Studente: Alessandro Commodaro

Matricola: 274065

Studente: Daniele Commodaro

Matricola: 267250

Corso di Programmazione Procedurale e Logica
Progetto per la sessione estiva 2016/2017

Docente: Prof. Marco Bernardo

Specifica del problema.

Specifica del problema:

Una lista con doppio collegamento è una struttura dati dinamica lineare in cui ogni elemento ha un puntatore a quello successivo e un puntatore a quello precedente. Assumendo che il contenuto informativo di ciascun elemento sia un numero intero, scrivere una libreria ANSI C che, per una lista con doppio collegamento, esporta le funzioni per inserire un dato valore all'inizio o alla fine della lista oppure subito prima o subito dopo un elemento della lista che contiene un certo valore, rimuovere l'elemento che si trova all'inizio o alla fine della lista oppure che contiene un certo valore, visitare la lista all'avanti o all'indietro a partire da un certo elemento specificato attraverso la sua posizione nella lista.

Analisi del problema.

Analisi del problema:

Analisi degli input:

Gli input sono costituiti da valori numerici interi richiesti all'utente adibiti a:

- definire il contenuto dei singoli nodi della lista;
- scegliere l'algoritmo da eseguire;
- specificare la locazione sulla lista dove procedere con determinate azioni.

Analisi degli output:

Gli output sono costituiti dalla stampa a video:

- dell'intera lista corrente al termine di ogni operazione;
- delle azioni richieste all'utente per poter proseguire;
- di eventuali messaggi d'errore dovuti a input non validi;
- degli elementi percorsi in seguito ad una visita di qualunque tipo.

Relazioni intercorrenti fra input e output:

Tutti gli elementi della lista sono connessi tra loro da una relazione di concatenazione.

Progettazione dell'algoritmo.

Progettazione dell'algoritmo:

1. Principali scelte di progetto:

Con l'obiettivo di realizzare una libreria di algoritmi aventi il compito di modificare o visitare una lista in diversi modi, si è reso necessario lo sviluppo di un'interfaccia volta a far compiere la scelta dell'azione all'utente. Questo insieme di funzioni, poiché non presenti nelle specifiche del problema, viene collocato all'interno del codice sorgente main.c, codice non richiesto ma necessario per il corretto sviluppo e controllo della libreria. Sempre con questo scopo si ritiene utile articolare il programma in funzioni così che queste possano essere riutilizzate ogni qual volta ne venga richiesto l'utilizzo.

La lista è inizialmente vuota. All'aggiunta di un nuovo elemento viene creato un nuovo nodo. A tale scopo è stata inserita in libreria una funzione per la creazione dei singoli nodi componenti la lista, la quale viene utilizzata dagli algoritmi d'inserimento. Inoltre, ad ogni aggiornamento della lista, anche questi nodi subiscono variazioni, poiché, in caso di rimozione di un elemento, esso va eliminato e i collegamenti con quello precedente e successivo, se ci sono, devono essere riconfigurati tra loro. Inoltre le estremità della lista devono essere sempre riconosciute come tali, quindi, all'aggiunta di una nuova testa/coda, queste devono essere segnate, poiché necessarie come riferimento per le operazioni.

Durante l'esecuzione del programma vengono richiesti all'utente alcuni input da tastiera, sotto forma di interi. Essi hanno il compito di distinguere l'azione da compiere, stabilire il numero da inserire o da rimuovere o di riferimento, determinare una posizione nella struttura. Per prevenire il passaggio di un carattere errato è stato quindi inserito un ciclo di validazione, che si assicura il passaggio di soli numeri interi, collocato all'interno dell'apposita funzione di acquisizione. Tuttavia, per la scelta dell'operazione da compiere viene usato un altro ciclo di validazione, poiché in questo caso i valori accettabili sono solo interi da 0 a 9.

2. Acquisizione e controllo dei valori:

Innanzitutto avviene la scelta della manovra da eseguire, che può essere:

- l'inserimento di un nuovo elemento nella prima posizione della lista, ovvero la testa;
- l'inserimento di un nuovo elemento nell'ultima posizione della lista, ovvero la coda;
- l'inserimento in un punto specifico della lista, precedente ad un valore già presente;
- l'inserimento in un punto specifico della lista, successivo ad un valore già presente;
- la rimozione del primo elemento della struttura, ovvero la testa;
- la rimozione dell'ultimo elemento della struttura, ovvero la coda;
- la rimozione di un valore specifico;
- la visita a partire da un elemento scorrendo verso destra;
- la visita a partire da un elemento scorrendo verso sinistra;
- la chiusura del programma.

Ognuna di esse è associata ad un valore numerico, riportato in un menu che viene stampato a schermo ad ogni ricorrenza tramite l'invocazione della rispettiva funzione. A seconda del caso scelto, i passaggi variano come segue:

- Per inserimenti agli estremi viene richiesto all'utente il numero intero da collocare nel nuovo

nodo che verrà formato e che diverrà la nuova testa/coda, con i rispettivi collegamenti ai nodi adiacenti.

- Per inserimenti prima/dopo di un elemento già presente viene prima di tutto chiesto all'utente il valore di riferimento, ovvero il nodo accanto al quale sarà collocato il nuovo intero. A seguire viene ovviamente richiesta la cifra da inserire, cosa che avviene solo se il valore di riferimento è effettivamente presente nella lista, altrimenti si ottiene un messaggio di errore per poi tornare alla scelta dell'operazione. In caso di lista vuota il nuovo valore viene automaticamente inserito come testa. Qualora vi fossero più riferimenti uguali, viene inoltre richiesto all'utente quale occorrenza considerare. In caso di un'occorrenza non presente, come 0 o un numero superiore alla quantità effettiva o altri caratteri, viene semplicemente riposta la domanda.
- Per le rimozioni all'inizio o alla fine della lista non sono presenti ulteriori scelte da compiere. Nel caso di lista vuota non è possibile alcuna eliminazione, quindi viene rilasciato un messaggio di errore, per poi tornare al menu di scelta.
- Per rimozioni specifiche si richiede il valore che si desidera eliminare e, nel caso in cui esso sia presente più volte, la sua occorrenza. In caso di input non accettabile, ovvero per un valore non presente o non numerico, viene rilasciato un messaggio di errore per poi tornare al menu. La validazione per l'occorrenza non possibile si comporta allo stesso modo del caso precedente.
- Per le visite a partire da un elemento della lista bisogna specificarne la posizione all'interno della struttura. Un messaggio di errore viene segnalato per input non validi come caratteri, posizioni non presenti poiché al di fuori della struttura, o in caso di lista vuota, e si viene rimandati al menu. Al termine della visita vengono stampati a video i valori incontrati durante l'operazione nell'ordine del processo tramite l'apposita funzione di libreria, la stessa che riproduce la lista nel suo stato attuale al termine di ogni azione.
- In seguito alla pressione dello 0, viene mostrata un'ultima volta la struttura dati attuale e il programma si chiude.

