Linked List C examples

A linked list is a set of dynamically allocated nodes, arranged in such a way that each node contains one value and one pointer. The pointer always points to the next member of the list. If the pointer is NULL, then it is the last node in the list.

A linked list is held using a local pointer variable which points to the first item of the list. If that pointer is also NULL, then the list is considered to be empty.

```
typedef struct node {
    int val;
    struct node * next;
} node_t;

node_t * head = NULL;
head = (node_t *) malloc(sizeof(node_t));
if (head == NULL) {
    return 1;
}

head->val = 1;
head->next = NULL;

// We've just created the first variable in the list. We must set the
value, and the next item to be empty, if we want to finish populating
the list. Notice that we should always check if malloc returned a NULL
value or not.

// To add a variable to the end of the list, we can just continue
advancing to the next pointer:
node_t * head = NULL;
head = (node_t *) malloc(sizeof(node_t));
head->val = 1;
head->next = (node_t *) malloc(sizeof(node_t));
head->next->val = 2;
head->next->next = NULL;

void print_list(node_t * head) {
    node_t * current = head;

    while (current != NULL) {
        printf("%d\n", current->val);
        current = current->next;
    }
}
```

Source

- Article 1
- Linked lists for absolute beginners - CodeVault

Programma di gestione delle liste con salvataggio su file

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.

```
typedef struct Node{
    int value;
    struct Node *next;
} Node;
```

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.

```
int inserisciInTesta(Node **header, int value){
    Node *element = malloc(sizeof(Node));

    if(element == NULL){
        value = -1;
    }
    else{
        element -> value = value;

        if(*header != NULL){
            element -> next = *header;
        }
        else{
            element -> next = NULL;
        }

        *header = element;
    }
    return value;
}
```

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.

```
int inserisciInCoda(Node **header, int value){
    Node *element = malloc(sizeof(Node));
    Node *curr = *header;

    if(element == NULL){
        value = -1;
    }
    else{
        element -> value = value;
        element -> next = NULL;

        if(*header == NULL){
            *header = element;
        }
        else{
            while(curr -> next != NULL){
                curr = curr -> next;
            }
            curr -> next = element;
        }
    }
    return value;
}
```

Esercizio 1.4

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

```
int rimuoviInTesta(Node **header){
    int value = 0;
    Node *curr = *header;
    if(*header == NULL){
        value = -1;
    }
    else{
        *header = curr -> next;
        free(curr);
    }
    return value;
}
```

Esercizio 1.5

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

```
int rimuoviInCoda(Node **header){
    int value = 0;
    Node *curr = *header;
    Node *prec = NULL;
    if(*header == NULL){
        value = -1;
    }
    else{
        while(curr -> next != NULL){
            prec = curr;
            curr = curr -> next;

        }
            prec -> next = NULL;
            free(curr);
    }
    return value;
}
```

Esercizio 1.6

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

```
void cancella(Node **header){
    Node *curr = *header;
    Node *next;
    while(curr != NULL){
        next = curr -> next;
        free(curr);
        curr = next;
    }
    *header = NULL;
    return;
}
```

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.

```c
void salva(Node *header, char name[100]){
    FILE *ptr = fopen(name,  "w");

    if(ptr != NULL){
        for(Node *curr = header; curr != NULL ; curr = curr -> next){
            fprintf(ptr, "%d", curr -> value);
            if(curr -> next != NULL){
                fprintf(ptr, "\n");
            }
        }
        fclose(ptr);
    }
    else{
        printf("\nErrore nella creazione del file\n");
    }
    return;
}
```

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.

```c
void carica(Node **header, char name[100]){
    FILE *ptr = fopen(name,  "r");
    Node *curr = *header;
    Node *element;
    int value = 0;
    char string[BUFSIZ] = "";

    if(ptr != NULL){
        if(curr == NULL){
            if(!feof(ptr)){
                fscanf(ptr, "%d" ,&value);
                element = malloc(sizeof(Node));
                element -> value = value;
                element -> next = NULL;
                curr = element;
            }
        }

        while(!feof(ptr)){
            fscanf(ptr, "%d" ,&value);
            element = malloc(sizeof(Node));
            element -> value = value;
            element -> next = NULL;
            curr -> next = element;
```

```
//            fgets(NULL, BUFSIZ, ptr);
        }
        fclose(ptr);
    }
    else{
        printf("\nErrore nell'apertura del file\n");
    }
}
```

