Università degli Studi di Urbino "Carlo Bo"

Corso di Laurea in Informatica Applicata

Progetto di

Programmazione Procedurale e Logica

Sviluppatore:

Angelo Sasso
Matricola n° 263143
a.sasso4@campus.uniurb.it

A.A. 2016/2017

Capitolo 1

Specifiche 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.

Capitolo 2

Analisi del problema

2.1 Input

Gli input presi in considerazione da questo programma, sono identificati dagli input delle nove funzioni che la libreria deve esportare. Tutti gli input da tastiera sono di tipo intero, inoltre ogni funzione prende in input la lista stessa.
Le funzioni di inserimento *inserisci_testa* e *inserisci_coda* hanno come input il valore dell'informazione del nuovo nodo; le funzioni *inserisci_prima* e *inserisci_dopo* richiedono due input: la posizione prima o dopo del quale inserire il nuovo nodo e successivamente l'informazione che conterrà.
Le funzioni di rimozione *cancella_testa* e *cancella_coda* non richiedono input in quanto, se la lista non è vuota, gli elementi di testa e di coda sono già definiti.
La funzione di rimozione *cancella_info* riceve in input un intero che è il valore dell'informazione da cercare e rimuovere.
Le funzioni di visita e stampa della lista *stampa_avanti* e *stampa_indietro* ricevono in input un intero che sarà la posizione dal quale partire la visita seguendo poi il verso in base alla funzione.

2.2 Output

Gli output da prendere in considerazione sono gli output delle nove funzioni che compongono la libreria.
Le funzioni che modificano la lista restituiscono la lista modificata mentre le funzioni di stampa sono di tipo void.
Le funzioni di inserimento quali *inserisci_testa, inserisci_coda, inserisci_prima* e *inserisci_dopo* hanno tutte come output una nuova informazione nella lista. Le funzioni di rimozione *cancella_testa, cancella_coda* e *cancella_info* hanno come output una modifica delle informazioni nella lista; la funzione *cancella_info* in particolare modifica più volte la lista a seconda di quante volte ricorre l'informazione.
Le funzioni di visita e stampa della lista *stampa_avanti* e *stampa_indietro* hanno come output la stampa a video della lista.

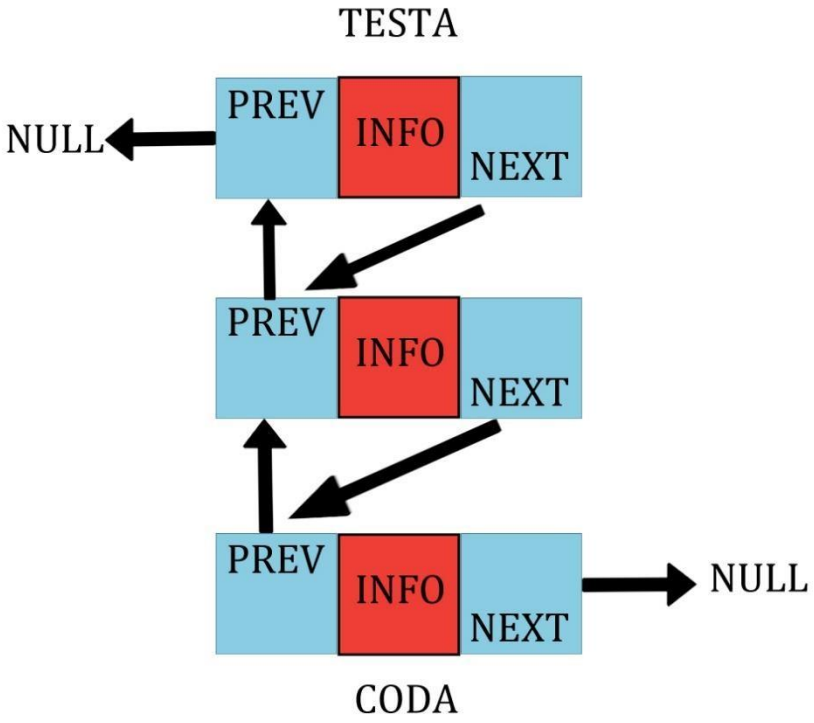
2.3 Relazioni

Le relazioni tra input e output sono le segnalazioni d’errore stampate a video dalle funzioni esportate che possono manifestarsi se l’utente inserisce informazioni che non siano un intero o ancora se seleziona una funzione non disponibile.
Dopo ogni messaggio d’errore è possibile continuare l’esecuzione del programma.

Teoria sulle liste

Una lista doppiamente concatenata è una collezione di oggetti numerati e successivi.
Essa può essere consultata con l'ausilio di uno o più puntatori ad elementi della lista.
Tramite l'utilizzo di opportuni algoritmi è possibile quindi posizionarsi su di un elemento per modificare l'informazione contenuta in esso o per rimuoverla o per aggiungere un elemento.
La struttura dati da utilizzare deve essere dinamica e lineare.
Lista Doppia Puntata è un insieme dinamico in cui ogni elemento ha uno o più campi contenenti informazioni e due riferimenti, uno all'elemento successivo della lista ed uno all'elemento precedente.
Le liste doppiamente concatenate in particolare permettono di scorrere facilmente gli elementi in entrambi i versi.

Segue un semplice esempio grafico di lista doppiamente concatenata:



Capitolo 3

Progettazione dell'algoritmo

3.1 Scelte di progetto

Si è tenuto presente che i file sorgente devono essere articolati in funzioni. Le scelte di progetto considerate, fanno riferimento a:

1. Utilizzo di due strutture dati;
2. Creazione degli elementi;
3. Rimozione degli elementi;
4. Visita degli elementi;
5. Segnalazione errori;
6. Indentazione del codice.

1. Si chiede di progettare una struttura dati dinamica e lineare che rispecchi quella di una lista doppiamente concatenata.
   La struttura contiene quindi due puntatori ad elemento e un campo intero. Su questa deve essere possibile effettuare delle operazioni.
   Utile alla gestione dei puntatori della lista e della grandezza della stessa è la struttura dati dinamica *lista* che contiene due puntatori ausiliari per scorrere la lista e permettere la modifica di elementi anche successivi tramite uno scambio di informazioni tra essi, inoltre contiene un campo intero per tenere il conto della lunghezza della lista che viene aggiornata dalle funzioni di inserimento e rimozione.
   La creazione e la rimozione degli elementi avverrà dinamicamente.

2. Inizialmente la lista sarà vuota, in base alla funzione di inserimento scelta, una funzione di libreria *"crea"* che comprende la validazione stretta dell'informazione. Essa allocherà lo spazio di un nuovo elemento e la funzione si occuperà di collegare i puntatori in base alla posizione dove sarà inserito l'elemento.
   Questa funzione è stata implementata per ridurre il codice che sarebbe risultato ridondante nelle quattro funzioni di inserimento.

3. Per rimuovere uno o più elementi dalla lista è necessario che essa non sia vuota. Tutte le funzioni di rimozione valutano i differenti stati della lista e operano di conseguenza ricollegando i puntatori dei nodi e liberando la parte di memoria dedicata al nodo.
   È stato riscontrato un caso particolare per la rimozione dei nodi:
   se un'informazione è presente più volte nella lista, l'algoritmo procederà con la rimozione di tutti i nodi contenenti tale informazione.
   Si è scelto di stampare a video la conferma di avvenuta eliminazione e il numero di nodi della lista dopo ogni rimozione. Se le rimozioni sono multiple, questa stampa viene aggiornata più volte.

4. Per effettuare le operazioni sulla lista è necessario effettuare una visita.
   Queste operazioni vengono agevolate dalla struttura di supporto contenente i puntatori.
   Se la lista è vuota questi vengono inizializzati a *null*, altrimenti con l'utilizzo di un ciclo *for* si scorrono gli elementi e con un ciclo *while* è possibile riconoscere e raggiungere la fine della lista.

