

# Introduzione alla programmazione in C

Appunti delle lezioni di  
Tecniche di programmazione

*Giorgio Grisetti*

*Luca Iocchi*

*Daniele Nardi*

*Fabio Patrizi*

*Alberto Pretto*

*Dipartimento di Ingegneria Informatica, Automatica e Gestionale*

*Facoltà di Ingegneria dell'Informazione, Informatica, Statistica*

*Università di Roma "La Sapienza"*

*Edizione 2019/2020*



2020



# Indice

9. Strutture collegate lineari	239
9.1. Limitazioni degli array	239
9.2. Strutture collegate	239
9.2.1. Dichiarazione di una Struttura Collegata Lineare	240
9.2.2. Operazioni sulle strutture collegate lineari	240
9.2.3. Creazione e collegamento di nodi	241
9.3. Operazioni sul nodo in prima posizione	241
9.3.1. Inserimento di un nodo in prima posizione	242
9.3.2. Eliminazione di un nodo in prima posizione	242
9.4. Ricorsione per le operazioni su SCL	243
9.4.1. Schema di ricorsione per SCL	243
9.4.2. Verifica SCL vuota	243
9.5. Operazioni che non modificano la SCL	244
9.5.1. Scrittura di una SCL	244
9.5.2. Verifica presenza di un elemento	245
9.5.3. Ricerca	246
9.5.4. Lunghezza	246
9.6. Operazioni che modificano il contenuto della SCL	246
9.6.1. Modifica dell'informazione in un nodo	247
9.6.2. Sostituzione di un elemento	247
9.6.3. Sostituzione di occorrenze multiple	247
9.7. Operazioni che modificano la SCL	248
9.7.1. Costruzione di una SCL	248
9.7.2. Lettura di una SCL	248

9.7.3. Copia	250
9.7.4. Eliminazione	250
9.8. Operazioni basate sulla posizione	250
9.8.1. Ricerca di un elemento tramite posizione	251
9.8.2. Inserimento in posizione data	251
9.8.3. Eliminazione di un elemento in posizione data	252
9.9. Operazioni iterative su SCL	252
9.9.1. Operazioni che non modificano la SCL	252
9.9.2. Operazioni che modificano il contenuto della SCL	253
9.9.3. Operazioni che modificano la struttura della SCL	254
9.10. SCL: Implementazione funzionale	259
9.10.1. Funzioni primitive	259
9.10.2. Primo	260
9.10.3. Resto	260
9.10.4. Costruzione di una SCL	260
9.10.5. Copia di una SCL	261
9.10.6. Inserimento in posizione n	261
9.10.7. Eliminazione in posizione n	261
9.10.8. Modifica in posizione n	262

## 9. Strutture collegate lineari

### 9.1. Limitazioni degli array

L'uso di array per memorizzare collezioni di oggetti presenta alcune limitazioni nella gestione della memoria:

- la dimensione dell'array è stabilita al momento della sua creazione e può essere inefficiente modificarla successivamente;
- l'array utilizza uno spazio di memoria proporzionale alla sua dimensione, indipendentemente dal numero di elementi validi effettivamente memorizzati;
- per mantenere un ordinamento degli oggetti della collezione e inserire (o rimuovere) un valore in una posizione specifica dobbiamo fare uno spostamento per una buona parte degli elementi dell'array.

Le strutture collegate che introduciamo in questa unità sono definite quindi appositamente per consentire di allocare e deallocare memoria in maniera dinamica, a seconda delle esigenze del programma nella memorizzazione dei dati di interesse per l'applicazione.

### 9.2. Strutture collegate

Un meccanismo molto flessibile per la gestione dinamica della memoria è dato dalle *strutture collegate*, che vengono realizzate in maniera tale da consentire facilmente la modifica della propria struttura, gli inserimenti e le cancellazioni in posizioni specifiche, ecc.

In questa unità vedremo le *strutture collegate lineari* (SCL), che consentono quindi di memorizzare collezioni di oggetti organizzate sotto forma di sequenze (cioè in cui ogni elemento ha un solo successore).

Successivamente vedremo invece gli alberi, che sono un esempio di strutture collegate non lineari.