Esercizio 2.3

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

```
void stampa(Node **header){
    printf("\n************************************\n");
    for(Node *curr = *header; curr != NULL; curr = curr -> next){
        printf("elemento: %d\n", curr -> value);
    }
    printf("************************************\n");
    return;
}
```

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.

```
int rimuovi(Node **header, int value){
    int returner;
    Node *curr = *header;
    Node *prec = NULL;
    if(*header == NULL){
        returner = -1;
    }
    else if(curr -> value == value){
        returner = value;
        *header = curr -> next;
        free(curr);
    }
    else{
        returner = -1;
        while((curr -> next != NULL) && (returner == -1)){
            prec = curr;
```

```
            curr = curr -> next;
            if(curr -> value == value){
                returner = value;
                prec -> next = curr -> next;
                free(curr);
            }
        }
    }
    return returner;
}
```

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 $22, 37, 12, 19$.

```
void stampaRicorsiva(Node *header){
    Node *curr = header;
    if(curr -> next != NULL){
        stampaRicorsiva(curr -> next);
    }
    printf("\nelemento: %d", curr -> value);
    return;
}
```

Esercizio 2.6

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.

11. Esci dal programma

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

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

// Node structure
typedef struct Node{
    int value;
    struct Node *next;
} Node;

int inserisciInTesta(Node **header, int value);
int inserisciInCoda(Node **header, int value);
int rimuoviInTesta(Node **header);
int rimuoviInCoda(Node **header);
int rimuoviElemento(Node **header, int value);
void printNodeList(Node **header);
void cancella(Node **header);
void stampaInversa(Node *header);
void salva(Node *header, char name[100]);
void carica(Node **header, char name[100]);

int main(void){
    Node *header = NULL;

    inserisciInTesta(&header, 19);
    inserisciInTesta(&header, 18);
    inserisciInTesta(&header, 17);
    inserisciInTesta(&header, 16);
    inserisciInCoda(&header, 20);
    printNodeList(&header);

    int choose = 11;
    char input[BUFSIZ];

    do{
        printf("\n*******Menu*******");
        printf("\n1 Aggiungi un elemento in testa");
        printf("\n2 Rimuovi un elemento dalla testa");
        printf("\n3 Aggiungi un elemento in coda");
        printf("\n4 Rimuovi un elemento dalla coda");
        printf("\n5 Rimuovi un elemento in lista");
        printf("\n6 Carica lista da file");
        printf("\n7 Salva lista su file");
        printf("\n8 Stampa");
        printf("\n9 Stampa la lista in ordine inverso");
```

```
        printf("\n10 Cancella gli elementi di una lista.");
        printf("\n0 Esci dal programma");
        printf("\n******************");
        printf("\nScelta: ");
        if (scanf("%d", &choose) == 0) {
            scanf("%s", input);
            printf("\n*******%s is not a valid option*******\n",
input);

        }
        else if (choose < 0 || choose > 10){
            printf("\n*******%d is not a valid option*******\n",
choose);
        }
        switch(choose){
            case 0:
                printf("\n***********************************");
                printf("\n********PROGRAMMA TERMINATO********");
                printf("\n***********************************");
                break;
            case 1:
                inserisciInTesta(&header, 4);
                break;
            case 2:
                rimuoviInTesta(&header);
                break;
            case 3:
                inserisciInCoda(&header, 5);
                break;
            case 4:
                rimuoviInCoda(&header);
                break;
            case 5:
                rimuoviElemento(&header, 0);
                break;
            case 6:
                carica(&header, "content.txt");
                break;
            case 7:
                salva(header, "content.txt");
                break;
            case 8:
                printNodeList(&header);
                break;
            case 9:
                printf("\n************************************");
                stampaInversa(header);
                printf("\n************************************\n");
                break;
            case 10:
```

```
                cancella(&header);
                break;
            }
        }
        while(choose != 0);
        cancella(&header);
        return 0;
}

void printNodeList(Node **header){
        printf("\n************************************\n");
        for(Node *curr = *header; curr != NULL; curr = curr -> next){
            printf("elemento: %d\n", curr -> value);
        }
        printf("************************************\n");
        return;
}

void cancella(Node **header){
        Node *curr = *header;
        Node *next;
        while(curr != NULL){
            next = curr -> next;
            free(curr);
            curr = next;
        }
        *header = NULL;
        return;
}

int inserisciInTesta(Node **header, int value){
        Node *element = malloc(sizeof(Node));

        if(element == NULL){
            value = -1;
        }
        else{
            element -> value = value;

            if(*header != NULL){
                element -> next = *header;
            }
            else{
                element -> next = NULL;
            }