5. L’utente deve inserire informazioni pertinenti al tipo richiesto *(intero)*, ma può capitare che vengano inseriti accidentalmente dei tipi diversi.
   Tutte le funzioni dispongono di validazioni e controlli riguardanti la grandezza della lista e la stampa a video della situazione.
   Ogni acquisizione da tastiera è gestita da una variabile di tipo intero*, esito,* che stampa a video un messaggio di errore in caso di tipi non accettati*.*
   Inoltre se l’utente inserisce una posizione che non rientra nella grandezza della lista questo viene stampato a video*.*

6. Tra i vari stili d’indentazione è stato scelto l’utilizzo dello stile Allman (chiamato anche stile BSD in ricordo di alcune utility BSD Unix da lui progettate). Le parentesi graffe di apertura associate ad una istruzione di controllo del flusso si trovano su una nuova riga, allo stesso livello di indentazione dell'istruzione, mentre le istruzioni nel blocco hanno un livello maggiore di indentazione.

3.2 Progettazione

Le funzioni della libreria “lista_dc.c” da progettare sono nove di cui quattro adibite all’inserimento degli elementi, tre di rimozione e due di visita.
Verranno utilizzate molteplici validazioni per permettere l’esecuzione del programma anche in caso di errore da parte dell’utente.

1. La funzione *inserisci_testa* può essere chiamata sia nel caso in cui la lista sia vuota sia che sia popolata e dopo la chiamata alla funzione *crea* collega il nuovo elemento in testa alla lista.

2. Per la funzione *inserisci_coda* il discorso è analogo ma i puntatori devono essere collegati in modo che il nuovo elemento sia la coda della lista.

3. La funzione *inserisci_prima* utilizza un intero per memorizzare la posizione scelta dall’utente. Successivamente vengono valutati vari casi. Se l’inserimento va a buon fine, la lunghezza della lista viene incrementata e il nuovo nodo avrà i puntatori collegati in modo da risultare prima della posizione scelta.

4. La funzione *inserisci_dopo* in maniera analoga utilizza un intero per memorizzare la posizione scelta dall’utente. Successivamente vengono valutati vari casi. Se l’inserimento va a buon fine, la lunghezza della lista viene incrementata e il nuovo nodo avrà i puntatori collegati in modo da risultare dopo della posizione scelta.

5. La funzione *cancella_testa* utilizza un puntatore temporaneo per posizionarsi sull’elemento di testa, successivamente modifica i puntatori dell’elemento successivo e lo marca come testa della lista, aggiorna la lunghezza della lista e libera la memoria allocata.

6. La funzione *cancella_coda* utilizza il puntatore temporaneo per posizionarsi sull’elemento di coda, successivamente modifica i puntatori dell’elemento precedente e lo marca come coda della lista, la lunghezza della lista viene decrementata mentre la memoria precedentemente allocata viene liberata.

7. La funzione *cancella_info* utilizza una validazione stretta dell’input e un ciclo per scorrere tutti gli elementi.
   Dopo ogni rimozione vengono effettuate due stampe a video differenti;
   Se la rimozione va a buon fine verrà visualizzata la conferma e il numero di nodi della lista, oppure un messaggio di errore nel caso in cui l’utente accidentalmente non inserisce un valore intero.
   Se l’informazione è contenuta nel nodo di testa o nel nodo di coda vengono chiamate rispettivamente la funzione *cancella_testa* e *cancella_coda.*

8. La funzione *stampa_avanti* si occupa della visita e della stampa a video degli elementi della lista. Tramite questa funzione è possibile conoscere le informazioni memorizzate dall'utente in fase di utilizzo del programma.
   Viene acquisito un intero da tastiera che sarà la posizione dal quale iniziare la stampa. Grazie all'uso della validazione stretta è possibile stampare a video un messaggio di errore se l'utente accidentalmente non inserisce un valore intero. Per visitare la lista si valutano più casi e se la lista è vuota viene stampato a video un messaggio informativo, altrimenti il puntatore temporaneo scorre gli elementi per poi stamparli a partire dalla posizione scelta fino alla coda della lista nel verso del nodo successivo (avanti).

9. Anche la funzione *stampa_indietro* comprende la visita e la stampa della lista creata in fase di utilizzo dall'utente ed utilizza gli stessi ciclo e controlli della funzione *stampa_avanti* ma in questo caso la stampa partirà dalla posizione scelta dall'utente per concludersi con la testa della lista nel verso del nodo precedente (indietro).

   Grazie all'utilizzo di una lista doppiamente concatenata queste funzioni sono di facile progettazione, è necessario ricorrere a soluzioni differenti in caso di liste singolarmente concatenate o liste circolari.

3.3 Schematizzazione

- Avvio del programma;
- Stampa a video del menù di selezione;
- Scelta dell’utente;
- Chiamata a funzione;
- Scelta dell’utente.

Capitolo 4

Implementazione del programma

Il programma è composto da due file sorgente “.c”, da un header file “.h” e dal Makefile.