### 9.2.1. Dichiarazione di una Struttura Collegata Lineare

Una struttura collegata lineare (SCL) è definita tramite record.

```
typedef ... TipoInfoSCL;

struct ElemSCL {
    TipoInfoSCL info;
    struct ElemSCL *next;
};

typedef struct ElemSCL TipoNodoSCL;
typedef TipoNodoSCL * TipoSCL;
```

[TipoInfoSCL](#) è il tipo dei dati contenuti nella SCL.

[TipoSCL](#) è il tipo puntatore alla struttura che definisce i nodi (record di tipo [TipoNodoSCL](#)) della SCL.

### 9.2.2. Operazioni sulle strutture collegate lineari

Le operazioni più comuni sulle strutture collegate lineari sono: operazioni che non modificano la SCL

- *scrittura* (su file)
- *ricerca* di un'informazione nella struttura
- calcolo della *dimensione* della struttura

operazioni che modificano il contenuto (ma non la struttura) della SCL

- *modifica* di elementi

operazioni che modificano la struttura della SCL

- *creazione*
- *inserimento ed eliminazione*

- *lettura* (da file)
- creazione tramite *copia*
- *distruzione*

### 9.2.3. Creazione e collegamento di nodi

Le operazioni di base sulle SCL sono: l'inizializzazione di una SCL vuota, la creazione di un nodo e il collegamento tra due nodi.

SCL vuota:

```
TipoSCL scl = NULL;
```

Creazione di un nodo:

```
TipoSCL scl = (TipoNodoSCL*) malloc(sizeof(TipoNodoSCL));
scl->info = e1;
scl->next = NULL;
```

Collegamento di nodi:

```
TipoSCL temp = (TipoNodoSCL*) malloc(sizeof(TipoNodoSCL));
temp->info = e2;
temp->next = NULL;
scl->next = temp;
```

## 9.3. Operazioni sul nodo in prima posizione

Le funzioni base per la gestione delle SCL sono inserimento ed eliminazione del nodo che si trova in prima posizione.

### 9.3.1. Inserimento di un nodo in prima posizione

```
void addSCL(TipoSCL *scl, TipoInfoSCL e) {
    TipoSCL temp = *scl;
    *scl = (TipoNodoSCL*) malloc(sizeof(TipoNodoSCL));
    (*scl)->info = e;
    (*scl)->next = temp;
}
```

Si noti che la funzione `addSCL` opera correttamente anche se la struttura collegata è inizialmente vuota, nel qual caso viene costruita una SCL con un solo elemento.

*Esempio:* Il seguente frammento di codice genera una SCL contenente i valori (1, 2, 3).

```
TipoSCL scl = NULL;
addSCL(&scl, 3);
addSCL(&scl, 2);
addSCL(&scl, 1);
```

### 9.3.2. Eliminazione di un nodo in prima posizione

```
void delSCL(TipoSCL * scl) {
    TipoSCL temp = *scl;
    *scl = (*scl)->next;
    free(temp);
}
```

Si noti che la funzione `delSCL` si comporta correttamente anche se la struttura collegata è costituita da un solo elemento, nel qual caso viene restituita una SCL vuota.

*Esempio:* Il seguente frammento di codice dealloca una SCL contenente tre elementi.

```
...
delSCL(&scl);
delSCL(&scl);
delSCL(&scl);
```



## 9.4. Ricorsione per le operazioni su SCL

L'implementazione ricorsiva delle operazioni sulle strutture collegate lineari risulta generalmente più semplice di quella iterativa. Ciò deriva dal modo in cui sono definite le strutture collegate. Una struttura collegata è:

- vuota;
- formata un elemento seguito da una struttura collegata.

### 9.4.1. Schema di ricorsione per SCL

Schema ricorsivo di lettura

```
if (SCL vuota) {  
    passo base  
}  
else {  
    elaborazione primo elemento della SCL  
    chiamata ricorsiva sul resto della SCL  
}
```

Schema ricorsivo di scrittura

```
leggi dato  
if (dato valido) {  
    inserimento del dato nella SCL  
    chiamata ricorsiva sui dati rimanenti  
}
```

### 9.4.2. Verifica SCL vuota

```
// restituisce true se scl e' vuota  
int emptySCL(TipoSCL scl) {  
    return scl == NULL;  
}
```

## 9.5. Operazioni che non modificano la SCL

Le operazioni che non modificano la SCL sono definite come funzioni in cui il parametro usato per indicare la SCL è di tipo `TipoSCL`.