3. Manipolazione della struttura dati:

I nodi della lista, poiché doppiamente collegata, sono tutti connessi tra loro in modo da garantire bidirezionalità. Dunque, per tutte le funzioni di modifica, è stato necessario ricreare i collegamenti e aggiornare le nuove estremità dove fosse necessario. In particolare :

- con l'inserimento di una nuova testa/coda viene creato un nuovo nodo tramite l'apposita funzione, la quale si occupa anche di collegarlo ad eventuali celle già presenti, e viene inoltre etichettato come nuova estremità;
- se l'inserimento avviene tra altri nodi, i vecchi collegamenti devono essere sostituiti con quelli che comprendono la nuova cella appena creata;
- quando avviene una rimozione i collegamenti col nodo eliminato vengono rimpiazzati con quelli tra le celle precedente e successiva, se ci sono, mentre, per le estremità, l'identificatore di testa/coda viene assegnato alla cella adiacente.

Il marcatore di fine lista è sempre il puntatore nullo.

Implementazione dell'algoritmo.

Implementazione dell'algoritmo:

Di seguito è riportato il Makefile per la compilazione dei file sorgente e della libreria secondo lo standard ANSI con relativa segnalazione di warning:

```
progetto_ppl:  main.o lista.o Makefile
	gcc -ansi -Wall -O main.o lista.o -o progetto_ppl

main.o:		main.c lista.h Makefile
	gcc -ansi -Wall -O -c main.c

lista.o: lista.c lista.h Makefile
	gcc -ansi -Wall -O -c lista.c

pulisci:
	rm -f *.o

pulisci_tutto:
	rm -f progetto_ppl *.o
```

Ecco il codice del file sorgente *main.c*:

```
/* Inclusione delle librerie standard */
#include <stdio.h>
#include <stdlib.h>

/* Inclusione della libreria per la gestione della lista */
#include "lista.h"

/* Dichiarazione delle funzioni */
void menu();
void selezione();
int acquisisci(int);

/* Funzione principale */
int main() {

	/* Selezione degli algoritmi */
	selezione();

	return 0;
}

/* Funzione per la selezione dell'operazione desiderata */
void selezione() {

	nodo_t *lista = NULL;	/* puntatore al primo elemento */
	int scelta;		/* var. per contere la scelta */
	int lettura;		/* var. validazione degli input */
	char rimosso;		/* var. per svuotare il buffer */

	/* Acquisizione della scelta con validazione */
	do {

		menu();		/* stampa le possibili scelte */
		lettura = scanf("%d", &scelta); /* acquisizione */

		/* Verifica la validità degli input */
		if((lettura != 1) || ((scelta < 0) || (scelta > 9))) {
			printf("\nScelta non presente!\n");
			do
				rimosso = getchar();
			while(rimosso != '\n');
		}

		/* Distinzione delle operazioni da eseguire */
		printf("\n");
		switch(scelta) {
			case 1:
				/* Aggiunge testa alla lista */
				in_testa(acquisisci(2), &lista);
				break;
			case 2:
				/* Aggiunge coda alla lista */
				in_coda(acquisisci(2), &lista);
				break;
			case 3:
			case 4:
				/* Inserisce un nuovo elemento
```

```
                        basandosi sulla posizione di
                        un altro già presente */
                        ins_spec(acquisisci(2),
                                acquisisci(0),
                                &lista, scelta);
                        break;
                case 5:
                        /* Rimuove testa alla lista */
                        rim_testa(&lista);
                        break;
                case 6:
                        /* Rimuove coda alla lista */
                        rim_coda(&lista);
                        break;
                case 7:
                        /* Rimuove elemento specifico */
                        rim_elem(acquisisci(2), &lista);
                        break;
                case 8:
                case 9:
                        /* Visita la lista in avanti o
                        all'indietro, in base alla scelta */
                        visita(acquisisci(1), lista,
                                scelta);
                        break;
                case 0:
                        /* Condizione di terminazione */
                        break;
        }

        /* Stampa della lista attuale */
        printf("\nLista");
        mostra(lista);
        printf(" -> NULL\n\n");

    /* Condizioni di ripetizione */
    } while (scelta != 0 || lettura != 1 ||
             scelta > 9);
}

/* Funzione per l'acquisizione dei valori */
int acquisisci(int selez) {

    int lettura;        /* var. validazione degli input */
    int pos;            /* var. per il valore acquisito */
    char rimosso;       /* var. per svuotare il buffer */

    /* Acquisizione del nuovo valore con validazione */
    do {
        /* Con selez = 1, richiede la posizione */
        if(selez == 1)
            printf("Inserire posizione\n");

        /* Con sele = 2, chiede l'elemento
        su cui agire */
        else if(selez == 2)
            printf("Inserire elemento\n");

        /* Con selez != 1, richiede il valore
```

```
                da prendere come riferimento */
                else
                        printf("Inserire valore di riferimento \n");

                lettura = scanf("%d", &pos);

                /* Verifica la validità degli input */
                if(lettura != 1 || pos < 0) {
                        printf("\nScelta non presente\n");
                        do
                                rimosso = getchar();
                        while(rimosso != '\n');
                }

        /* Condizioni di ripetizione */
        } while (lettura != 1 || pos < 0);

        /* Restituisce il valore inserito */
        return pos;
}

/* Funzione per la stampa delle opzioni */
void menu() {

        printf("\n********** MENU **********\n");
        printf("1. Inserimento testa\n");
        printf("2. Inserimento coda\n");
        printf("3. Inserimento prima elemento\n");
        printf("4. Inserimento dopo elemento\n");
        printf("5. Rimozione testa\n6. Rimozione coda\n");
        printf("7. Rimozione elemento\n");
        printf("8. Visita all'avanti\n");
        printf("9. Visita all'indietro\n");
        printf("0. Per terminare\n");

}
```

