Quiz ed esercizi teorici

1)

Per ciascuno dei seguenti tipi, scriverne la dichiarazione (utilizzando un unico comando/dichiarazione):

a) “r” è un puntatore a un array di 8 elementi, ciascuno dei quali è un puntatore a float;

b) “s” è una funzione che accetta come parametri un intero e un puntatore a char, e restituisce un puntatore a double;

c) “t” è un array di 5 puntatori a funzioni che accettano un float e restituiscono un int.

Esempio: per "y è un puntatore a intero" scriveremmo: int *y;

2)

Considera la seguente funzione:

```
void enigma(int *a, int n) {
    int *p = a + n - 1;
    while (a < p) {
        *a ^= *p;
        *p ^= *a;
        *a ^= *p;
        a++;
        p--;
    }
}
```

a) Cosa fa questa funzione? (2 punti)

b) Qual è la complessità temporale di questa funzione in termini di n? (1 punto)

c) Se chiamassimo enigma(arr, 5) con arr = {1, 2, 3, 4, 5}, quale sarebbe il contenuto di arr dopo l'esecuzione? (1 punto)

La risposta deve seguire questo schema:

- Funzione della enigma: <risposta qui>
- Complessità temporale: O(<risposta qui>)
- arr dopo enigma(arr, 5): <risposta qui>

3)

Per ciascuno dei seguenti tipi, scriverne la dichiarazione (utilizzando in un unico comando/dichiarazione):

a) p è un puntatore ad un array di 6 elementi, ciascuno dei quali è un puntatore ad un intero;

b) z è un array di 11 elementi, ciascuno dei quali è un puntatore a intero

c) q è un puntatore ad una funzione che riceve come parametro un reale a doppia precisione e restituisce un carattere

4)

Dato il seguente codice, specificare cosa stampa ciascuna printf (indicate con 1) - 3)) ed indicare, per le stesse tre linee di codice, se ci sono errori, se viene fatto qualcosa di scorretto, motivando brevemente le risposte.

```
#include <stdio.h>


int x = 10;


int fun(int y) {
    static int x = 1;
    x += y;
    return x;
}


int main(void) {
    int y = fun(2);
    printf("%d\n", x/y * 3);    // 1)
    {
        int x = 3;
        int y = fun(x);
        printf("%d\n", fun(y)+x);  // 2)
    }
    printf("%d\n", fun(y)+x);   // 3)
}
```

5)

La seguente funzione prende in input un array b (di interi) e due interi i,j e modifica l'array b invertendo il sotto-array b[i] ... b[j] (che quindi diventa b[j], b[j-1], ..., b[i]). La funzione assume l'esistenza di una funzione ausiliaria swap che scambia il contenuto di b[i] con quello di b[j].

- Scrivere PRE e POST condizioni (notare che le PRE devono rendere i possibili i controlli con l'idea di sotto-array b[i] ... b[j], comprendendo anche il caso del sotto-array vuoto).

- Se volessimo dimostrare la correttezza della funzione, su cosa dovremmo fare induzione? Argomentare la risposta (ovvero spiegare perché la risposta data consente di ottenere la correttezza induttivamente).

```
void reverse(int* b, int i, int j) {
    if (i>=j) {return;}
    swap(b[i], b[j]);
    reverse(b, i+1, j-1);
}
```

6)

Data la seguente funzione, scrivere PRE e POST condizioni e dimostrarne la correttezza.

```
int f(int X[], int dim) {
    if (dim==0)
        return 1;
    if (X[0]%3==0)
        return f(X+1, dim-1);
    else
        return 0;
}
```

7)

Implementare una funzione ricorsiva che, dati due array A e B, restituisce:

- 1 se tutti i valori di A sono presenti in B (ciascuno con lo stesso segno o con segno opposto) nello stesso ordine;
- 0 altrimenti.

Esempi:

- La funzione restituisce 1 per A={1,-2,3} e B={1,9,8,2,8,6,3,7,9}
- La funzione restituisce 0 per A={1,2,3} e B={1,9,8,4,8,6,3,7,2}

N.B. Soluzioni iterative che richiamano la funzione stessa al termine senza calcolare niente ricorsivamente non saranno accettate.

8)

La funzione rimuovi_triple(), dato un array di interi A, elimina tutti gli elementi di A uguali al precedente e al successivo (il primo e l'ultimo elemento non verranno mai eliminati). Il tipo restituito dalla funzione è void. I parametri formali della funzione sono l'array A contenente gli interi e la sua dimensione dim (un intero). Dopo l'invocazione della funzione, i parametri attuali corrispondenti ad A e dim devono rispettare le seguenti condizioni:

- dim equivale al numero di elementi superstiti (non cancellati) dell'array A.
- A deve contenere nelle prime dim posizioni i valori superstiti.

Esempi di input/output:

- A=1,1,1,2,6,5 diventa A=1,1,2,6,5
- A=2,1,1,1,1,2,6,5 diventa A=2,1,1,2,6,5

Scrivere la PRE della funzione rimuovi_triple().

Funzioni

Per ognuna delle funzioni presenti, scrivi PRE e POST. Se prevedi una versione iterativa, prova anche quella ricorsiva, anche solo al fine di esercitarti.

1)

Scrivere una funzione che trovi il livello dell'albero con la somma massima dei valori dei nodi.

```
int livello_somma_massima(struct nodo_albero* radice);
```

2)

Implementare una funzione ricorsiva che verifichi se una lista concatenata è palindroma.

```
int is_palindromo(struct nodo* head);
```

3)

Scrivere una funzione che inserisca un nuovo nodo in una lista mantenendo l'ordine crescente.

```
struct nodo* inserisci_ordinato(struct nodo* lista, int valore);
```

4)

Implementare una funzione che fonda due liste alternando i loro elementi.

```
struct nodo* fondi_alternate(struct nodo* lista1, struct nodo* lista2);
```

5)

Scrivere una funzione che trovi il sotto-array contiguo con la somma massima all'interno di un array di interi.

```
int sotto_array_somma_massima(int* array, int lunghezza, int* inizio,
int* fine);
```