Ad esempio, la funzione di scrittura definita con `void writeSCL(TipoSCL scl)` viene invocata come segue:

```
TipoSCL scl;  
...  
writeSCL(scl);
```

Si noti che questo passaggio di parametri non garantisce comunque che la struttura o il contenuto della SCL rimangano invariati.

### 9.5.1. Scrittura di una SCL

Scrittura dell'informazione del nodo:

```
// scrittura dell'informazione nel nodo (esempio int)  
void writeTipoInfo(TipoInfoSCL d) {  
    printf("%d ",d);  
}  
// scrittura tramite file  
void writeTipoInfoF(FILE *f, TipoInfoSCL d) {  
    fprintf(f,"%d ",d);  
}
```

Scrittura della SCL:

```
void writeSCL(TipoSCL scl) {  
    if (emptySCL(scl))  
        printf("\n");  
    else {  
        writeTipoInfo(scl->info);  
        writeSCL(scl->next);  
    }  
}
```

Scrittura su file:

```

void writeSCLF(char *nfile, TipoSCL scl) {
    FILE *datafile;
    datafile = fopen(nfile, "w");
    if (datafile == NULL) { // errore in apertura in
        scrittura
        printf("Errore apertura file '%s' in scrittura\n",
            nfile);
    }
    else {
        writeSCLF_r(datafile, scl);
        fclose(datafile);
    }
}

```

Funzione di scrittura ricorsiva:

```

void writeSCLF_r(FILE *o, TipoSCL scl) {
    if (!emptySCL(scl)) {
        writeTipoInfoF(o, scl->info);
        writeSCLF_r(o, scl->next);
    }
}

```

### 9.5.2. Verifica presenza di un elemento

```

// restituisce true se trova e in scl
int isinSCL(TipoSCL scl, TipoInfoSCL e) {
    if (emptySCL(scl))
        return 0;
    else if (scl->info==e)
        return 1;
    else
        return isinSCL(scl->next, e);
}

```

*Nota:* in generale, il confronto tra le informazioni contenute nei nodi viene effettuato con l'operatore `==` per i tipi di dati primitivi, ma occorre definire funzioni specifiche per i tipi strutturati definiti dall'utente.

### 9.5.3. Ricerca

```
// restituisce il puntatore al nodo che contiene e,
// se trova e in scl, altrimenti NULL
void findSCL(TipoSCL scl, TipoInfoSCL e, TipoSCL *ris)
{
    if (emptySCL(scl))
        *ris = NULL;
    else if (scl->info==e)
        *ris = scl;
    else {
        findSCL(scl->next, e, ris);
    }
    return;
}
```

### 9.5.4. Lunghezza

```
// restituisce la lunghezza della scl
int lengthSCL(TipoSCL scl) {
    if (emptySCL(scl))
        return 0;
    else
        return 1 + lengthSCL(scl->next);
}
```

## 9.6. Operazioni che modificano il contenuto della SCL

Le operazioni che modificano il contenuto della SCL ma non la sua struttura sono definite come funzioni in cui il parametro usato per indicare la SCL è di tipo `TipoSCL`.

Ad esempio, la funzione di scrittura definita con `void substElemSCL(TipoSCL scl, TipoInfoSCL e, TipoInfoSCL n)` viene invocata come segue:

```
TipoSCL scl;
...
substElemSCL(scl, e, n);
```

Si noti che questo passaggio di parametri non garantisce comunque che la struttura della SCL rimanga invariata.

### 9.6.1. Modifica dell'informazione in un nodo

Sia **p** (il riferimento a) un nodo qualsiasi della struttura collegata, vogliamo modificare la sua informazione con il valore della stringa **x**.

```
TipoNodoSCL * p = ... // p e' il riferimento al nodo
TipoInfoSCL x = ... // x e' il nuovo valore da
    memorizzare in p
p->info = x;
```

In questo caso non è necessario modificare la struttura collegata, ma solamente il contenuto dell'informazione del nodo in questione.

### 9.6.2. Sostituzione di un elemento

```
// sostituisce la prima occorrenza dell'elemento e di
// scl con n
void substElemSCL(TipoSCL scl, TipoInfoSCL e,
    TipoInfoSCL n) {
    if (!emptySCL(scl)) {
        if (scl->info==e)
            scl->info = n;
        else
            substElemSCL(scl->next, e, n);
    }
}
```

### 9.6.3. Sostituzione di occorrenze multiple

```
// sostituisce tutte le occorrenze dell'elemento e di
// scl con n
void substNElemSCL(TipoSCL scl, TipoInfoSCL e,
    TipoInfoSCL n) {
    if (!emptySCL(scl)) {
        if (scl->info==e)
            scl->info = n;
        substNElemSCL(scl->next, e, n);
    }
}
```

## 9.7. Operazioni che modificano la SCL

Le operazioni che modificano la struttura della SCL sono definite come funzioni in cui il parametro usato per indicare la SCL è di tipo `TipoSCL *`.