4.1 Implementazione dei file sorgente “.c”

Il file sorgente della libreria per la gestione della lista doppiamente concatenata denominato “lista_dc.c”.

```
/* File sorgente "lista_dc.c". */

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

/* Definizione dei tipi da esportare e interni. */

/* Viene definito il tipo "nodo" che sarà la struttura predisposta
 * alla gestione di ogni nodo della lista.
 * Contiene un campo intero che ospita l'informazione del nodo e
 * due puntatori che saranno rispettivamente il puntatore all'elemento successivo
 * e il puntatore all'elemento precedente della lista. */
typedef struct   nodo_t
{
        int                 info;
        struct   nodo_t *successivo;
        struct   nodo_t *precedente;
}       nodo;


/* Viene inoltre definito il tipo "lista" che sarà una
 * struttura di sostegno utile alla definizione della lista.
 * Contiene un valore intero che terrà traccia della lunghezza della lista.
 * Vengono definiti quattro puntatori ad elementi particolari della lista
 * quali l'elemento di testa, quello di coda, un puntatore ausiliario
 * e uno temporaneo. */
typedef struct   lista_t
{
        int                 conto;
        struct   nodo_t *testa;
        struct   nodo_t *coda;
        struct   nodo_t *aus;
        struct  nodo_t   *temp;
}       lista;
```

```
/* Dichiarazione delle variabili. */

/* Dichiarazione delle funzioni da esportare e interne. */
lista * inserisci_testa(lista *lista_t);
lista * inserisci_coda(lista *lista_t);
lista * inserisci_prima(lista *lista_t);
lista * inserisci_dopo(lista *lista_t);
void    stampa_avanti(lista *lista_t);
void    stampa_indietro(lista *lista_t);
lista * cancella_info(lista *lista_t);
lista * cancella_testa(lista *lista_t);
lista * cancella_coda(lista *lista_t);
lista * crea(lista *lista_t);
void      svuota_buffer();

/* Definizione delle funzioni da esportare e interne (no main). */

/* La funzione "inserisci_testa" prende in input la lista.
 * Inserisce un nuovo elemento in testa alla lista chiamando
 * la funzione "crea()" per poi ricollegare i puntatori.
 * Restituisce la lista modificata. */
lista * inserisci_testa(lista *lista_t)
{
        if(lista_t->testa == NULL)
        {
                lista_t = crea(lista_t);
                /* Il nuovo elemento è la testa della lista. */
                lista_t->testa = lista_t->temp;
                lista_t->coda = lista_t->testa;
        }
        else
        {
                lista_t = crea(lista_t);
                lista_t -> temp -> successivo = lista_t -> testa;
                lista_t -> temp -> precedente = lista_t -> temp;
                /* Il nuovo elemento è la testa della lista. */
                lista_t->testa = lista_t->temp;
        }
        return(lista_t);
}
```

```
/* La funzione "inserisci_coda" ha la lista in input.
 * Inserisce un nuovo elemento in coda alla lista chiamando
 * la funzione "crea()" per poi ricollegare i puntatori.
 * Restituisce la lista modificata. */
lista * inserisci_coda(lista *lista_t)
{
	/* Caso in cui la lista è vuota. */
	if(lista_t->testa == NULL)
	{
		/* Creo l'elemento. */
		lista_t = crea(lista_t);
		lista_t->testa = lista_t->temp;
		lista_t->testa->successivo = NULL;
		lista_t->testa->precedente = NULL;
		/* Il nuovo elemento sarà la coda e la testa della lista. */
		lista_t->coda = lista_t->testa;
	}

	else
	{
		/* Creo l'elemento. */
		lista_t = crea(lista_t);
		lista_t->aus = lista_t->coda;
		lista_t->coda->successivo = lista_t->temp;
		lista_t->temp->precedente = lista_t->coda;
		/* Il nuovo elemento è la coda della lista. */
		lista_t->coda = lista_t->temp;
		lista_t->coda->successivo = NULL;
	}
	return(lista_t);
}

/* "inserisci_prima_di_posizione()" è la funzione che si occupa di
 * creare un nuovo elemento con l'utilizzo della funzione "crea()"
 * e successivamente posizionarlo prima della posizione scelta dall'utente
 * per poi ricollegare i puntatori ad operazioni concluse. */
lista * inserisci_prima(lista *lista_t)
{
	/* Dichiarazione delle variabili */
	int	posizione;	/* Variabile per acquisire da tastiera la posizione dove inserire l'elemento. */
	int	i = 2;		/* Variabile per il ciclo while. */
	int	esito;		/* Variabile per il controllo dei caratteri. */
```

```
if(lista_t->testa==NULL)
{
	printf("\nLa lista è vuota.\n");
	printf("E' possibile inserire un elemento in posizione 1.\n");
}

/* Stampa per l'acquisizione della posizione. */
do
{
	printf("\nPrima di quale posizione inserire il nuovo nodo?\t");

	/* Controllo dei caratteri inseriti. */
	esito = scanf("%d", &posizione);

	if(esito != 1)
	{
		printf("\nPer favore, inserire una posizione valida (un intero):\t");
		svuota_buffer();
	}
}	while(esito != 1 || posizione < 0);

/* Il puntatore aus punta alla testa della lista. */
lista_t->aus = lista_t->testa;

/* Caso in cui la posizione è minore di 1 o al di fuori della grandezza della lista. */
if((posizione < 1) || posizione >= (lista_t -> conto ))
{
	printf("\nLa posizione è fuori portata.");
}

/* Caso in cui la lista è vuota e la posizione è diversa da 1. */
else if((lista_t->testa == NULL) && (posizione != 1))
{
	printf("\nE' possibile inserire un nuovo nodo solo in testa.");
	printf("\nLa lista è vuota.");
}

/* Caso in cui la lista non è vuota e la posizione è uguale ad 1. */
else if((lista_t->testa != NULL) && (posizione ==1))
{
	lista_t = inserisci_testa(lista_t);
}
```

```
	/* Caso in cui la lista è vuota e la posizione è uguale ad 1. */
	else if((lista_t->testa == NULL) && (posizione == 1))
	{
		lista_t = crea(lista_t);
		lista_t->testa = lista_t->temp;
		lista_t->coda = lista_t->testa;
	}

	else
	{
		/* Ciclo per posizionare il puntatore ausiliario. */
		while (i < posizione)
		{
			lista_t->aus = lista_t->aus->successivo;
			i++;
		}

		/* Creo l'elemento. */
		lista_t = crea(lista_t);

		/* Organizzo i puntatori. */
		lista_t->temp->precedente = lista_t->aus;
		lista_t->temp->successivo = lista_t->aus->successivo;
		(lista_t->aus->successivo)->precedente = lista_t->temp;
		lista_t->aus->successivo = lista_t->temp;
	}
	return(lista_t);
}

/* "inserisci_dopo_di_posizione()" è la funzione che si occupa di
 * creare un nuovo elemento con l'utilizzo della funzione "crea()"
 * e successivamente posizionarlo dopo la posizione scelta dall'utente
 * per poi ricollegare i puntatori ad operazioni concluse. */
lista * inserisci_dopo(lista *lista_t)
{

	/* Dichiarazione delle variabili. */
	int	posizione = 0;		/* Variabile per acquisire da tastiera la posizione dove inserire l'elemento.  */
	int	i = 1;			/* Variabile per il ciclo while. */
	int	esito = 0;		/* Variabile per il controllo dei caratteri. */

	if(lista_t->testa==NULL)
	{
		printf("\nLa lista è vuota.\n");
		printf("E' possibile inserire un elemento in posizione 1.\n");
	}
```

```
/* Stampa per l'acquisizione della posizione. */
do
{
	printf("\nDopo di quale posizione inserire il nuovo nodo?\t");

	/* Controllo dei caratteri inseriti. */
	esito = scanf("%d", &posizione);

	if(esito != 1)
	{
		printf("\nPer favore, inserire una posizione valida (un intero):\t");
		svuota_buffer();
	}
}	while(esito != 1 || posizione < 0);

/* Il puntatore "aus" tiene memoria della testa della lista. */
lista_t->aus = lista_t -> testa;

/* Caso in cui la posizione è minore di 0 o maggiore del numero di elementi della lista */
if((posizione < 0) || (posizione > lista_t->conto))
{
	printf("\nLa posizione è fuori portata.");
}

/* Caso in cui la lista è vuota e la posizione inserita è diversa dalla testa. */
else if((lista_t->testa == NULL) && (posizione != 0))
{
	printf("\nLa lista è vuota.");
}

/* Caso in cui la lista non è vuota e la posizione inserita è la testa. */
else if((lista_t->testa != NULL) && (posizione == 0))
{
	lista_t = inserisci_testa(lista_t);
}

/* Caso in cui la lista è vuota e la posizione inserita è la testa. */
else if((lista_t->testa == NULL) && (posizione == 0))
{
	lista_t = inserisci_testa(lista_t);
}


/* Caso in cui la lista non è vuota e la posizione è la coda. */
else if(lista_t->testa != NULL && posizione == lista_t->conto)
{
	lista_t = inserisci_coda(lista_t);
}
```

```
	/* Caso in cui la lista non è vuota e la posizione è un intero
	   diverso dalla testa o dalla coda della lista. */
	else if((lista_t->testa != NULL) || (posizione != 0))
	{
		while(i < posizione)
		{
			lista_t->aus = lista_t->aus->successivo;
			i++;
		}

		/* Creo l'elemento. */
		lista_t = crea(lista_t);

		/* Collego i puntatori. */
		lista_t->temp->precedente = lista_t->aus;

		/* Se l'elemento è la coda della lista. */
		if(lista_t->aus->successivo == NULL)
		{
			/* Collego i puntatori. */
			lista_t->temp->successivo = NULL;
			lista_t->aus->successivo = lista_t->temp;
		}

		else
		{
			/* Collego i puntatori. */
			lista_t->temp->successivo = lista_t->aus->successivo;
			lista_t->aus->successivo = lista_t->temp->successivo;
		}
	}

	return(lista_t);
}

/* La funzione "stampa_posizione_coda()"
 * visita la lista per poi stamparla
 * dalla posizione scelta dall'utente alla coda. */
void	stampa_avanti(lista *lista_t)
{
	/* Dichiarazione delle variabili. */
	int	i;		/* Variabile per il ciclo for. */
	int	n;		/* Posizione da acquisire da tastiera dal quale iniziare la visita. */

	lista_t->temp = lista_t->testa;		/* Il puntatore temp tiene memoria della testa della lista. */

	/* Stampa per l'acquisizione di un intero, la posizione dal quale iniziare la stampa. */
	printf("\nInserire la posizione dal quale iniziare la visita.");
```

```
/* Ciclo while per il controllo dei caratteri. */
while(scanf("%d", &n) != 1)
{
	printf("\nErrore: posizione non valida.\n");
	printf("Inserire un intero compreso tra 1 e %d (inclusi).\t", lista_t->conto);
	getchar();
}

/* Caso in cui la lista è vuota e la posizione è diversa da 0. */
if(lista_t->testa == NULL && n != 0)
{
	printf("\nLa lista è vuota.");
}

/* Caso in cui la posizione sia al di fuori della grandezza della lista. */
else if(n > lista_t->conto)
{
	printf("\nLa posizione è fuori portata.");
}

/* Caso in cui la posizione scelta è uguale a 0. */
else if(n == 0)
{
	printf("\nLa posizione 0 non è accettata, la testa della lista ha come indice '1'.");
}

else
{
	printf("\nLista dalla posizione '%d' in avanti :\n", n);
	printf(" [");

	/* Ciclo for per incrementare "i" scorrendo gli elementi. */
	for(i=0; i<=n-2; i++)
	{
		/* Il puntatore temp scorre gli elementi. */
		lista_t->temp = lista_t->temp->successivo;
	}

	while(lista_t->temp->successivo != NULL)
	{
		printf(" (Informazione: %d) ", lista_t->temp->info);
		lista_t->temp = lista_t->temp->successivo;
	}
```

```
                /* Stampa dell'elemento di coda. */
                printf(" (Informazione: %d) ] Coda", lista_t->temp->info);
                printf("\n\t\tLunghezza della lista: %d.\n", lista_t->conto);

        }
}

/* La funzione "stampa_posizione_testa()"
 * visita la lista e la stampa
 * dalla posizione scelta dall'utente alla testa. */
void    stampa_indietro(lista *lista_t)
{
        /* Dichiarazione delle variabili. */
        int     i;              /* Variabile per per incrementare i puntatori nel ciclo for. */
        int     n;              /* Variabile per la posizione dal quale iniziare la visita. */

        /* Il puntatore temp tiene memoria della testa della lista. */
        lista_t->temp = lista_t->testa;

        /* Stampa per l'acquisizione della posizione dal quale iniziare la visita. */
        printf("\nInserire la posizione dal quale iniziare la visita:\t");

        /* Ciclo per il controllo dei caratteri. */
        while(scanf("%d", &n) != 1)
        {
                printf("\nErrore: posizione non valida.\n");
                printf("Inserire un intero compreso tra 1 e %d (inclusi).\t", lista_t->conto);
                getchar();
        }

        /* Caso in cui la lista è vuota. */
        if(lista_t->testa == NULL)
        {
                printf("\nLa lista è vuota.");
                getchar();
        }

        /* Caso in cui la posizione sia al di fuori della grandezza della lista. */
        else if(n > lista_t->conto)
        {
                printf("\nLa posizione è fuori portata.");
                getchar();
        }

        /* Caso in cui la posizione è uguale a 0. */
        else if(n==0)
        {
                printf("\nLa posizione 0 non è accettata, \nla lista ha come inizio la testa con indice '1'.");
        }
```

```
	else
	{
		printf("\nLista dalla posizione '%d' indietro :", n);
		printf(" [");

		/* Ciclo for per incrementare "i" scorrendo gli elementi. */
		for(i=0; i<n-1; i++)
		{
			/* Il puntatore temp scorre gli elementi. */
			lista_t->temp = lista_t->temp->successivo;
		}

		while(lista_t->temp->precedente != NULL)
		{
			printf(" (Informazione: %d) ", lista_t->temp->info);
			lista_t->temp = lista_t->temp->precedente;
		}

		/* Stampa del primo elemento della lista. */
		printf(" (Informazione: %d) ] Testa ", lista_t->temp->info);
		printf("\n\t\tLunghezza della lista: %d.\n", lista_t->conto);
	}
}

/* "cancella_info()" acquisisce da tastiera un intero
 * e visita la lista in cerca dell'informazione inserita.
 * Se l'informazione viene trovata in testa alla lista
 *	la funzione utilizza "cancella_testa()" per eliminare
 *	l'elemento dalla lista.
 * Se l'informazione viene trovata in coda alla lista
 *	invece verra utilizzata "cancella_coda()" per eliminare
 *	l'elemento dalla lista
 * Se l'informazione non corrisponde ai puntatori di testa e coda
 *	la funzione provvederà a cancellare l'elemento tramite "free()"
 *	per poi ricollegare i puntatori.
 * Se infine l'informazione non è presente nella lista
 *	questo verrà stampato a video. */
lista *cancella_info(lista *lista_t)
{
	/* Dichiarazione delle variabili. */
	int	esito;
	int	valore;		/* Variabile che conterrà l'informazione da eliminare. */
	int	i;		/* Variabile per verificare l'eliminazione di uno o più elementi. */

	/* Questa variabile manterrà il valore della lunghezza
	 * della lista prima dell'eleminazione di uno o più elementi. */
	i = lista_t->conto;
	lista_t->temp = lista_t->testa;		/* Il puntatore temp punta alla testa della lista. */
```

```
/* Se la lista è vuota non è possibile cancellare un'informazione. */
if(lista_t->temp == NULL)
{
	printf("\nLa lista è vuota.");
}

else

/* Stampa per l'acquisizione del valore da eliminare. */
do
{
	printf("\nInserire l'informazione da eliminare:\t");
	/* Controllo dei caratteri inseriti. */
	esito = scanf("%d", &valore);

	if(esito != 1)
	{
		printf("\nPer favore, inserire un'informazione valida (un intero):\t");
		svuota_buffer();
	}
}	while(esito != 1);

while(lista_t->temp != NULL)
{
	if(lista_t->temp->info != valore)
	{
		lista_t->temp = lista_t->temp->successivo;
	}

	else if(lista_t->temp->info == valore)
	{
		if(lista_t->temp == lista_t->testa && lista_t->temp->info == valore)
		{
			lista_t = cancella_testa(lista_t);
			/* Stampa a video della lunghezza della lista. */
			printf("\n\tNumero di nodi: %d.\n", lista_t->conto);

		}

		else if(lista_t->temp == lista_t -> coda && lista_t->temp->info == valore)
		{
			lista_t = cancella_coda(lista_t);

			/* Stampa a video della lunghezza della lista. */
			printf("\n\tNumero di nodi: %d.\n", lista_t->conto);
		}
```

```
                else if(lista_t->temp->precedente != NULL &&
                        lista_t->temp->successivo != NULL && lista_t->temp->info == valore)
                {
                        (lista_t->temp->precedente)->successivo = lista_t->temp->successivo;
                        (lista_t->temp->successivo)->precedente = lista_t->temp->precedente;
                        free(lista_t->temp);

                        /* Aggiorno la lunghezza della lista. */
                        lista_t->conto -=1;
                        printf("\nInforazione eliminata.");

                        /* Stampa a video della lunghezza della lista. */
                        printf("\n\tNumero di nodi: %d.\n", lista_t->conto);
                }
        }
    }

    /* Con questo ciclo è possibile stampare a video l'esito della funzione. */
    /* Se la lunghezza della lista è minore della lunghezza in principio. */
    if(i > lista_t->conto)
    {
        printf("\nInformazione '%d' cancellata dalla lista.\n", valore);
    }

    /* Se la lunghezza non è cambiata, l'informazione non era presente in lista. */
    if(i == lista_t->conto)
    {
        printf("\nInformazione '%d' non presente in lista.\n", valore);
    }

    return(lista_t);
}

/* La funzione "crea()" viene chiamata per creare un nuovo elemento della lista.
 * L'utente digita un intero che sarà associato all'informazione del nuovo nodo.
 * Verrà allocato dello spazio in memoria della dimensione della struttura nodo.
 * La funzione inoltre inizializza i puntatori a NULL,
 * questi verranno poi rielaborati in base alla posizione del nodo. */
lista *crea(lista *lista_t)
{
    /* Dichiarazione delle variabili. */
    int     valore;         /* Variabile che conterrà l'intero da inserire nell'elemento. */
    int     risultato;      /* Variabile utile al controllo dei caratteri inseriti. */

    /* Alloco lo spazio per il nuovo elemento di tipo "nodo". */
    lista_t->temp = (struct nodo_t *)malloc(sizeof(struct nodo_t));

    do
    {
```

```
		printf("\nInserisci informazione al nodo (intero) :");

		/* Controllo dei caratteri inseriti. */
		risultato = scanf("%d", &valore);

		if(risultato != 1)
		{
			printf("\nPer favore, inserire un intero:\t");
			svuota_buffer();
		}
	}	while(risultato != 1);

	{
	/* Viene riempito il campo "info" dell'elemento con la nuova informazione. */
		lista_t->temp->info = valore;

		/* Aggiorna la lunghezza della lista. */
		lista_t->conto +=1;
	}
	return(lista_t);
}

/* La funzione "cancella_testa()"si occupa di ricollegare i puntatori della lista
 * e di cancellare l'informazione in testa alla lista liberando lo spazio di memoria
 * da essa occupato. */
lista *cancella_testa(lista *lista_t)
{
	/* Puntatore temp punta alla testa della lista. */
	lista_t->temp = lista_t->testa;

	/* Verifica che la lista non sia vuota. */
	if(lista_t->temp==NULL)
	{
		printf("\nLa lista è vuota.");
	}

	else
	{
		/* Verifico che la testa non sia anche la coda della lista. */
		if(lista_t->testa->successivo != NULL)
		{
			lista_t->testa = lista_t->temp->successivo;
			lista_t->temp->successivo->precedente = NULL;
			lista_t->testa = lista_t->temp->successivo;
			lista_t->conto -=1;
		}
```

```
		else
		{
			lista_t->testa = NULL;
			lista_t->conto-=1;
		}
	}

	/* Libero la memoria occupata da temp. */
	free(lista_t->temp);

	/* Stampa a video della avvenuta cancellazione. */
	printf("\nInformazione in testa cancellata dalla lista.");

	return(lista_t);
}

/* La funzione "cancella_coda()" si occupa di ricollegare i puntatori della lista
 * e di cancellare l'informazione in coda alla lista. */
lista * cancella_coda(lista *lista_t)
{

	/* Puntatore temp punta alla coda della lista. */
	lista_t->temp = lista_t->coda;

	/* Verifica che la lista non sia vuota. */
	if(lista_t->temp == NULL)
	{
		printf("\nLa lista è vuota.");
	}

	else
	{
		/* Verifica che l'elemento non sia anche la testa della lista. */
		if(lista_t->coda->precedente != NULL)
		{
			lista_t->coda =  lista_t->temp -> precedente;
			(lista_t->temp->precedente)->successivo = NULL;

			/* Aggiorno la lunghezza della lista. */
			lista_t->conto-=1;
		}

		else
		{
			lista_t->coda = lista_t -> temp -> precedente = NULL;
			lista_t->temp->successivo = NULL;
			lista_t->testa = lista_t->coda;
```

```
                    /* Aggiorno la lunghezza della lista. */
                    lista_t->conto-=1;
                }
        }

        /* Libero la memoria occupata da temp. */
        free(lista_t->temp);

        /* Stampa a video della avvenuta cancellazione. */
        printf("\nInformazione in coda cancellata dalla lista.");

        return(lista_t);
}
```

