Capitolo 9

Lezione 3 maggio 2019
Esercitazione 5

9.1 Modalità di consegna e valutazione esercitazioni

V. Esercitazione 1, 2, 3, 4.

Date di assegnazione e consegna esercitazione 5 3 Maggio → 12 maggio

Note Le funzioni consegnate non devono produrre nessun output su terminale. Nel caso siano presenti istruzioni che producono output (es. utilizzando printf), toglietele/commentatele prima di consegnare poiché possono allungare il tempo di esecuzione ed interferire con il processo di correzione. Il programma deve poter essere compilato senza errori o warning con le opzioni `-g3 -fsanitize=address -fsanitize=undefined -std=gnu89 -pedantic-errors -Wall -Wextra` ed essere eseguito senza errori.
IMPORTANTE: la memoria allocata dinamicamente deve essere rilasciata prima del termine dell'esecuzione. Si raccomanda di seguire il corretto ordine di rilascio della memoria allocata per prevenire effetti indesiderati, ad esempio rilasciando prima gli elementi di un array frastagliato e poi l'array stesso.
Per tutti gli esercizi, è consentito utilizzare sia soluzioni ricorsive che soluzioni iterative.

Valutazione Come per le esercitazioni precedenti ciascun esercizio corretto (compila senza errori o warning, non causa errori di esecuzione, ritorna il valore corretto per tutti i test case) viene valutato 1. La soglia per poter accedere alla valutazione del progetto rimane fissata a 9 punti totali per le 5 esercitazioni.

9.2 Esercitazione 5 - Esercizio A

Scrivere una funzione C che, dati una lista di interi ordinata in ordine crescente ed un intero, modifichi la lista inserendo l'intero in modo tale che la lista risultante sia ordinata.

Consegna Per la lista di interi utilizzare il tipo `tlista` definito come segue:

```
struct cella {
   int valore;
   struct cella *next;
};
typedef struct cella *tlista;
```

Per l'ultimo elemento della lista il valore di next deve essere NULL. Una lista vuota è rappresentata dal valore NULL. Il prototipo della funzione **DEVE** essere

```
void inserisci(tlista *lista, int valore);
```

9.3 Esercitazione 5 - Esercizio B

Scrivere una funzione C che, date due liste di interi ordinate, ritorni una nuova lista ordinata che contenga solo gli elementi che compaiono in entrambe, **duplicando le celle delle liste sorgente.** La lista risultante non deve contenere duplicati, che invece possono essere presenti nelle liste di partenza.

Consegna Il prototipo della funzione **DEVE** essere

```
tlista intersezione(tlista lista1, tlista lista2);
```

Utilizzare la definizione di `tlista` indicata nell'esercizio A.

9.4 Esercitazione 5 - Esercizio C

Scrivere una funzione C che, date due liste di interi non ordinate, determini se la prima lista contiene la seconda come sottosequenza. La funzione deve ritornare 1 nel caso lista2 sia una sottosequenza di lista1, 0 altrimenti.

Consegna Il prototipo della funzione **DEVE** essere

```
int  sottosequenza(tlista lista1, tlista lista2);
```

Utilizzare la definizione di `tlista` indicata nell'esercizio A.

Esempio Date le liste

```
s1: 10, 2, 3, 4, 5
s2: 2, 3, 4
s3: 2, 3, 5
s4: 2, 3, 10
s5: 2, 3, 3, 4
```

s2 è sottosequenza di s1, mentre s3 (3 e 5 non sono consecutivi in s1), s4 (gli elementi non sono nell'ordine corretto) ed s5 (l'elemento 3 è ripetuto) non lo sono.

9.5 Esercitazione 5 - Esercizio D

Scrivere una funzione C che, date due liste di interi non ordinate, determini se la prima lista contiene tutti gli elementi della seconda nel medesimo ordine. Si supponga che per entrambe le liste tutti gli elementi siano distinti. La funzione deve ritornare 1 in caso affermativo, 0 altrimenti.

Consegna Il prototipo della funzione **DEVE** essere

```
int  contiene_in_ordine(tlista lista1, tlista lista2);
```

Utilizzare la definizione di `tlista` indicata nell'esercizio A.

Esempio Date le liste

```
s1: 10, 2, 3, 4, 5
s2: 2, 3, 4
s3: 2, 3, 5
s4: 2, 3, 10
s5: 2, 3, 8
```

gli elementi di s2 ed s3 sono contenuti in s1 nel medesimo ordine, mentre gli elementi di s4 (ordine non rispettato) ed s5 (8 assente in s1) non lo sono.

9.6 Esercitazione 5 - Esercizio E

Implementare le operazioni indicate per uno stack (pila) di interi definito come segue:

```
struct cella {
   int valori[VALORI_PER_CELLA];
   int numero_valori;
   struct cella *next;
};
typedef struct cella *tstack;
```

Una pila vuota è rappresentata da NULL. Per l'ultima cella, next è uguale a NULL. VALORI_PER_CELLA è dichiarato nel file .h utilizzando #define.

Gli elementi devono essere inseriti nell'array valori fino a che ci sono posizioni disponibili, quando l'array è pieno deve essere aggiunta una nuova cella. **Si ricorda che gli inserimenti e le estrazioni in uno stack seguono una disciplica LIFO: last in first out (ogni estrazione rimuove l'elemento più recente disponibile)**.

Consegna I prototipi delle funzioni **DEVONO** essere

```
/* ritorna uno stack vuoto */
tstack getempty();
/*  ritorna 1 se lo stack s è vuoto, 0 altrimenti */
int isempty(tstack s);
/* inserisce l'intero v nello stack s. Ritorna 1 in caso di successo, 0 altrimenti (errore di
    allocazione). */
int push(tstack *s, int v);
/* estrae un elemento dallo stack s (secondo la disciplina LIFO) e ne copia il valore nella
    locazione di memoria indicata dal puntatore v. Ritorna 1 in caso di successo (stack non vuoto)
    , 0 altrimenti (stack vuoto). */
int pop(tstack *s, int *v);
```