Ad esempio, la funzione di scrittura definita con `void readSCL(TipoSCL *scl)` viene invocata come segue:

```
TipoSCL scl;  
...  
readSCL(&scl);
```

### 9.7.1. Costruzione di una SCL

Costruzione di una SCL di `n` elementi inizializzati con il valore `e`:

```
void creaSCL(TipoSCL *scl, int n, TipoInfoSCL e) {  
    if (n == 0)  
        *scl = NULL;  
    else {  
        *scl = (TipoNodoSCL*) malloc(sizeof(TipoNodoSCL));  
        (*scl)->info = e;  
        creaSCL(&((*scl)->next), n-1, e);  
    }  
}
```

### 9.7.2. Lettura di una SCL

Lettura dell'informazione del nodo:

```
// lettura dell'informazione nel nodo (esempio int)  
int readTipoInfo(TipoInfoSCL *d) {  
    return scanf("%d", d);  
}  
  
// lettura tramite file  
int readTipoInfoF(FILE *f, TipoInfoSCL *d) {  
    return fscanf(f, "%d", d);  
}
```

Lettura della SCL:

```

void readSCL(TipoSCL *scl) {
    TipoInfoSCL dat;
    if (readTipoInfo(&dat)==EOF)
        *scl = NULL;
    else {
        *scl = (TipoNodoSCL*) malloc(sizeof(TipoNodoSCL));
        (*scl)->info = dat;
        readSCL(&((*scl)->next));
    }
}

```

Variante della lettura con [addSCL](#):

```

void readAddSCL(TipoSCL * scl) {
    TipoInfoSCL dat;
    if (readTipoInfo(&dat)==EOF)
        *scl = NULL;
    else {
        readAddSCL(scl);
        addSCL(scl,dat);
    }
}

```

Lettura da file:

```

void readSCLF(char *nfile, int *n, TipoSCL *scl) {
    FILE *datafile;
    datafile = fopen(nfile, "r");
    (*n) = 0;
    if (datafile == NULL) { // errore in apertura in
        lettura
        printf("Errore apertura file '%s' in lettura\n",
            nfile);
        *scl = NULL;
    }
    else
        readSCLF_r(datafile, n, scl);
    fclose(datafile);
}

```

Funzione di lettura ricorsiva:

```

void readSCLF_r(FILE *i, int *n, TipoSCL *scl) {
    TipoInfoSCL dat;
    if (readTipoInfoF(i, &dat)==EOF)
        *scl = NULL;
    else {
        (*n)++;
        *scl = (TipoNodoSCL*) malloc(sizeof(TipoNodoSCL));
        (*scl)->info = dat;
        readSCLF_r(i, n, &((*scl)->next));
    }
}

```

### 9.7.3. Copia

```

// copia scl in ris
void copySCL(TipoSCL scl, TipoSCL *ris) {
    if (emptySCL(scl))
        *ris=NULL;
    else {
        *ris = (TipoNodoSCL*) malloc(sizeof(TipoNodoSCL));
        (*ris)->info = scl->info;
        copySCL(scl->next, &((*ris)->next));
    }
}

```

### 9.7.4. Eliminazione

```

void eliminaSCL(TipoSCL *scl) {
    if (*scl != NULL) {
        TipoSCL p = *scl;
        eliminaSCL(&((*scl)->next));
        free(p);
    }
}

```

## 9.8. Operazioni basate sulla posizione

Le operazioni basate sulla posizione più comuni sono:

- ricerca dell'elemento in posizione n  
`void findPosSCL(TipoSCL scl, int n, TipoSCL * ris)`



- inserimento di un elemento in posizione n  
`void addPosSCL(TipoSCL * scl, TipoInfoSCL e, int n);`
- eliminazione di un elemento in posizione n  
`void delPosSCL(TipoSCL * scl, int n);`

Numeriamo gli elementi di una SCL a partire da 0.