4.2 Implementazione del file di test “.c”

```
/* File sorgente "test.c". */

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

/* Inclusione della libreria per la gestione di una lista
 * doppiamente concatenata. */
#include "lista_dc.h"

/* Dichiarazione delle funzioni. */
void    svuota_buffer();
lista   *inizializza(lista *);

/* Funzione principale "main()". */
int main()
{
        /* Dichiarazione delle variabili. */
        /* Variabile per la selezione della funzione. */
        int     scelta;
        /* Variabile per la validazione stretta degli input. */
        int     esito;

        lista *elemento;
        elemento = (struct lista_t *)malloc(sizeof(struct lista_t));

        elemento -> temp = NULL;
        elemento -> testa = NULL;
        elemento -> aus = NULL;
        elemento -> coda = NULL;
        elemento -> conto = 0;

        elemento = inizializza(elemento);

        do
        {

                /* Si procede alla stampa del menù di selezione. */
                printf("\n\n------MENÙ DI SELEZIONE------\n");
                printf(" 0) Esci.");
                printf("\n\t**INSERIMENTO**");
                printf("\n 1) Inserisci in testa.");
                printf("\n 2) Inserisci in coda.");
                printf("\n 3) Inserisci prima di posizione 'i'.");
                printf("\n 4) Inserisci dopo di posizione 'i'.");
                printf("\n\t**RIMOZIONE**");
                printf("\n 5) Cancella informazione.");
                printf("\n 6) Cancella informazione in testa.");
                printf("\n 7) Cancella informazione in coda.");
                printf("\n\t**VISITA**");
                printf("\n 8) Stampa da posizione a coda.");
                printf("\n 9) Stampa da posizione a testa.");

                printf("\n\nScelta:\t");
```

```
do
{
	esito = scanf("%d", &scelta);

	if(esito != 1 || scelta > 9 || scelta < 0)
	{
		printf("\nOperazione non consentita, ripetere la scelta.\n");
		printf("Inserire un intero compreso tra 0 e 9 (inclusi).\n");

		svuota_buffer();
	}

	switch (scelta)
	{
		case 1:
			printf("\t- Inserimento in testa");
		elemento = inserisci_testa(elemento);
		break;

		case 2:
			printf("\t- Inserimento in coda");
		elemento = inserisci_coda(elemento);
		break;

		case 3:
			printf("\t- Inserimento prima di posizione");
		elemento = inserisci_prima(elemento);
		break;

		case 4:
			printf("\t- Inserimento dopo di posizione");
			elemento = inserisci_dopo(elemento);
			break;

		case 5:
			printf("\t- Cancella informazione");
		elemento = cancella_info(elemento);
		break;

		case 6:
			printf("\t- Cancella testa");
			elemento = cancella_testa(elemento);
		break;

		case 7:
			printf("\t- Cancella coda");
			elemento = cancella_coda(elemento);
		break;

		case 8:
			printf("\t- Stampa da posizione a coda");
			stampa_avanti(elemento);
		break;

		case 9:
```

```
                                        printf("\t- Stampa da posizione a testa");
                                        stampa_indietro(elemento);
                                break;

                                case 0:
                                        printf("\t- Chiusura");
                                        break;

                                default:
                                        printf("\nRipetere la scelta:\t");
                                        break;

                        }
                }       while (esito != 1 || scelta > 9 || scelta < 0);
        }       while (scelta != 0);

        printf("\nFine del programma di test.\n");
        return 0;
}

void    svuota_buffer()
{
        /* Variabile utile allo svuotamento del buffer. */
        char    carattere;

        do
        {
                carattere = getchar();
        }       while (carattere != '\n');
}

lista * inizializza(lista *lista_t)
{
        lista_t->temp = (struct nodo_t *)malloc(sizeof(struct nodo_t));
        lista_t-> temp -> precedente =  NULL;
        lista_t-> temp -> successivo = NULL;
        lista_t -> testa = lista_t -> coda = lista_t -> aus; /*= lista_t -> temp;*/

        return(lista_t);
}
```