Di seguito, il codice del file sorgente *lista.c*:

```
/* Librerie standard */
#include <stdio.h>
#include <stdlib.h>

/* Definizione del nuovo tipo */
typedef struct nodo  {
        int dato;               /* valore del nodo */
        struct nodo* succ_p;/* puntatore a elem. succ. */
        struct nodo* prec_p;/* puntatore a elem. prec. */
}nodo_t;                        /* ridefinizione del nuovo tipo */

/* Prototipi delle funzioni */

nodo_t *crea_nodo(int);
void in_testa(int, nodo_t **);
void in_coda(int, nodo_t **);
void mostra(nodo_t *);
void mostra_ind(nodo_t *);
void rim_testa(nodo_t **);
```

```
void rim_coda(nodo_t **);
void rim_elem(int, nodo_t **);
void ins_spec(int, int, nodo_t **, int);
void visita(int, nodo_t *, int);
int conta(nodo_t *, int);

/* Definizione delle funzioni */

/* Funzione per creare un nuovo nodo,
la funzione riceve in input il valore da
aggiungere ed alloca lo spazio necessario
all'inserimento, i campi puntatore precedente
e successivo vengono inizializzati a NULL.
Viene restituito l'indirizzo del nuovo elemento */
nodo_t *crea_nodo(int x) {

        nodo_t *nuovo               /* puntatore che ospiterà
                                    l'indirizzo del nuovo elem. */

                /* Viene allocato lo spazio necessario alla
                creazione del nuovo elemento */
                = (nodo_t *)malloc(sizeof(nodo_t *));

        nuovo->dato = x;    /* il valore viene acquisito */

        /* I campi puntatore precedente e successivo vengono
        inizializzati a NULL */
        nuovo->prec_p = nuovo->succ_p = NULL;

        /* Viene restituito l'indirizzo del nodo creato */
        return nuovo;
}

/* Corpo della funzione inserisci testa,
la quale riceve in input il valore da aggiungere e il
riferimento alla struttura localizzata tramite il suo
primo elemento.
Viene invocata la funzione crea_nodo() per aggiungere
il nuovo elemento; quest'ultimo diviene la nuova testa,
mentre la precedente diviene l'elemento successivo ad essa. */
void in_testa(int x, nodo_t **testa) {

        /* Invocazione della funzione crea nodo */
        nodo_t *nuovo = crea_nodo(x);

        /* Caso lista inizialmente vuota, il nuovo elemento
        viene posizionato in testa */
        if(*testa == NULL) {
                *testa = nuovo;
        }

        /* Se la lista non è vuota */
        else{
                /* Si crea lo spazio per il nuovo elemento
                all'indirizzo puntato da prec_p inizialmente
                settato a NULL */
                (*testa)->prec_p = nuovo;

                /* La vecchia testa diviene il secondo elem. */
```

```
                nuovo->succ_p = *testa;

                /* Viene aggiornata la nuova testa */
                *testa = nuovo;
        }
}

/* Funzione per inserire in coda, che riceve in input
il valore da inserire e il riferimento alla struttura.
Viene richiamata la funzione crea_nodo() per aggiungere
il nuovo nodo; in caso di lista vuota questo diventa la
nuova testa; altrimenti si scorre la lista fino alla sua
fine, per poi inizializzare il puntatore succ_p (con valore
NULL) come nuovo elemento */
void in_coda(int x, nodo_t **testa) {

        /* Dichiarazione e inizializzazione del puntatore
        interno alla funzione, che assume l'indirizzo del
        primo elemento della lista.
        Questo viene utilizzato come puntatore ausiliario
        per scorrere la lista ed aggingere l'elemento in coda */
        nodo_t *temp = *testa;

        /* Invocazione della funzione crea_nodo() */
        /* La variabile *nuovo contiene
        l'indirizzo del nuovo elemento generato dalla
        funzione crea_nodo() */
        nodo_t *nuovo = crea_nodo(x);

        /* Se la lista è priva di elementi, è sufficiente
        aggiornare la testa della lista con l'indirizzo
        restituito dalla funzione crea_nodo() */
        if(*testa == NULL) {
                *testa = nuovo;
        }

        /* Se la lista non è vuota */
        else{

        /* Si scorre fino alla fine */
                while(temp->succ_p != NULL)
                        temp = temp->succ_p;

        /* Il nuovo nodo viene inserito in coda */
                temp->succ_p = nuovo;

        /* Si collega il nuovo nodo con il suo precedente */
                nuovo->prec_p = temp;
        }
}

/* Funzione per rimuovere la testa, l'unico
parametro necessario è l'indirizzo della testa */
void rim_testa(nodo_t ** testa) {

        /* Se la lista è vuota si avverte l'utente */
        if(*testa == NULL)
                printf("Lista vuota!");
```

```
	/* Se la lista non è vuota */
	else {

		/* Si definisce un puntatore ausiliario, e lo
		si inizializza con l'indirizzo della testa */
		nodo_t *temp = *testa;

		/* Caso lista composta da un solo elemento */
		if(temp->prec_p == temp->succ_p) {
			*testa = NULL;
			free(temp);
		}

		/* Caso lista con più elementi */
		else {
			/* Il secondo elemento viene aggiornato
			e diviene la nuova testa, mentre quella
			precedente viene quindi eliminata */
			*testa = temp->succ_p;
			(*testa)->prec_p = NULL;
			free(temp);
		}
	}
}

/* Definizione del corpo della funzione per rimuovere
l'elemento in coda. In caso di lista vuota la funzione
termina alla prima riga ed il resto del codice non
viene eseguito; in caso contrario si distingue il caso
di lista comprensiva di un solo elemento che viene quindi
rimosso. Nel caso con più di un elemento si scorre fino
al carattere di terminazione (NULL)  */

void rim_coda(nodo_t **testa) {

	/* Se la lista è vuota la rimozione non può
	avere luogo, l'utente viene quindi avvisato */
	if(*testa == NULL)
		printf("Lista vuota!\n");

	/* Se la lista non è vuota */
	else {

		/* Viene dichiarato un puntatore ausiliario che
		assume l'indirizzo del primo elem. della lista */
		nodo_t *temp = *testa;

		/* Caso lista composta da un solo elemento */
		if(temp->prec_p == temp->succ_p) {
			*testa = NULL;
			free(temp);
		}

		/* Caso lista composta da più di un elem. */
		else {

			/* Si scorre l'intera lista */
			while(temp->succ_p != NULL)
				temp = temp->succ_p;
```

```
                /* Si elimina il collegamento all'ultimo elem.
                che può essere quindi rimosso */
                temp->prec_p->succ_p = NULL;
                free(temp);
                }
        }
}

/* Definizione del corpo della funzione per rimuovere un
nodo a scelta dell'utente; la funzione accetta come parametri
il valore da rimuovere deciso dall'utente durante l'esecuzione
e il riferimento al primo elemento della lista.
Si distinguono le tre casistiche: lista vuota, lista con un solo
elemento e lista composta da più elementi.  */

void rim_elem(int x, nodo_t **testa) {

        /* In caso di lista vuota la rimozione non può avere
        luogo, viene quindi notificato con un messaggio */
        if(*testa == NULL)
                printf("Lista vuota!");

        /* Caso in cui la lista consta di un solo elemento */
        else if((*testa)->succ_p == NULL)

                /* si invoca la funzione rim_testa() */
                rim_testa(*&testa);

        /* Caso lista con piu di un elemento */
        else {

                /* Si definisce un puntatore ausiliario, che
                viene inizializzato con l'indirizzo della testa */
                nodo_t *temp = *testa;

                /* Variabile che contiene il valore di ritorno della
                funzione conta (funzione adibita a scandire le
                occorrenze di elementi contenenti lo stesso
                valore) */
                int scelta = conta(*testa, x);

                /* Variabile usata per controllare l'esecuzione
                della funzione */
                int rimosso = 1;

                /* Il ciclo continua fino a che non si incontra
                il valore da rimuovere o se, nel caso il valore
                non fosse presente, la variabile rimosso viene
                settata a 0. Nel contempo la variabile scelta
                deve rimanere diversa dal valore 1, la quale
                rappresenta la casistica in cui si incontrino più
                occorrenze dello stesso valore */
                while(((temp->dato != x) &&
                        (rimosso != 0)) || (scelta != 1)) {

                        /* Se scorrendo la lista incontro il valore
                        cercato ma l'occorrenza di questo non è
                        quella desiderata, si decrementa la var.
```

```
		scelta di un unità */
		if((temp->dato == x) && (scelta != 1))
			scelta--;

		/* Caso in cui scorrendo tutta la lista
		il valore non viene trovato nemmeno una
		volta e si notifica con un messaggio */
		if(temp->succ_p == NULL) {
			printf("Valore non trovato!\n");

			/* Se rimosso viene impostato a 0
			l'esecuzione della funzione
			termina */
			rimosso = 0;
		}
		else
			/* Incremento del ciclo iterativo */
			temp = temp->succ_p;
}

/* Se l'elemento da eliminare è stato trovato */
if(rimosso != 0) {

		/* L'elemento da rimuovere è situato
		in testa alla lista */
		if(temp == *testa) {

			/* Il secondo elemento della lista
			viene aggiornato a nuova testa,
			e la precedente può essere quindi
			rimossa */
			*testa = (*testa)->succ_p;

			/* Si aggiorna il marcatore di
			fine lista della nuova testa */
			(*testa)->prec_p = NULL;
			free(temp);
		}

		/* Caso in cui l'elemento da rimuovere
		è situato in coda */
		else if(temp->succ_p == NULL)

			/* Si invoca la funzione per
			rimuovere un elem. dalla coda */
			rim_coda(*&testa);

		/* Caso in cui l'elemento da rimuovere non si
		trova agli estremi della lista */
		else {
			/* Si aggiorna il puntatore a
			elemento precedente del nodo
			successivo a quello da eliminare,
			che viene fatto	puntare al nodo che
				precede temp */
			temp->succ_p->prec_p = temp->prec_p;

			/* Lo stesso tipo di aggiornamento
			del collegamento risulta
```

```
                                necessario per il nodo che precede
                                quello da eliminare */
                                temp->prec_p->succ_p = temp->succ_p;
                                free(temp);
                        }
                }
        }
}

/* Funzione per inserire un nuovo elemento subito prima o subito
dopo un valore di riferimento. La funzione acquisisce come
parametri un intero da inserire, il valore a cui fare riferimento,
il primo nodo della lista ed il valore necessario per permettere la
scelta del tipo di inserimento (prima/dopo) */
void ins_spec(int x, int val_rif, nodo_t **testa, int selez) {

        /* Var. puntatore per contenere l'indirizzo del nuovo
        elemento */
        nodo_t *elem = NULL;

        /* Variabile usata per gestire l'uscita dalla funzione */
        int esci = 0;

        /* In caso di lista vuota la rimozione non può avere
        luogo, viene quindi notificato con un messaggio */
        if(*testa == NULL) {
                printf("Lista vuota!\n");
                printf("Il valore %d viene inserito in testa\n", x);