### 9.8.1. Ricerca di un elemento tramite posizione

```
// restituisce il puntatore al nodo in posizione n
void findPosSCL(TipoSCL scl, int n, TipoSCL *ris) {
    if (n == 0)
        *ris = scl;
    else
        findPosSCL(scl->next, n-1, ris);
}
```

### 9.8.2. Inserimento in posizione data

```
// inserisce un nodo in posizione n
void addPosSCL(TipoSCL * scl, TipoInfoSCL e, int n){
    if (n == 0) {
        TipoSCL temp = *scl;
        *scl = (TipoNodoSCL*) malloc(sizeof(TipoNodoSCL));
        (*scl)->info = e;
        (*scl)->next = temp;
    }
    else
        addPosSCL(&((*scl)->next), e, n-1);
}
```

### 9.8.3. Eliminazione di un elemento in posizione data

```
// elimina un nodo in posizione n
void delPosSCL(TipoSCL *scl, int n) {
    if (scl == NULL) {
        return;
    }
    else if (n == 0) {
        TipoSCL temp = *scl;
        (*scl) = (*scl)->next;
        free(temp);
    }
    else
        delPosSCL(&((*scl)->next), n-1);
}
```

## 9.9. Operazioni iterative su SCL

Per effettuare operazioni su tutti gli elementi di una struttura collegata lineare, oppure su uno specifico elemento caratterizzato da una proprietà, è possibile anche effettuare cicli di scansione per accedere agli elementi.

Lo schema di ciclo per accedere a tutti i nodi della struttura il cui primo nodo è puntato dalla variabile `scl` è il seguente.

```
TipoSCL scl = ...
TipoSCL p = scl;
while (p!=NULL) {
    elaborazione del nodo puntato da p
    p = p->next;
}
```

### 9.9.1. Operazioni che non modificano la SCL

Scrittura

```
void writeSCL(TipoSCL scl) {
    while (!emptySCL(scl)) {
        writeTipoInfo(scl->info);
        scl=scl->next;
    }
    printf("\n");
}
```

Verifica presenza di un elemento

```
// restituisce true se trova e in scl
int isinSCL(TipoSCL scl, TipoInfoSCL e) {
    int trovato = 0;
    while (!emptySCL(scl) && !trovato) {
        if (scl->info==e)
            trovato = 1; /* forza l'uscita dal ciclo */
        else
            scl=scl->next; /* aggiorna scl al resto della
            lista */
    }
    return trovato;
}
```

Lunghezza

```
// restituisce la lunghezza della scl
int lengthSCL(TipoSCL scl) {
    int cont = 0;
    while (!emptySCL(scl)) {
        cont++;
        scl=scl->next;
    }
    return cont;
}
```

### 9.9.2. Operazioni che modificano il contenuto della SCL

Sostituzione di un elemento

```

// sostituisce la prima occorrenza dell'elemento e di
// scl con n
void substElemSCL(TipoSCL scl, TipoInfoSCL e,
    TipoInfoSCL n) {
    int trovato = 0;
    while (!emptySCL(scl) && !trovato) {
        if(scl->info==e) {
            scl->info = n;
            trovato = 1;
        }
        else
            scl = scl->next;
    }
}

```

### Sostituzione di un elemento

```

// sostituisce tutte le occorrenze dell'elemento e di
// scl con n
void substNElemSCL(TipoSCL scl, TipoInfoSCL e,
    TipoInfoSCL n) {
    while (!emptySCL(scl)) {
        if(scl->info==e)
            scl->info = n;
        scl = scl->next;
    }
}

```

## 9.9.3. Operazioni che modificano la struttura della SCL

### Costruzione

Si usa la tecnica del *nodo generatore*.

