nLista ordinata per età:\n");
    stampaLista(*lista);

    // Lettura età da tastiera
    int etaCercata;
    printf("\nInserisci l'età da estrarre: ");
    if (scanf("%d", &etaCercata) != 1) {
        printf("Input non valido.\n");
        liberaLista(*lista);
        return;
    }

    // Estrazione degli studenti di quella età
    Studente *estratti = estraiPerEta(*lista, etaCercata);

    printf("\nStudenti con età %d:\n", etaCercata);
    stampaLista(estratti);

    // Libera memoria
    liberaLista(*lista);
    liberaLista(estratti);
}


// ========================
// Main
// ========================
int main() {
    Studente *lista = NULL;
    testLista();
    return 0;
}
```

Esercizio 2

Scrivere un programma in ANSI C che memorizzi un insieme di studenti in un albero binario di ricerca (BST).
Ogni studente è rappresentato da una struttura con i seguenti campi:

- matricola (intero positivo)
- eta (intero positivo)

L'albero deve essere ordinato per età (i nodi più giovani a sinistra, i più anziani a destra).

- Il programma deve:
- Leggere da tastiera coppie di valori matricola e età e inserirli nell'albero.
  L'inserimento termina quando l'utente digita 0 come matricola.
- Stampare l'albero in ordine crescente di età.
- Estrarre e stampare tutti gli studenti di una certa età (letta da tastiera).
- Liberare la memoria allocata dinamicamente.

Le funzioni di inserimento, stampa e estrazione devono essere RICORSIVE.

Soluzione (parte 1)

```c
#include <stdio.h>
#include <stdlib.h>

// ========================
// Struttura Studente (nodo dell'albero)
// ========================
typedef struct Studente {
    int matricola;
    int eta;
    struct Studente *left;
    struct Studente *right;
} Studente;


// ========================
// Funzione per creare un nuovo nodo
// ========================
Studente* creaNodo(int matricola, int eta) {
    Studente *nuovo = (Studente*)malloc(sizeof(Studente));
    if (nuovo == NULL) {
        printf("Errore: allocazione di memoria fallita.\n");
        exit(1);
    }
    nuovo->matricola = matricola;
    nuovo->eta = eta;
    nuovo->left = NULL;
    nuovo->right = NULL;
    return nuovo;
}
```

```c
// ========================
// Inserimento in albero binario (ordinato per età)
// ========================
Studente* inserisciStudente(Studente *radice, int matricola, int eta) {
    if (radice == NULL)
        return creaNodo(matricola, eta);

    if (eta < radice->eta)
        radice->left = inserisciStudente(radice->left, matricola, eta);
    else
        radice->right = inserisciStudente(radice->right, matricola, eta);

    return radice;
}


// ========================
// Stampa in-order (età crescente)
// ========================
void stampaInOrder(Studente *radice) {
    if (radice == NULL)
        return;

    stampaInOrder(radice->left);
    printf("Matricola: %d\tEtà: %d\n", radice->matricola, radice->eta);
    stampaInOrder(radice->right);
}


// ========================
// Stampa tutti gli studenti di una certa età
// ========================
void stampaEta(Studente *radice, int eta) {
    if (radice == NULL)
        return;

    stampaEta(radice->left, eta);
    if (radice->eta == eta)
        printf("Matricola: %d\tEtà: %d\n", radice->matricola, radice->eta);
    stampaEta(radice->right, eta);
}
```

Soluzione (parte 2)

```
// ========================
// Libera la memoria dell'albero
// ========================
void liberaAlbero(Studente *radice) {
    if (radice == NULL)
        return;

    liberaAlbero(radice->left);
    liberaAlbero(radice->right);
    free(radice);
}
```

```
// ========================
// Funzione principale
// ========================
int main() {
    Studente *radice = NULL;
    int matricola, eta;

    printf("Inserisci gli studenti (matricola eta). Digita 0 per terminare.\n");

    while (1) {
        printf("Matricola: ");
        if (scanf("%d", &matricola) != 1) {
            printf("Input non valido. Uscita.\n");
            return 1;
        }
        if (matricola == 0)
            break; // fine inserimento

        printf("Età: ");
        if (scanf("%d", &eta) != 1) {
            printf("Input non valido. Uscita.\n");
            return 1;
        }

        radice = inserisciStudente(radice, matricola, eta);
    }

    printf("\nAlbero in ordine di età:\n");
    stampaInOrder(radice);

    // Estrazione per età
    int etaCercata;
    printf("\nInserisci un'età da cercare: ");
    if (scanf("%d", &etaCercata) == 1) {
        printf("\nStudenti con età %d:\n", etaCercata);
        stampaEta(radice, etaCercata);
    }

    // Libera la memoria
    liberaAlbero(radice);

    return 0;
}
```