                /* Si richiama la funzione per la creazione di
                un nuovo nodo */
                elem = crea_nodo(x);

                /* L'elemento inserito diventa la nuova testa */
                *testa = elem;
        }
        else {

                /* Puntatore ausiliario per scorrere la lista */
                nodo_t *temp1 = *testa;

                /* Secondo puntatore ausiliario per la creazione
                del nuovo nodo */
                nodo_t *temp2 = *testa;

                /* Variabile che contiene il valore di ritorno della
                funzione conta (funzione adibita a scandire le
                occorrenze di elementi contenenti lo stesso
                valore) */
                int scelta = conta(*testa, val_rif);

                /* Continua a iterare fino a quando non trova
                il valore richiesto oppure fino ad arrivare
                alla fine della lista; allo stesso tempo il ciclo
                continua fino a quando la var. scelta ha un valore
                diverso da 1.  */
                while(((temp1->dato != val_rif) &&
                        (esci == 0)) || (scelta != 1)) {
```

```
            /* Quando trova un nodo nella lista
            contenente un valore uguale a quello di
            riferimento, scelta viene decrementato
            di un unità */
            if(temp1->dato == val_rif)
                    scelta--;

            /* Se arriva fino a fine della lista allora
            il valore di riferimento non è presente */
            if(temp1->succ_p == NULL) {

                    /* Notifica con un messagio */
                    printf("Il valore %d non è presente",
                            val_rif);

                    /* La var. esci viene impostata
                    ad 1, questo consente di terminare
                    l'esecuzione della funzione */
                    esci = 1;
            }
            else

                    /* Incremento del while */
                    temp1 = temp1->succ_p;

}

/* Se la variabile selez ha valore 3, l'inserimento
avrà luogo subito prima del valore usato come
riferimento */
if(selez == 3) {

            /* Se l'elemento di riferimento si trova in
            cima alla lista */
            if((temp1->prec_p == NULL) && (esci == 0))

                    /* Viene invocata la funzione per
                    aggiungere un nuovo elemento
                    in testa */
                    in_testa(x, &*testa);

            /* Caso in cui l'elemento da rimuovere non
            sitrova agli estremi della lista */
            else if(esci == 0) {

                    /* Viene invocata la funzione
                    che genera un nuovo nodo */
                    elem = crea_nodo(x);

                    /* Il puntatore temp2 viene
                    associato al campo puntatore
                    precedente di temp1 */
                    temp2 = temp1->prec_p;

                    /* Il nuovo elemento diventa il nodo
                    precedente a temp1 */
                    temp1->prec_p = elem;

                    /* Il campo puntatore al
```

```
                    nodo successivo del nuovo elemento
                    viene associato a temp1 */
                    elem->succ_p = temp1;

                    /* Viene definito il collegamento
                    con il nodo che precede quello
                    appena creato, che viene
                    associato a temp2 */
                    elem->prec_p = temp2;

                    /* temp2 viene collegato al nodo
                    appena creato che lo succede */
                    temp2->succ_p = elem;
            }
}

/* Se la variabile selez ha valore diverso da 3,
l'inserimento del nuovo elem. avrà luogo subito
dopo il valore di riferimento */
else {

        /* Se l'elemento di riferimento si trova
        in coda */
        if((temp1->succ_p == NULL) && (esci == 0))

                /* L'elemento viene aggiunto in coda
                attraverso l'invocazione della
                funzione in_coda() */
                in_coda(x, &*testa);

        /* Se l'elemento di riferimento non si trova
        in coda */
        else if(esci == 0){

                /* Invoca la funzione per creare un
                nuovo nodo */
                elem = crea_nodo(x);

                /* Il puntatore temp2 viene
                associato al campo puntatore
                successivo di temp1 */
                temp2 = temp1->succ_p;

                /* Il nuovo elemento diventa il nodo
                successivo a temp1 */
                temp1->succ_p = elem;

                /* Il campo puntatore al nodo
                precedente del nuovo elemento
                viene associato a temp1 */
                elem->prec_p = temp1;

                /* Viene definito il collegamento
                con il nodo che succede quello
                appena creato,che viene associato
                a temp2 */
                elem->succ_p = temp2;

                /* temp2 viene collegato al nodo
```

```
                        appena creato che lo precede */
                        temp2->prec_p = elem;
                    }
                }
            }
}

/* Funzione di visita della lista, la quale riceve in
input la posizione del nodo da cui iniziare la visita, la testa
della lista e un valore usato per scegliere la direzione da
seguire (avanti/indietro) */
void visita(int posizione, nodo_t *testa, int direz) {

        /* Puntatore ausiliario per scorrere la lista */
        nodo_t *corr = NULL;

        /* Variabile usata per notificare l'assenza
        della posizione richiesta dall'utente */
        int errore = posizione;

        /* Ciclo iterativo, il puntatore ausiliario viene
        inizializzato con l'indirizzo del primo elemento della
        lista; si scorre la lista fino a quando la variabile
        posizione non risulta essere uguale ad 1, questo implica
        l'arrivo nella posizione richiesta dall'utente */
        for(corr = testa; ((corr != NULL) && (posizione != 1));
                corr = corr->succ_p)

                /* La var. posizione viene decrementata di una
                unità ad ogni passo del ciclo */
                posizione--;

        /* Caso in cui si scorre fino al termine della lista,
        senza raggiungere la posizione richiesta dall'utente */
        if(corr == NULL)

                /* Viene notificato un messaggio di errore */
                printf("Posizione %d non presente\n", errore);
        else {

                /* Visita della lista */
                printf("Visita");

                /* Se la var. direz ha valore 8,
                la visita viene effettuata all'avanti */
                if(direz == 8)

                        /* Si invoca la funzione mostra,
                        che visualizza la lista in avanti
                        partendo dalla posizione del
                        puntatore ausiliario corr */
                        mostra(corr);

                /* Se la var. direz ha valore diverso da 8,
                la visita viene effettuata all'indietro */
                else

                        /* Si invoca la funzione mostra_ind,
                        che visualizza la lista all'indietro
```

```
                        partendo dalla posizione del
                        puntatore ausiliario corr */
                        mostra_ind(corr);

                /* Lascia uno spazio */
                printf("\n");
        }
}

/* Funzione utilizzata per contare le occorrenze, i parametri
necessari risultano essere il primo nodo della lista e il valore
di cui si vogliono contare le occorrenze */
int conta(nodo_t *testa, int val) {

        /* Puntatore ausiliario per lo scorrimento della lista */
        nodo_t *corr = testa;

        /* Variabile contatore delle occorrenze di un valore
        all'interno della lista */
        int n_occ = 0;

        /* Valore di ritorno della funzione, che contiene la
        scelta dell'utente nel caso di più occorrenze dello
        stesso valore nei nodi della lista */
        int scelta = 1;

        /* Ciclo che scorre l'intera la lista, lo scopo è
        quello di contare le occorrenze all'interno della
        lista del valore richiesto dall'utente */
        while(corr != NULL) {

                /* Ogni volta che la variabile val viene
                incontrata in un nodo */
                if(corr->dato == val)

                        /* La var. n_occ viene incrementata di
                        un unità */
                        n_occ++;

                /* Incremento del while */
                corr = corr->succ_p;
        }

        /* Se si rileva più di un occorrenza di val nella lista */
        if(n_occ > 1) {

                /* Variabile per la validazione degli input */
                int lettura;

                /* Variabile per svuotare il buffer */
                char rimosso;

                /* L'utente viene avvertito che sono state trovate
                n occorrenze all'interno della lista */
                printf("Sono presenti %d occorrenze di %d\n",
                        n_occ, val);

                /* Acquisizione degli input con validazione */
                do {
```

```
                        /* Scelta dell'occorrenza desiderata */
                        printf("Quale selezionare?\n");
                        lettura = scanf("%d", &scelta);

                        /* Verifica la validità degli input */
                        if(lettura != 1 || scelta < 1 ||
                          scelta > n_occ) {
                                printf("\nSelezione impossibile\n");
                                do
                                        rimosso = getchar();
                                while(rimosso != '\n');
                        }

                  /* Condizioni di ripetizione */
                } while (lettura != 1 || scelta < 1 ||
                        scelta > n_occ);
        }
        /* La funzione restituisce l'occorrenza desiderata */
        return scelta;
}

/* Funzione ricorsiva che mostra la lista scorrendola in avanti,
l'unico parametro richiesto è l'indirizzo del primo nodo */
void mostra(nodo_t *p) {

        /* Se il nodo è diverso da NULL */
        if(p != NULL){

                /* IL nodo viene stampato a video */
                printf(" -> %d", p->dato);

                /* La funzione viene invocata
                sul nodo successivo */
                mostra(p->succ_p);
        }
}

/* Funzione ricorsiva che mostra la lista scorrendola all'indietro,
l'unico parametro risulta essere l'indirizzo del primo nodo */
void mostra_ind(nodo_t *p) {

        /* Se il nodo è diverso da NULL */
        if(p != NULL){

                /* Il nodo viene stampato a video */
                printf(" -> %d", p->dato);

                /* La funzione viene richiamata
                sul nodo precedente */
                mostra_ind(p->prec_p);
        }
}
```