4.3 Implementazione dell'header file ".h"

```
/* Header file "lista_dc.h". */

/* Ridefinizione delle costanti simboliche esportate */

typedef struct   nodo_t

{

        int                 info;           /* Campo contenente l'informazione. */

        struct   nodo_t *successivo;    /* Puntatore all'elemento successivo. */

        struct   nodo_t *precedente;    /* Puntatore all'elemento precedente. */

}       nodo;


typedefstruct   lista_t

{

        int                 conto;

        struct   nodo_t *testa;

        struct   nodo_t *coda;

        struct   nodo_t *aus;

        struct   nodo_t *temp;

}       lista;


/* Ridichiarazione delle funzioni esportate (precedute da extern) */

extern  lista     *inserisci_testa(lista *lista_t);

extern  lista     *inserisci_coda(lista *lista_t);

extern  lista     *inserisci_prima(lista *lista_t);

extern  lista     *inserisci_dopo(lista *lista_t);

extern  lista     *stampa_avanti(lista *lista_t);

extern  lista     *stampa_indietro(lista *lista_t);

extern  lista     *cancella_info(lista *lista_t);

extern  lista     *cancella_testa(lista *lista_t);

extern  lista     *cancella_coda(lista *lista_t);
```

4.4 Makefile

```
test: Makefile lista_dc.o test.o lista_dc.h
	gcc	-ansi	-Wall	-O	lista_dc.o test.o -o test -lm
lista_dc.o:	Makefile lista_dc.c lista_dc.h
	gcc	-ansi	-Wall	-c lista_dc.c -lm
test.o:	Makefile lista_dc.o test.c lista_dc.h
	gcc	-ansi	-Wall	-c	test.c	-lm
pulisci:
	rm -f *.o
pulisci_tutto:
	rm -f test *.o
```

Capitolo 5

5.1 Test del programma

Di seguito vengono riportati alcuni screenshot effettuati durante vari test del programma in ambiente Linux.

