Laurea in Ingegneria dell'Informazione

Esercitazioni Guidate di Tecniche della Programmazione

Note introduttive:

1) Nel titolo di ogni sezione di questo documento è specificato tra parentesi il nome del (o dei) file in cui è proposta una soluzione (se disponibile nella directory "`programmi`" di questa EG).
2) I programmi che scriveremo dovranno essere in accordo con la definizione standard **ANSI C** del linguaggio C.
    a. Se usate un sistema diverso dal DEV, provvedete a che la compilazione avvenga con il compilatore standard C.
    b. Ricordate che un programma C è in un file con estensione "`.c`"
    c. Se usate il DEVC++, per configurare bene la compilazione bisogna
        i. andare nel menù "Tools", selezionare "Compiler Options", scegliere "Settings" e poi "C Compiler"; poi selezionare almeno "Support all ANSI Standard C Programs")
        ii. (se l'interfaccia è in italiano …) andare nel menu' "Strumenti", selezionare "Opzioni di compilazione", "Compilatore", "Generazione di Codice …", "Compilatore C" e poi far apparire "Yes" almeno accanto a "Supporto programmi ANSI standard C.

NB Spesso ci sono svariati esercizi proposti nelle slide delle lezioni; questi esercizi potrebbero essere affrontati durante la EG, anche se non vi compaiono. A meno che non li abbiate già fatti …. Se li avete fatti potete sempre farmeli vedere. Se non li avete fatti provate a farli …

10. Esercitazione 10 – liste concatenate *rappresentate mediante record e puntatori*

10.1. liste rappresentate mediante strutture e puntatori (`liste1.c`, `liste2.c`)

Scrivere un programma che
- costruisce una lista di k interi (con k letto da input)
- la stampa
- chiede un nuovo intero e lo inserisce in testa alla lista
- stampa di nuovo la lista

Per la costruzione della lista usare la funzione di costruzione **`costruisciLista()`** che, ricevendo un parametro intero `n`, costruisca una lista di `n` elementi mediante inserimenti in testa.

Per la stampa degli elementi della lista si usa la funzione **`stampaLista()`**che, ricevendo il puntatore al primo elemento di una lista, stampa la lista.

Suggerimento segue …

Suggerimento 1 di 2:

Ecco i prototipi delle funzioni richieste (+1):

```
void insTestaLista(TipoLista * plis, TipoElemLista elem);
/* inserisce elem nella lista; plis è l'indirizzo della
   variabile puntatore che punta all'inizio della lista */

TipoLista costruisciLista (int n);
   /* costruisce una lista di n elementi e ne restituisce il
   puntatore all'inizio (usa insTestaLista() */

void stampaLista(TipoLista lis);
   /* stampa tutti i dati contenuti nella lista */
```

Suggerimento 2 di 2:
questi potrebbero essere i tipi utilizzati:
NB
tutte queste strutture dati devono essere definite "top level", in modo che siano visibili a tutte le funzioni definite poi (la **`main()`** e le altre funzioni). Infatti se definissimo queste strutture dati direttamente nella **`main()`**, esse non sarebbeo poi utilizzabili da **`stampaLista()`** o **`costruisciLista`**() o quant'altro

```
typedef int
        TipoElemLista;    /* si tratta di liste di interi quindi il tipo
                             delle informazioni nei nodi della lista è int */

struct StructLista {          /* il tipo per i singoli nodi della lista */
   TipoElemLista info;
   struct StructLista *next;
};

typedef struct StructLista *
        TipoLista;             /* il tipo delle variabili che rappresentano
                                  liste all'interno del programma */

typedef TipoLista
    PuntNodoLista;        /* un tipo ausiliario: sarà il tipo delle
                             variabili puntatore a nodi di liste, ad esempio
                             usate per le scansioni di lista */
```

10.2. Funzioni adatte al tipo delle informazioni contenute in lista (`liste2.c`)

Per rendere il programma meno dipendente dal tipo di elementi contenuti nei nodi della lista, non sarebbe male corredare i programma di due funzioni
`stampaElemLista()`
e **`leggiElemLista()`**
adatte per leggere e stampare singoli elementi della lista.