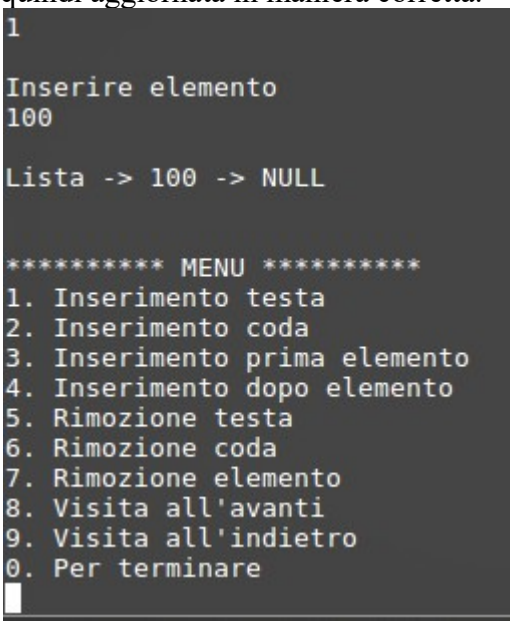
Infine, il codice della libreria *lista.h*:

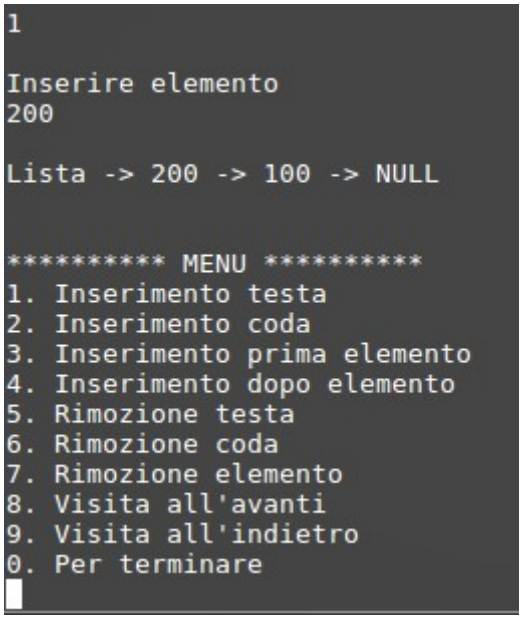
```
/* Definizione del nuvo tipo struttura,
costituita da un campo intero e due
puntatori per il doppio collegamento */
typedef struct nodo  {
        int dato;               /* valore del nodo */
        struct nodo* succ_p;/* puntatore elem. succ. */
        struct nodo* prec_p;/* puntatore elem. prec. */
}nodo_t;                    /* ridefinizione nuovo tipo */

/* Funzioni da esportare */
extern nodo_t *crea_nodo(int);
extern void in_testa(int, nodo_t **);
extern void in_coda(int, nodo_t **);
extern void mostra(nodo_t *);
extern void mostra_ind(nodo_t *);
extern void rim_testa(nodo_t **);
extern void rim_coda(nodo_t **);
extern void rim_elem(int, nodo_t **);
extern void ins_spec(int, int, nodo_t **, int);
extern void visita(int, nodo_t *, int);
```

Testing del programma.