Il nodo generatore è un nodo ausiliario che viene anteposto alla lista che vogliamo costruire e da cui partono le operazioni di creazione dei nodi successivi. Al termine della creazione della lista tale nodo viene rimosso e viene restituita la lista a partire dal nodo successivo. Tale soluzione è usata per trattare uniformemente tutti gli elementi della lista, compreso il primo. Si osservi infatti che il metodo è corretto anche nel caso in cui la lista su cui viene invocato sia la lista vuota.

```
void creaSCL(TipoSCL *scl, int n, TipoInfoSCL e) {
    TipoSCL prec; // puntatore a elemento precedente
    // creazione del nodo generatore
    prec = (TipoNodoSCL*) malloc(sizeof(TipoNodoSCL));
    // scansione e copia della lista
    *scl = prec;
    while (n > 0) {
        // copia dell'elemento corrente
        prec->next = (TipoNodoSCL*) malloc(sizeof(
            TipoNodoSCL));
        prec = prec->next;
        prec->info = e;
        n--;
    }
    prec->next = NULL; // chiusura della scl
    // eliminazione del nodo generatore
    prec = *scl;
    *scl = (*scl)->next;
    free(prec);
}
```

```

void readSCLF(char *nfile, int * n, TipoSCL * scl) {
    FILE *datafile;
    datafile = fopen(nfile, "r");
    (*n) = 0;
    if (datafile == NULL) { // errore in apertura in
        lettura
        printf("Errore apertura file '%s' in lettura\n", nfile);
        *scl = NULL;
        return;
    }
    TipoInfoSCL elem;
    /* creazione del nodo generatore */
    *scl = (TipoNodoSCL*) malloc(sizeof(TipoNodoSCL));
    TipoSCL paux = *scl;
    /* ciclo lettura */
    while (readTipoInfoF(i, &elem) != EOF) {
        /* allocazione di un nodo e lettura del nuovo
        elemento */
        paux->next = (TipoNodoSCL*) malloc(sizeof(
        TipoNodoSCL));
        paux = paux->next;
        paux->info = elem;
        (*n)++;
    }
    paux->next = NULL; /* chiusura della scl */
    /* cancellazione del record generatore */
    paux = *scl;
    *scl = (*scl)->next;
    free(paux);
    fclose(datafile);
}

```

```

void copySCL(TipoSCL scl, TipoSCL *ris) {
    TipoSCL prec; // puntatore a elemento precedente
    // creazione del nodo generatore
    prec = (TipoNodoSCL*) malloc(sizeof(TipoNodoSCL));
    // scansione e copia della lista
    *ris = prec;
    while (!emptySCL(scl)) {
        // copia dell'elemento corrente
        prec->next = (TipoNodoSCL*) malloc(sizeof(
            TipoNodoSCL));
        prec = prec->next;
        prec->info = scl->info;
        scl = scl->next;
    }
    prec->next = NULL; // chiusura della scl
    // eliminazione del nodo generatore
    prec = *ris;
    *ris = (*ris)->next;
    free(prec);
}

```

Eliminazione primo nodo contenente elemento dato

```

void eliminaInfoSCL(TipoSCL *scl, TipoInfoSCL e) {
    /* si assume scl!=NULL e n>=0 */
    while (!emptySCL(*scl) && (*scl) -> info != e) {
        scl = &((*scl)-> next);
    }
    if (*scl != NULL){
        TipoSCL temp = *scl;
        *scl = temp -> next;
        free(temp);
    }
}

```

Eliminazione nodi contenenti elemento dato

```

void eliminaTuttiInfoSCL(TipoSCL *scl, TipoInfoSCL e)
{
    /* si assume scl!=NULL e n>=0 */
    TipoSCL temp;
    while (!emptySCL(*scl)) {
        if ((*scl) -> info == e){
            temp = *scl;
            *scl = temp -> next;
            free(temp);
        }
        scl = &((*scl)-> next);
    }
}

```

### Eliminazione in posizione data

```

void eliminaPosSCL(TipoSCL *scl, int n) {
    /* si assume scl!=NULL e n>=0 */
    while (!emptySCL(*scl) && n>0) {
        scl = &((*scl)-> next);
        n--;
    }
    TipoSCL temp = *scl;
    *scl = temp -> next;
    free(temp);
}

```

### Eliminazione

```

void eliminaSCL(TipoSCL *scl) {
    while (!emptySCL(*scl)) {
        TipoSCL p = *scl;
        *scl = (*scl)->next;
        free(p);
    }
}

```

### Inversa



```
void invertiSCL(TipoSCL *scl) {
    TipoSCL prec = NULL; /* elemento precedente */
    TipoSCL suc;         /* elemento successivo */
    while (!emptySCL(*scl)) {
        suc = *scl;
        *scl = (*scl)->next;
        suc->next = prec;
        prec = suc;
    }
    *scl = prec;
}
```

## 9.10. SCL: Implementazione funzionale

Nelle implementazioni funzionali, il risultato delle operazioni viene restituito in una nuova SCL, anziché modificare una variabile già allocata.