            *header = element;
        }
        return value;
}
```

```c
int inserisciInCoda(Node **header, int value){
    Node *element = malloc(sizeof(Node));
    Node *curr = *header;

    if(element == NULL){
        value = -1;
    }
    else{
        element -> value = value;
        element -> next = NULL;

        if(*header == NULL){
            *header = element;
        }
        else{
            while(curr -> next != NULL){
                curr = curr -> next;
            }
            curr -> next = element;
        }
    }
    return value;
}

int rimuoviInTesta(Node **header){
    int value = 0;
    Node *curr = *header;
    if(*header == NULL){
        value = -1;
    }
    else{
        *header = curr -> next;
        free(curr);
    }
    return value;
}

int rimuoviInCoda(Node **header){
    int value = 0;
    Node *curr = *header;
    Node *prec = NULL;
    if(*header == NULL){
        value = -1;
    }
    else{
        while(curr -> next != NULL){
            prec = curr;
            curr = curr -> next;
```

```
        }
            prec -> next = NULL;
            free(curr);
    }
    return value;
}

int rimuoviElemento(Node **header, int value){
    int returner;
    Node *curr = *header;
    Node *prec = NULL;
    if(*header == NULL){
        returner = -1;
    }
    else if(curr -> value == value){
        returner = value;
        *header = curr -> next;
        free(curr);
    }
    else{
        returner = -1;
        while((curr -> next != NULL) && (returner == -1)){
            prec = curr;
            curr = curr -> next;
            if(curr -> value == value){
                returner = value;
                prec -> next = curr -> next;
                free(curr);
            }
        }
    }
    return returner;
}

void stampaInversa(Node *header){
    Node *curr = header;
    if(curr -> next != NULL){
        stampaInversa(curr -> next);
    }
    printf("\nelemento: %d", curr -> value);
    return;
}

void salva(Node *header, char name[100]){
    FILE *ptr = fopen(name,  "w");

    if(ptr != NULL){
        for(Node *curr = header; curr != NULL ; curr = curr -> next){
            fprintf(ptr, "%d", curr -> value);
            if(curr -> next != NULL){
```

```
                fprintf(ptr, "\n");
            }
        }
        fclose(ptr);
    }
    else{
        printf("\nErrore nella creazione del file\n");
    }
    return;
}

void carica(Node **header, char name[100]){
    FILE *ptr = fopen(name,  "r");
    Node *curr = *header;
    Node *element;
    int value = 0;
    char string[BUFSIZ] = "";

    if(ptr != NULL){
        if(curr == NULL){
            if(!feof(ptr)){
                fscanf(ptr, "%d" ,&value);
                element = malloc(sizeof(Node));
                element -> value = value;
                element -> next = NULL;
                curr = element;
            }
        }

        while(!feof(ptr)){
            fscanf(ptr, "%d" ,&value);
            element = malloc(sizeof(Node));
            element -> value = value;
            element -> next = NULL;
            curr -> next = element;
//            fgets(NULL, BUFSIZ, ptr);
        }
        fclose(ptr);
    }
    else{
        printf("\nErrore nell'apertura del file\n");
    }
}
```

Creare la struttura

Come scontato verranno usati gli `struct`, al cui interno ci sarà un valore di qualsiasi tipo e un puntatore ad un altro `struct`:

```
typedef struct Node
{
    int value;
    struct Node *next;
} Node;
```

Scorrere la struttura

[Iterating over a linked list in C - CodeVault]

Arrays	Linked Lists
int i = 0	Node* curr = &root
i < n	curr != NULL
i++	curr = curr -> next

While

```
Node* curr = &root;
while(curr != NULL)
{
        printf("\ncurr: %d", curr -> value);
        curr = curr -> next;
}
```

For

```
for(Node* curr = &root; curr != NULL; curr = curr -> next)
{
        printf("\ncurr: %d", curr -> value);
}
```

Aggiungere elementi alla struttura

[Adding elements to a linked list]

```
void add_tail(int **root, int value)
{
        Node *element = malloc(sizeof(Node));
        element->value = value;
        element->next = NULL;
        Node *curr = *root;
        if(curr == NULL)
        {
                curr = element;
        }
        else
```

```
        {
                while(curr->next != NULL)
                {
                        curr = curr->next;
                }
                curr->next = element;
        }
        return;
}
```

Deallocare una linked list