Viene lanciato da terminale il file eseguibile con il comando “./test”.

```
angelo@envy:~/Scrivania/ppl$ ./test

------MENÙ DI SELEZIONE------
 0) Esci.
        **INSERIMENTO**
 1) Inserisci in testa.
 2) Inserisci in coda.
 3) Inserisci prima di posizione 'i'.
 4) Inserisci dopo di posizione 'i'.
        **RIMOZIONE**
 5) Cancella informazione.
 6) Cancella informazione in testa.
 7) Cancella informazione in coda.
        **VISITA**
 8) Stampa da posizione a coda.
 9) Stampa da posizione a testa.

Scelta:
```

Se si digita “8” quando non sono mai stati aggiunti elementi alla lista, il programma stampa a video un messaggio informativo.

```
------MENÙ DI SELEZIONE------
 0) Esci.
        **INSERIMENTO**
 1) Inserisci in testa.
 2) Inserisci in coda.
 3) Inserisci prima di posizione 'i'.
 4) Inserisci dopo di posizione 'i'.
        **RIMOZIONE**
 5) Cancella informazione.
 6) Cancella informazione in testa.
 7) Cancella informazione in coda.
        **VISITA**
 8) Stampa da posizione a coda.
 9) Stampa da posizione a testa.

Scelta: 8
        - Stampa da posizione a coda
Inserire la posizione dal quale iniziare la visita.1

La lista è vuota.

Scelta:
```

Digitando “1” è possibile inserire un nuovo elemento come testa della lista. Inserito il valore intero, è possibile stampare a video la lista fino ad ora creata digitando “8” dal menù di selezione e successivamente “1” per stamparla dal primo elemento.

```
angelo@envy:~/Scrivania/ppl$ ./test

------MENÙ DI SELEZIONE------
 0) Esci.
        **INSERIMENTO**
 1) Inserisci in testa.
 2) Inserisci in coda.
 3) Inserisci prima di posizione 'i'.
 4) Inserisci dopo di posizione 'i'.
        **RIMOZIONE**
 5) Cancella informazione.
 6) Cancella informazione in testa.
 7) Cancella informazione in coda.
        **VISITA**
 8) Stampa da posizione a coda.
 9) Stampa da posizione a testa.

Scelta: 1
        - Inserimento in testa
Inserisci informazione al nodo (intero) :111


Scelta: 8
        - Stampa da posizione a coda
Inserire la posizione dal quale iniziare la visita.1

Lista dalla posizione '1' in avanti :
 [ (Informazione: 111) ] Coda
                Lunghezza della lista: 1.


Scelta:
```

Successivamente, digitando “2” si seleziona la funzione di inserimento di un elemento in coda alla lista, inserito un intero, tramite la funzione di stampa in avanti è possibile visualizzare gli elementi della lista aggiunti. Viene stampato anche il numero di nodi della lista.

```
Scelta: 1
        - Inserimento in testa
Inserisci informazione al nodo (intero) :111


Scelta: 8
        - Stampa da posizione a coda
Inserire la posizione dal quale iniziare la visita.1

Lista dalla posizione '1' in avanti :
 [ (Informazione: 111) ] Coda
                Lunghezza della lista: 1.


Scelta: 2
        - Inserimento in coda
Inserisci informazione al nodo (intero) :222


Scelta: 8
        - Stampa da posizione a coda
Inserire la posizione dal quale iniziare la visita.1

Lista dalla posizione '1' in avanti :
 [ (Informazione: 111)  (Informazione: 222) ] Coda
                Lunghezza della lista: 2.


Scelta: 2
        - Inserimento in coda
Inserisci informazione al nodo (intero) :333


Scelta: 8
        - Stampa da posizione a coda
Inserire la posizione dal quale iniziare la visita.1

Lista dalla posizione '1' in avanti :
 [ (Informazione: 111)  (Informazione: 222)  (Informazione: 333) ] Coda
                Lunghezza della lista: 3.


Scelta:
```

Digitando “9”, viene chiamata la funzione di stampa della lista a partire dalla posizione scelta, “3” in questo screenshot, alla testa della lista. Viene inoltre stampato il numero di nodi della lista.

```
Scelta: 8
        - Stampa da posizione a coda
Inserire la posizione dal quale iniziare la visita.1

Lista dalla posizione '1' in avanti :
 [ (Informazione: 111)  (Informazione: 222)  (Informazione: 333) ] Coda
                Lunghezza della lista: 3.


Scelta: 9
        - Stampa da posizione a testa
Inserire la posizione dal quale iniziare la visita:     3

Lista dalla posizione '3' indietro : [ (Informazione: 333)  (Informazione: 222)  (Informazione: 111) ] Testa
                Lunghezza della lista: 3.


Scelta:
```

In questo screenshot l’utente digita “8” seguito da “2”. Il programma si comporta di conseguenza stampando la lista dalla seconda posizione alla coda.
Successivamente l’utente digita “3” , nuovamente “3” e inserisce l’informazione “999” che, in seguito alla scelta “8” con posizione “1”, stampa la lista dal primo elemento alla coda e appare l’informazione “999” prima della posizione “3”.

```
Scelta: 9
        - Stampa da posizione a testa
Inserire la posizione dal quale iniziare la visita:     3

Lista dalla posizione '3' indietro : [ (Informazione: 333)  (Informazione: 222)  (Informazione: 111)
                Lunghezza della lista: 3.


Scelta: 8
        - Stampa da posizione a coda
Inserire la posizione dal quale iniziare la visita.2

Lista dalla posizione '2' in avanti :
 [ (Informazione: 222)  (Informazione: 333) ] Coda
                Lunghezza della lista: 3.


Scelta: 3
        - Inserimento prima di posizione
Prima di quale posizione inserire il nuovo nodo?
(Nodo 1: Testa; Nodo 3: Coda;)  3

Inserisci informazione al nodo (intero) :999


Scelta: 8
        - Stampa da posizione a coda
Inserire la posizione dal quale iniziare la visita.1

Lista dalla posizione '1' in avanti :
 [ (Informazione: 111)  (Informazione: 222)  (Informazione: 999)  (Informazione: 333) ] Coda
                Lunghezza della lista: 4.


Scelta: 4
        - Inserimento dopo di posizione
Dopo di quale posizione inserire il nuovo nodo?
(Nodo 1: Testa; Nodo 4: Coda;)  2

Inserisci informazione al nodo (intero) :999


Scelta:
```

Viene allora chiamata la funzione di inserimento dopo di una posizione digitando “4” e successivamente digitando “2”. Il nuovo elemento verrà inserito dopo il secondo elemento.

```
Scelta: 9
        - Stampa da posizione a testa
Inserire la posizione dal quale iniziare la visita:     3

Lista dalla posizione '3' indietro : [ (Informazione: 333)  (Informazione: 222)  (Informazione: 111)
                Lunghezza della lista: 3.


Scelta: 8
        - Stampa da posizione a coda
Inserire la posizione dal quale iniziare la visita.2

Lista dalla posizione '2' in avanti :
 [ (Informazione: 222)  (Informazione: 333) ] Coda
                Lunghezza della lista: 3.


Scelta: 3
        - Inserimento prima di posizione
Prima di quale posizione inserire il nuovo nodo?
(Nodo 1: Testa; Nodo 3: Coda;)  3

Inserisci informazione al nodo (intero) :999


Scelta: 8
        - Stampa da posizione a coda
Inserire la posizione dal quale iniziare la visita.1

Lista dalla posizione '1' in avanti :
 [ (Informazione: 111)  (Informazione: 222)  (Informazione: 999)  (Informazione: 333) ] Coda
                Lunghezza della lista: 4.


Scelta: 4
        - Inserimento dopo di posizione
Dopo di quale posizione inserire il nuovo nodo?
(Nodo 1: Testa; Nodo 4: Coda;)  2

Inserisci informazione al nodo (intero) :999


Scelta:
```