*Esempio:*

```
TipoSCL addSCL(TipoSCL scl, TipoInfoSCL e);
```

invece di

```
void addSCL(TipoSCL *scl, TipoInfoSCL e);
```

### 9.10.1. Funzioni primitive

```
// restituisce true se scl e' NULL
int emptySCL(TipoSCL scl);

// restituisce il primo elemento di una scl non vuota
TipoInfoSCL primoSCL(TipoSCL scl);

// restituisce il resto di una scl non vuota
TipoSCL restoSCL(TipoSCL scl);

// aggiunge l'elemento e in prima posizione alla SCL
TipoSCL addSCL(TipoSCL scl, TipoInfoSCL e);
```

### 9.10.2. Primo

```
TipoInfoSCL primoSCL(TipoSCL scl) {  
    if (!emptySCL(scl))  
        return scl->info;  
    else {  
        printf("primo di una lista vuota");  
        return ErrInfoSCL;  
    }  
}
```

### 9.10.3. Resto

```
TipoSCL restoSCL(TipoSCL scl) {  
    if (!emptySCL(scl))  
        return scl->next;  
    else {  
        printf("resto di una lista vuota");  
        return NULL;  
    }  
}
```

### 9.10.4. Costruzione di una SCL

```
TipoSCL addSCL(TipoSCL scl, TipoInfoSCL e){  
    TipoSCL temp = (TipoNodoSCL*) malloc(sizeof(  
        TipoNodoSCL));  
    temp->info = e;  
    temp->next = scl;  
    return temp;  
}
```

### 9.10.5. Copia di una SCL

```
// copia la SCL
TipoSCL copySCL(TipoSCL scl){
    if (emptySCL(scl))
        return NULL;
    else
        return addSCL(copySCL(restoSCL(scl)),
                      primoSCL(scl));
}
```

### 9.10.6. Inserimento in posizione n

Implementazione tramite memoria condivisa

```
// aggiunge e in posizione n > 0
// assume che la SCL contenga almeno n-1 elementi
TipoSCL addPosSCL(TipoSCL scl, TipoInfoSCL e, int n) {
    if (n == 0)
        return addSCL(scl,e);
    else
        return addSCL(addPosSCL(restoSCL(scl),e,n-1),
                      primoSCL(scl));
}
```

Implementazione tramite copia

```
// aggiunge e in posizione n > 0
// assume che la SCL contenga almeno n-1 elementi
TipoSCL addPosSCL(TipoSCL scl, TipoInfoSCL e, int n) {
    if (n == 0)
        return addSCL(copySCL(scl),e);
    else
        return addSCL(addPosSCL(restoSCL(scl),e,n-1),
                      primoSCL(scl));
}
```

### 9.10.7. Eliminazione in posizione n

Implementazione tramite memoria condivisa

```
// restituisce una SCL senza l'elemento in posizione n
// assume che la SCL contenga almeno n elementi
TipoSCL delPosSCL(TipoSCL scl, int n) {
    if (n == 0)
        return restoSCL(scl);
    else
        return addSCL(delPosSCL(restoSCL(scl),n-1),
                      primoSCL(scl));
}
```

Implementazione tramite copia

```
// restituisce una SCL senza l'elemento in posizione n
// assume che la SCL contenga almeno n elementi
TipoSCL delPosSCL(TipoSCL scl, int n) {
    if (n == 0)
        return copySCL(restoSCL(scl));
    else
        return addSCL(delPosSCL(restoSCL(scl),n-1),
                      primoSCL(scl));
}
```

### 9.10.8. Modifica in posizione n

Implementazione tramite memoria condivisa

```
// modifica elemento in posizione n > 0 con valore e
// assume che la SCL contenga almeno n-1 elementi
TipoSCL setPosSCL(TipoSCL scl, TipoInfoSCL e, int n) {
    if (n == 0)
        return addSCL(scl->next,e);
    else
        return addSCL(setPosSCL(restoSCL(scl),e,n-1),
                      primoSCL(scl));
}
```

Implementazione tramite copia

```
// modifica elemento in posizione n > 0 con valore e
// assume che la SCL contenga almeno n-1 elementi
TipoSCL setPosSCL(TipoSCL scl, TipoInfoSCL e, int n) {
    if (n == 0)
        return addSCL(copySCL(scl->next),e);
    else
        return addSCL(addPosSCL(restoSCL(scl),e,n-1),
                       primoSCL(scl));
}
```