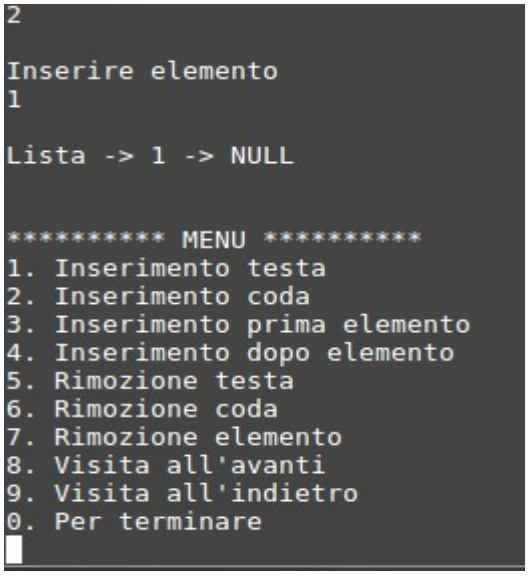
Testing del programma:

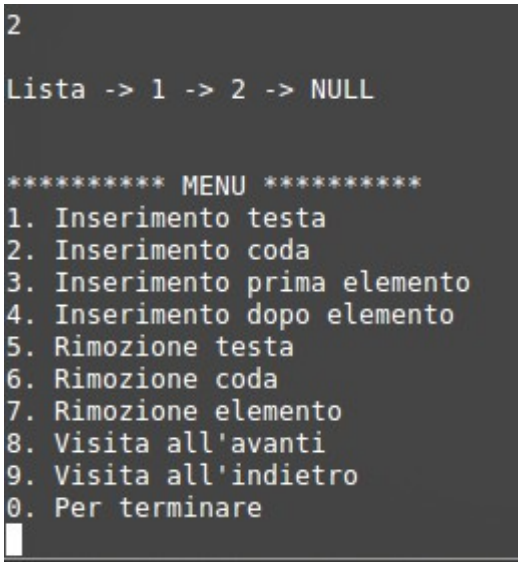
1. Scelta del primo algoritmo(1), per inserire a inizio lista:
   vengono inseriti gli elementi 100 e 200 in questo ordine, la rappresentazione a video è corretta perché il secondo elemento inserito(200) precede il primo, la testa della lista viene quindi aggiornata in maniera corretta.



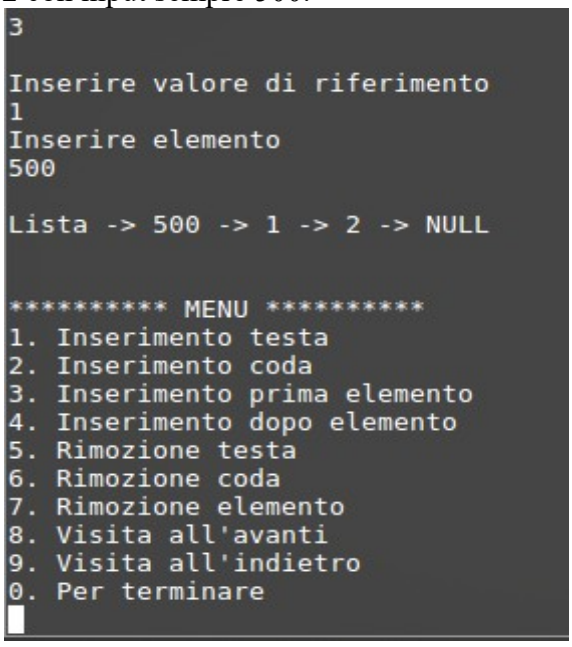


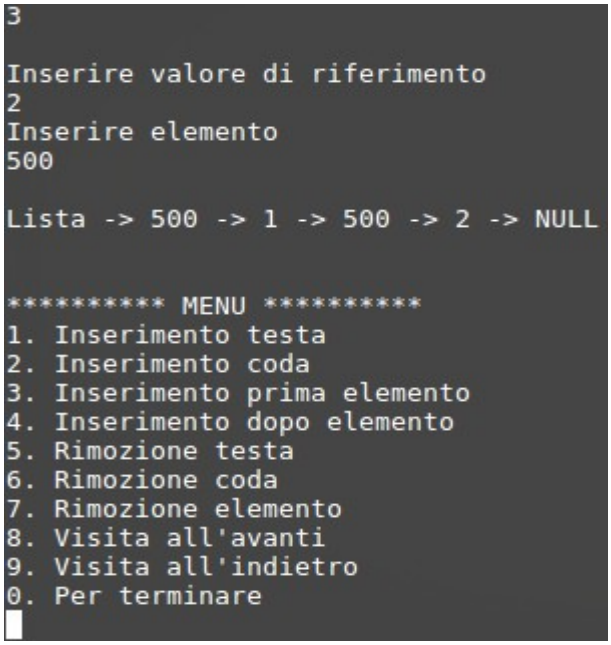
2. Scelta del secondo algoritmo(2), per inserire alla fine della lista:
   vengo inseriti gli elementi 1 e 2 in questo ordine, il secondo succede il primo che non cambia la sua posizione, la rappresentazione risulta quindi corretta.



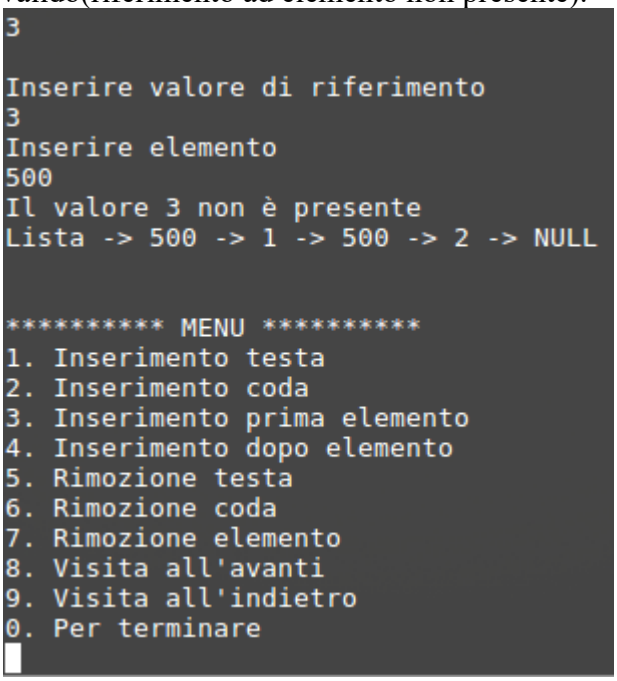


3. Scelta del terzo algoritmo(3), per inserire un elemento prima di un elemento dato: riprendiamo la lista del caso precedente, prendiamo come valore di riferimento l' 1 della nostra lista e vegliamo inserire il 500. La stessa operazione viene eseguita anche sul numero 2 con input sempre 500.





L 'inserimento avviene in maniera corretta sia come nuovo primo elemento che in mezzo ad altri, la rappresentazione è quella desiderata.
Successivamente proviamo ad ottenere un messaggio di errore inserendo un input non valido(riferimento ad elemento non presente).