Segue un suggerimento

Suggerimento:

ecco i prototipi delle funzioni richieste:

```
void stampaElemLista(TipoElemLista v);
   /* stampa l'oggetto v di tipo TipoElemLista */

void leggiElemLista(TipoElemLista *pelem);
   /* legge un oggetto di tipo TipoElemLista; e lo memorizza
   nella locazione puntata da pelem */
```

In questo programma l'uso delle funzioni suggerite sopra può sembrare ridondante: si tratta di semplici usi di **`printf()`** e **`scanf()`** su variabili intere;

ma in programmi in cui l'informazione associata ai nodi della lista è di tipo più complesso queste funzioni permetteranno una migliore strutturazione del programma.

10.3. Liste di caratteri (`liste3.c`)

Analogamente al caso precedente, scrivere un programma che
- costruisce una lista di **k** CARATTERI (con **k** letto da input)
- la stampa

Segue un suggerimento

Suggerimento:
Si può sfruttare il programma precedente, avendo l'accortezza di adattare le funzioni di lettura e stampa di elementi da inserire in lista (e solo quelle ...)

Se consultate la soluzione trovate un **easter egg**

10.4. Liste di caratteri senza sapere prima quanti sono (`liste4.c`)

Scrivere un programma che legga una sequenza di caratteri inserita da tastiera e terminata da un `'\n'` (invio) e costruisca e stampi la lista corrispondente.

Diversamente dall'esercizio precedente, qui non sappiamo prima della costruzione quanti saranno i dati da inserire in lista.

Suggerimento più sotto …

Suggerimento:
Dobbiamo realizzare un ciclo di inserimenti in testa che termina quando il carattere letto da input è `'\n'`.

Altro suggerimento più avanti …

Suggerimento 2:

uno schema possibile, facendo a meno della funzione **`costruisciLista()`** e realizzando il ciclo di lettura dati ed inserimento direttamente nella funzione **`main()`**:

- lettura primo dato
- mentre il dato letto non è **`'\n'`**
    - inserimento del dato lettpo precedentemente (con **`insTestaLista()`**)
    - lettura del prossimo dato (con **`leggiElemLista()`**)

Altro suggerimento poco più avanti …

Suggerimento 3:
ecco uno stralcio del programma nel file **`liste4.c`**:
mostriamo solo il ciclo che esegue l'inserimento dell'ultimo dato letto (diverso da '\n' e la lettura del prossimo.

Le variabili citate sono definite nella **`main()`**, con significato evocato dai rispettivi identificatori)

```
  while (datiTerminati==0) {
    insTestaLista(&list, el);
    leggiElemLista(&el);       /* lettura elemento successivo */

    datiTerminati=(el=='\n'); /* assegnazione variabile per la
                                 condizione di ripetizione   */
  }
```

10.5. Inserimento in coda (`liste5.c`)

Scrivere un programma che legga una sequenza di caratteri inserita da tastiera e terminata da un **`'\n'`** (invio) e costruisca e stampi la lista corrispondente.

Ma stavolta è richiesto che l'ordine degli elementi in lista corrisponda esattamente a quello di inserimento: il primo elemento in lista deve essere quello inserito in input per primo e così via ...

Si consiglia di costruire il ciclo di inserimenti in coda direttamente nella **`main()`**.

ecco i prototipi delle funzioni definite nel file `liste5.c`:

```
int ugualiElem(TipoElemLista, TipoElemLista);
/* aggiunta, per confrontare elemnti della lista */

void stampaLista(TipoLista lis);
void stampaElemLista(TipoElemLista v);
void leggiElemLista(TipoElemLista *pelem);
```