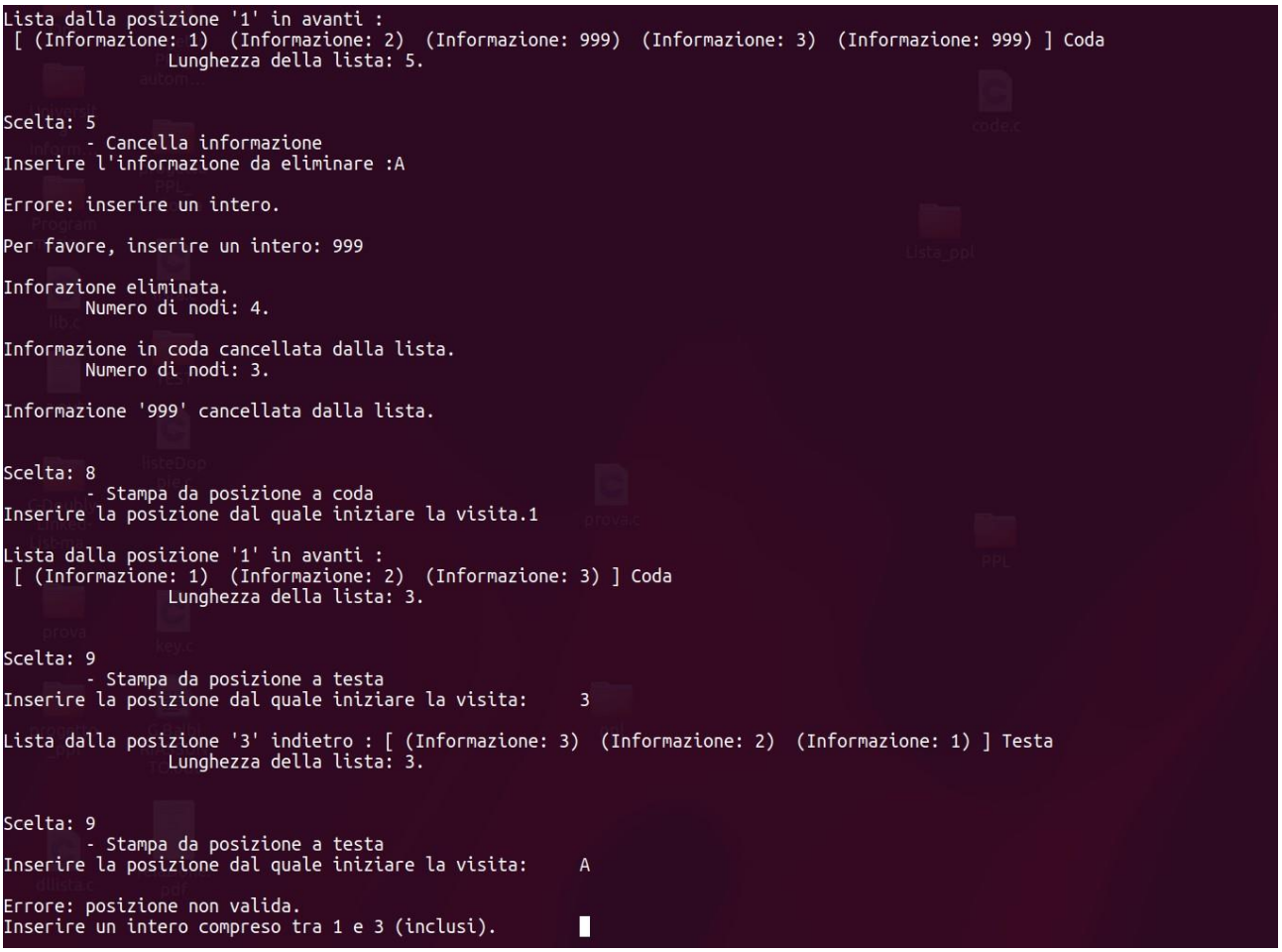
Un esempio di errore di inserimento: viene inserito un carattere quando l'informazione deve essere un intero. Il programma stampa a video l'errore e chiede nuovamente di inserire un intero.

```
angelo@envy:~/Scrivania/ppl$ ./test

------MENÙ DI SELEZIONE------
 0) Esci.
        **INSERIMENTO**
 1) Inserisci in testa.
 2) Inserisci in coda.
 3) Inserisci prima di posizione 'i'.
 4) Inserisci dopo di posizione 'i'.
        **RIMOZIONE**
 5) Cancella informazione.
 6) Cancella informazione in testa.
 7) Cancella informazione in coda.
        **VISITA**
 8) Stampa da posizione a coda.
 9) Stampa da posizione a testa.

Scelta: 1
        - Inserimento in testa
Inserisci informazione al nodo (intero) :a

Errore: l'informazione inserita non è un valore intero.

Per favore, inserire un intero:
```

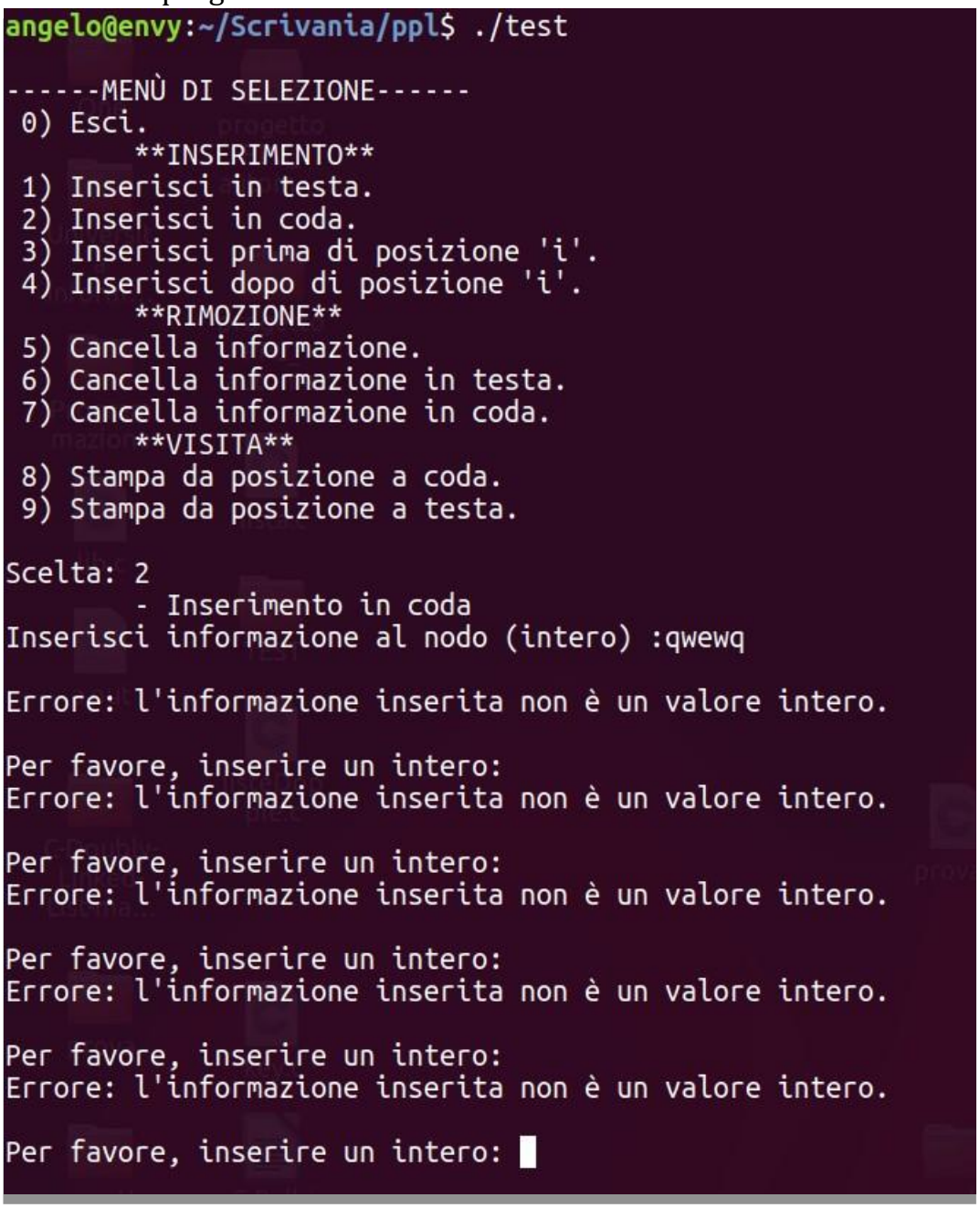
Nelle due funzioni di stampa della lista, se l'utente inserisce una posizione fuori dalla grandezza della lista, questo viene stampato a video.
Nell'esempio si ha una lista composta da quattro elementi e l'utente dopo aver digitato "8", inserisce "9" che è chiaramente fuori dalla portata della lista. Questo viene stampato a video e l'utente dovrà inserire nuovamente una scelta valida dal menù di selezione.

```
Scelta: 8
        - Stampa da posizione a coda
Inserire la posizione dal quale iniziare la visita.1

Lista dalla posizione '1' in avanti :
 [ (Informazione: 1)  (Informazione: 2)  (Informazione: 3)  (Informazione: 4) ] Coda
                Lunghezza della lista: 4.


Scelta: 8
        - Stampa da posizione a coda
Inserire la posizione dal quale iniziare la visita.9

La posizione è fuori portata.

Scelta:
```