Il comportamento risulta essere quello previsto.
Verifichiamo anche il caso in cui nella lista siano presenti più occorrenze del valore al quale si sta facendo riferimento per inserire, per farlo utilizziamo la lista composta di soli 1, con l' intento di inserire il numero 555 al centro:

```
Inserire valore di riferimento
1
Inserire elemento
555
Sono presenti 10 occorrenze di 1
Quale selezionare?
6

Lista -> 1 -> 1 -> 1 -> 1 -> 1 -> 555 -> 1 -> 1 -> 1 -> 1 -> 1 -> NULL


********** MENU **********
1. Inserimento testa
2. Inserimento coda
3. Inserimento prima elemento
4. Inserimento dopo elemento
5. Rimozione testa
6. Rimozione coda
7. Rimozione elemento
8. Visita all'avanti
9. Visita all'indietro
0. Per terminare
```

Le operazioni sono state eseguite correttamente.

4. Scelta del quarto algoritmo(4), per inserire un elemento subito dopo un elemento dato: viene ripresa la lista dei casi precedenti (500 -> 1 -> 500 -> 2), vengono presi come elementi di riferimento i numeri 2 e successivamente 1, per provare un inserimento come nuova coda della lista ed uno in un altro punto qualsiasi.

```
4

Inserire valore di riferimento
2
Inserire elemento
600

Lista -> 500 -> 1 -> 500 -> 2 -> 600 -> NULL


********** MENU **********
1. Inserimento testa
2. Inserimento coda
3. Inserimento prima elemento
4. Inserimento dopo elemento
5. Rimozione testa
6. Rimozione coda
7. Rimozione elemento
8. Visita all'avanti
9. Visita all'indietro
0. Per terminare
```

```
4

Inserire valore di riferimento
1
Inserire elemento
600

Lista -> 500 -> 1 -> 600 -> 500 -> 2 -> 600 -> NULL


********** MENU **********
1. Inserimento testa
2. Inserimento coda
3. Inserimento prima elemento
4. Inserimento dopo elemento
5. Rimozione testa
6. Rimozione coda
7. Rimozione elemento
8. Visita all'avanti
9. Visita all'indietro
0. Per terminare
```

Il risultato è quello sperato, gli input inseriti(600) succedono i nodi 1 e 2.

Un altro test per verificare i messaggi di errore.

```
4

Inserire valore di riferimento
3
Inserire elemento
600
Il valore 3 non è presente
Lista -> 500 -> 1 -> 600 -> 500 -> 2 -> 600 -> NULL


********** MENU **********
1. Inserimento testa
2. Inserimento coda
3. Inserimento prima elemento
4. Inserimento dopo elemento
5. Rimozione testa
6. Rimozione coda
7. Rimozione elemento
8. Visita all'avanti
9. Visita all'indietro
0. Per terminare
```

Analogamente a caso 3 verifichiamo il caso in cui nella lista siano presenti più occorrenze del valore al quale si sta facendo riferimento per inserire, riutilizziamo la lista composta da soli 1, con l' intento di inserire il numero 777 al centro:

```
Inserire valore di riferimento
1
Inserire elemento
777
Sono presenti 10 occorrenze di 1
Quale selezionare?
5

Lista -> 1 -> 1 -> 1 -> 1 -> 1 -> 777 -> 1 -> 1 -> 1 -> 1 -> 1 -> NULL


********** MENU **********
1. Inserimento testa
2. Inserimento coda
3. Inserimento prima elemento
4. Inserimento dopo elemento
5. Rimozione testa
6. Rimozione coda
7. Rimozione elemento
8. Visita all'avanti
9. Visita all'indietro
0. Per terminare
```

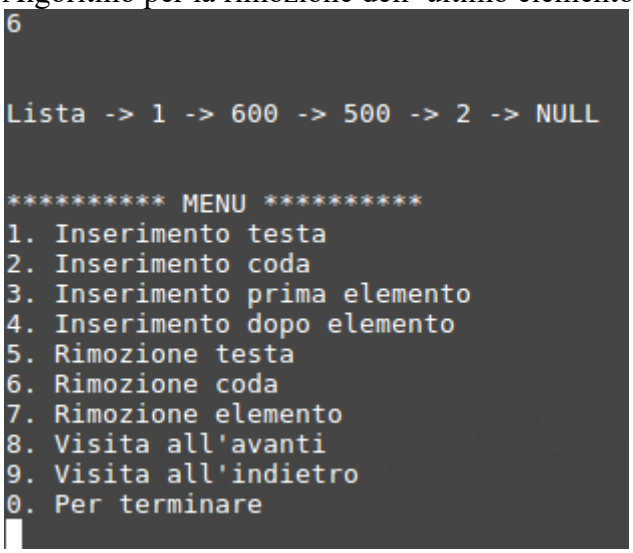
Il nuovo nodo è stato inserito subito dopo il quinto, ovvero al centro della lista.

5. Algoritmo per la rimozione del primo elemento della lista(5):
   continuando sulla stessa lista si esamina il comportamento della scelta numero 5 rimuovendo la testa della lista, e a seguire al punto 6 la rimozione della coda.

```
5


Lista -> 1 -> 600 -> 500 -> 2 -> 600 -> NULL


********** MENU **********
1. Inserimento testa
2. Inserimento coda
3. Inserimento prima elemento
4. Inserimento dopo elemento
5. Rimozione testa
6. Rimozione coda
7. Rimozione elemento
8. Visita all'avanti
9. Visita all'indietro
0. Per terminare
```

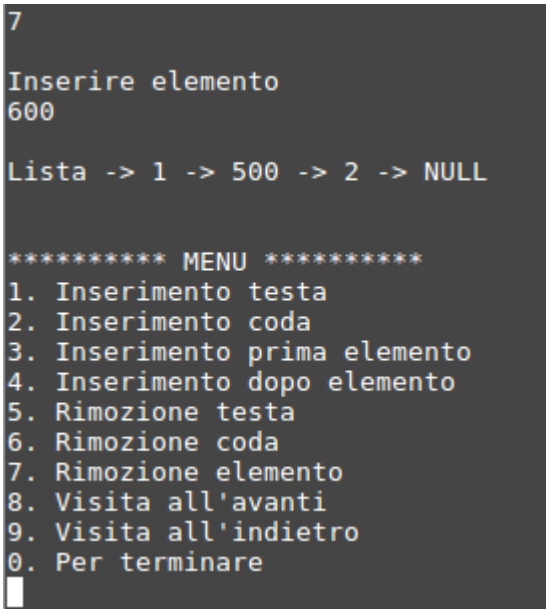
Il primo elemento (500) viene rimosso.

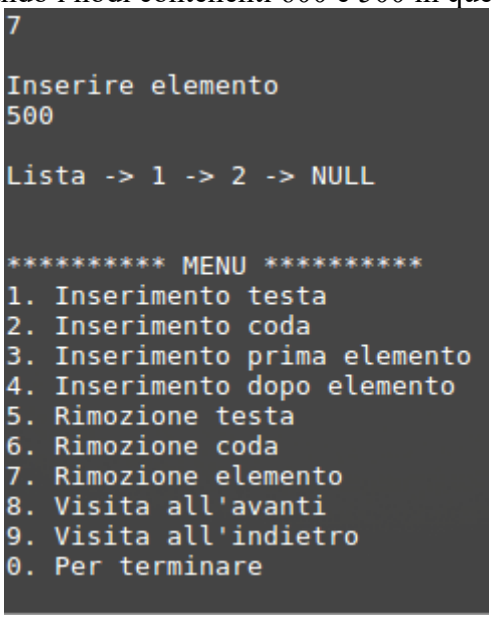
6. Algoritmo per la rimozione dell' ultimo elemento della lista(6):



Anche la coda della lista(600) viene rimossa correttamente.

7. Scelta dell'algoritmo per l'eliminazione dei vari nodi(7):
verifichiamo la corretta esecuzione eliminando i nodi contenenti 600 e 500 in questo ordine.





Eliminiamo anche i nodi 1 e 2 per ottenere una lista vuota, e si testiamo i messaggi di errore.

```
7

Inserire elemento
2

Lista -> NULL


********** MENU **********
1. Inserimento testa
2. Inserimento coda
3. Inserimento prima elemento
4. Inserimento dopo elemento
5. Rimozione testa
6. Rimozione coda
7. Rimozione elemento
8. Visita all'avanti
9. Visita all'indietro
0. Per terminare
```

```
7

Inserire elemento
123
Lista vuota!
Lista -> NULL


********** MENU **********
1. Inserimento testa
2. Inserimento coda
3. Inserimento prima elemento
4. Inserimento dopo elemento
5. Rimozione testa
6. Rimozione coda
7. Rimozione elemento
8. Visita all'avanti
9. Visita all'indietro
0. Per terminare
```

Il programma si comporta correttamente.
Infine testiamo il comportamento dell'algoritmo nel caso in cui siano presenti più occorrenze dello stesso numero da eliminare.
Creiamo una lista composta da 5 elementi i quali contengono 0 e 1 in forma alternata (partendo con 0), l'obbiettivo è quello di ottenere due 1 consecutivi.
Lista di partenza:

```
Lista -> 0 -> 1 -> 0 -> 1 -> 0 -> NULL


********** MENU **********
1. Inserimento testa
2. Inserimento coda
3. Inserimento prima elemento
4. Inserimento dopo elemento
5. Rimozione testa
6. Rimozione coda
7. Rimozione elemento
8. Visita all'avanti
9. Visita all'indietro
0. Per terminare
```

Lista modificata:

```
7

Inserire elemento
0
Sono presenti 3 occorrenze di 0
Quale selezionare?
2

Lista -> 0 -> 1 -> 1 -> 0 -> NULL


********** MENU **********
1. Inserimento testa
2. Inserimento coda
3. Inserimento prima elemento
4. Inserimento dopo elemento
5. Rimozione testa
6. Rimozione coda
7. Rimozione elemento
8. Visita all'avanti
9. Visita all'indietro
0. Per terminare
```

L'obbiettivo è stato raggiunto.

8. Visita della lista scorrendo verso destra(8):
   per questo test creiamo una lista ordinata contenente numeri nell'intervallo [1, 10], con l'obbiettivo di effettuarne la visita partendo dalla quinta posizione, ossia il nodo numero 5.

```
8

Inserire posizione
5
Visita -> 5 -> 6 -> 7 -> 8 -> 9 -> 10

Lista -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> NULL


********** MENU **********
1. Inserimento testa
2. Inserimento coda
3. Inserimento prima elemento
4. Inserimento dopo elemento
5. Rimozione testa
6. Rimozione coda
7. Rimozione elemento
8. Visita all'avanti
9. Visita all'indietro
0. Per terminare
```

La visita ci mostra metà della nostra lista, che è quello che ci aspettavamo.

Verifichiamo i messaggi di errore chiedendo di visitare a partire da un elemento(11) non presente.

```
8

Inserire posizione
11
Posizione 11 non presente

Lista -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> NULL


********** MENU **********
1. Inserimento testa
2. Inserimento coda
3. Inserimento prima elemento
4. Inserimento dopo elemento
5. Rimozione testa
6. Rimozione coda
7. Rimozione elemento
8. Visita all'avanti
9. Visita all'indietro
0. Per terminare
```

9. Algoritmo per la visita della lista scorrendo verso destra(9):
   analogamente al caso precedente, utilizzando la lista ordinata si effettua una visita partendo dal nodo centrale, e si effettua un inserimento di posizione inesistente(12) per i messaggi di errore.

```
9

Inserire posizione
5
Visita -> 5 -> 4 -> 3 -> 2 -> 1

Lista -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> NULL


********** MENU **********
1. Inserimento testa
2. Inserimento coda
3. Inserimento prima elemento
4. Inserimento dopo elemento
5. Rimozione testa
6. Rimozione coda
7. Rimozione elemento
8. Visita all'avanti
9. Visita all'indietro
0. Per terminare
```

```
9

Inserire posizione
12
Posizione 12 non presente

Lista -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> NULL


********** MENU **********
1. Inserimento testa
2. Inserimento coda
3. Inserimento prima elemento
4. Inserimento dopo elemento
5. Rimozione testa
6. Rimozione coda
7. Rimozione elemento
8. Visita all'avanti
9. Visita all'indietro
0. Per terminare
```

Viene stampata la lista inversa nel primo caso ed i messaggi di errore nel secondo, che sono i risultati desiderati

10. Termina il programma(0):
    Nel menu di scelta premiamo 0 per terminare

```
0


Lista -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> NULL

daniele@daniele-VirtualBox ~/progetto2 $
```

Termina correttamente.

Verifica del programma.

Verifica del programma:

Segmento di codice verificato:

```
for(corr = testa; ((corr != NULL) && (posizione != 1));
        corr = corr → succ_p)
        posizione--;
```

Postcondizione R:

$corr = (corr \rightarrow succ\_p \ \wedge \ posizione - 1)$

Invariante di ciclo:

$(corr \neq \ NULL \wedge \ posizione \neq 1)$

Funzione tr(i):

$(dim - corr \rightarrow succ\_p)$

Predicato P:
$(testa \leq corr \rightarrow succ\_p \leq dim) \wedge (corr \rightarrow succ\_p \wedge posizione - 1)$

Ipotesi di limitatezza:

$(P \wedge tr(i) \leq 0) =$
$= (testa \leq corr \rightarrow succ\_p \leq dim) \wedge (corr \rightarrow succ\_p \wedge posizione - 1) \wedge \ (dim - corr) \ \leq$

$\leq 0 \equiv (corr = dim \wedge (corr \rightarrow succ\_p \wedge posizione - 1))$

$\models corr \nleqslant dim$

Poiché:
$(p \wedge \ corr \nleqslant dim) = (testa \leq corr \leq dim \wedge \ corr \rightarrow succ\_p) \equiv$

$\equiv (corr = dim \wedge \ corr \rightarrow succ\_p)$

$\models R$

P viene usato come precondizione dell'intera istruzione di ripetizione

$Pcorr,0 = (0 \leq testa \leq dim \wedge corr \rightarrow succ\_p) \equiv (corr = 0)$

$P'corr \rightarrow succ\_p = (testa = 0) \equiv vero$