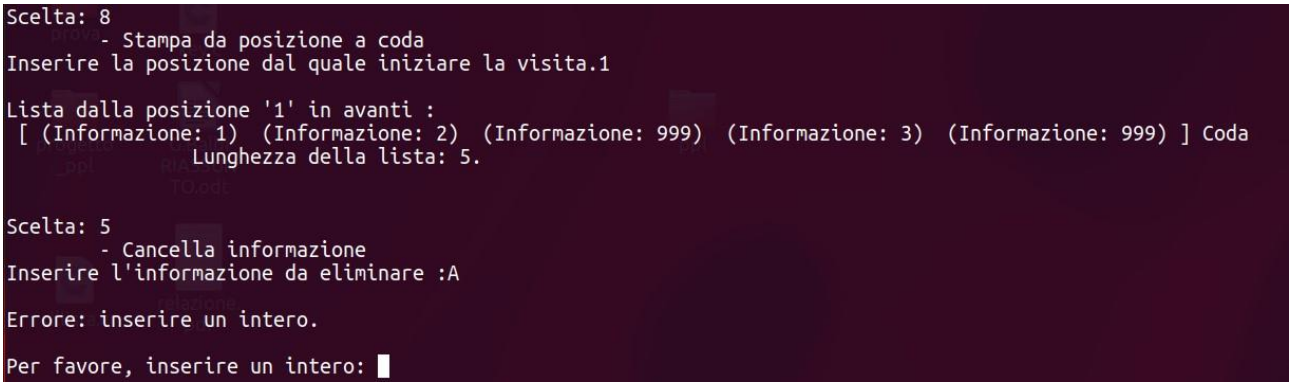
Questo screenshot ha inizio con la stampa della lista dal primo elemento alla coda. Successivamente l'utente decide di eleminare un'informazione digitando "A", questo è un carattere, la funzione effettua il controllo e stampa a video una nuova richiesta di inserimento.
L'utente inserisce l'informazione "999" che è presente due volte in lista.
Vengono stampate due conferme di avvenuta eliminazione e due volte la lunghezza della lista per aggiornare l'utente di quante informazioni uguali sono state cancellate. Dopo un'ulteriore stampa a video della lista conclusa, l'utente desidera di chiamare nuovamente la funzione ma digitando questa volta un carattere, che non è ammesso dai controlli. Viene stampato a video un messaggio di errore ed uno informativo, l'utente dovrà inserire un intero.



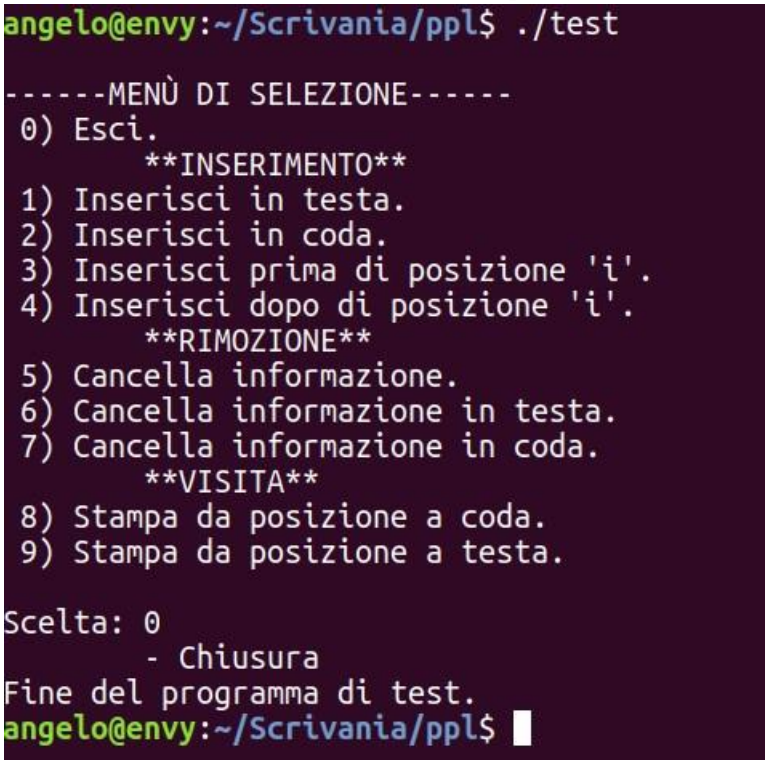
Il seguente screenshot rappresenta un errore di inserimento.
La funzione chiamata è quella per inserire un’informazione in coda alla lista ma l’input dell’utente è una stringa. In questo caso l’errore viene stampato per ogni carattere inserito e il programma chiede nuovamente di inserire un intero.



Lo screenshot seguente raffigura la stampa a video della lista, chiamata digitando “8”, dalla posizione “1” fino alla coda della lista. Viene stampato il numero di nodi della lista. Successivamente l’utente digita “5” per rimuovere un’informazione che però è un carattere.
Viene stampato a video un messaggio di errore e il programma chiede nuovamente di inserire un valore intero.



Quest’ultimo screenshot mostra il comportamento del programma quando l’utente digita “0” nel menù di selezione. Il programma si chiude correttamente.



Capitolo 6

Verifica della correttezza del programma

Per la verifica della correttezza del programma prendiamo in considerazione la seguente parte di codice dal file sorgente “lista_dc.c” e adibita alla verifica dell’esistenza della lista e alla rimozione dell’elemento di testa:

```
/* Verifica che la lista non sia vuota. */
        if(lista_t->temp==NULL)
        {
                printf("\nLa lista è vuota.");
        }

        else
        {
                /* Verifico che la testa non sia anche la coda della lista. */
                if(lista_t->testa->successivo != NULL)
                {
                        lista_t->testa = lista_t->temp->successivo;
                        lista_t->temp->successivo->precedente = NULL;
                        lista_t->testa = lista_t->temp->successivo;
                        lista_t->conto -=1;
                }

                else
                {
                        lista_t->testa = NULL;
                        lista_t->conto-=1;
                }
        }
```

La semantica assiomatica dei linguaggi di programmazione si basa sul concetto di formula di correttezza o tripla di Hoare.
Identifichiamo ora la pre-condizione *Q,* la post-condizione *R e* il comando di cui voler verificare la correttezza, ovvero *S.*
S: identifichiamo il comando di cui vogliamo verificare la correttezza come l'insieme dei due comandi *if* ed *else.*
R: identifichiamo la post-condizione come il ciclo if-else annidato oltre al decremento della variabile conto.
Q: La pre-condizione va dunque riconosciuta negli argomenti del comando *"if".*

Se la lista non è vuota è possibile accedere al ciclo annidato.
Se la lista è vuota, l'algoritmo si conclude correttamente con la stampa a video.

Il comando *S* è corretto rispetto alla pre-condizione *Q* poiché termina correttamente con una stampa a video nel caso in cui la lista è vuota e termina correttamente rispetto la post-condizione *R* poiché nel caso in cui la lista non sia vuota il programma scorre gli elementi e termina col decremento dell'elemento conto.

Pre-condizione:
((lista->testa->successivo != NULL)) → decremento ≡ vero
((lista->testa->successivo == NULL)) → decremento ≡ vero
(vero ⋀ vero) ≡ vero

Post-condizione:

(lista->testa = NULL) → decremento vero

(vero ⋀ vero) ≡ vero

Strumenti utilizzati

Per la realizzazione della relazione sono stati utilizzati i seguenti software:
Microsoft Word, Adobe PhotoShop, Libre Office.
Per la realizzazione del progetto sono stati utilizzati i seguenti software opensource*: gcc* e *gVim.*
Il progetto è stato sviluppato sia in ambiente *Windows* (Windows 10) che in ambiente *Linux* (Ubuntu 16.04)*.*