Teoria

Data la seguente dichiarazione di array bidimensionale:

```
int A[4][6];
```

Utilizzando l'aritmetica dei puntatori, scrivere la formula per ottenere l'indirizzo dell'elemento A[2][4]

Scrivere la PRE e la POST della seguente funzione:

```
int conta_pari(int *arr, int size);
```

che, dato un array di interi `arr` di dimensione `size`, conti il numero di elementi pari presenti nell'array

Scrivere la PRE e la POST della seguente funzione:

```
void elimina_multipli(int *arr, int *size, int k);
```

che, dato un array di interi `arr` di dimensione `*size` e un intero `k`, elimini dall'array tutti i multipli di `k`. La funzione deve aggiornare il valore di `*size` con la nuova dimensione dell'array dopo l'eliminazione dei multipli.

Considerata la seguente struttura:

```
struct punto {
    int x: 4;
    int y: 4;
    int z: 8;
};
```

Quanto vale `3*sizeof(struct punto)-2`?

Data la seguente funzione ricorsiva:

```
int f(int n) {
    if (n <= 0) return 0;
    return n + f(n-3);
}
```

- Qual è il parametro su cui viene fatta la ricorsione?
- Qual è la misura di complessità del problema?
- Spiegare perché questa misura decresce ad ogni chiamata ricorsiva.
- Determinare il fattore minimo di decrescita della misura di complessità ad ogni chiamata ricorsiva.

Data la seguente dichiarazione:

```
int (*f)(int, int);
```

Cosa rappresenta f? Come si potrebbe assegnare una funzione a questo puntatore?

Dato il seguente codice:

```
int x = 5;

void func(int *p) {
    int x = 10;
    *p = x;
}

int main(void) {
    int *p = &x;
    func(p);
    printf("%d\n", x);
}
```

Cosa stampa il programma? Spiegare il concetto di passaggio per riferimento e il suo effetto sulle variabili.

Data la funzione:

```
void reverse_string(char *str, int start, int end) {
    if (start >= end) return;
    char temp = str[start];
    str[start] = str[end];
    str[end] = temp;
```

```
    reverse_string(str, start+1, end-1);
}
```

- Scrivere le pre e post condizioni della funzione.
- Spiegare il funzionamento della funzione e dimostrarne la correttezza usando l'induzione

Considerando il seguente codice:

```
#define MULTIPLY(a, b) a * b
int x = 5, y = 10;
int result = MULTIPLY(x+2, y+3);
printf("%d", result);
```

Cosa viene stampato? Spiegare il problema di precedenza degli operatori nelle macro e come risolverlo

Considerando il seguente codice:

```
int add(int a, int b) { return a + b; }
int subtract(int a, int b) { return a - b; }
int multiply(int a, int b) { return a * b; }
int divide(int a, int b) { return a / b; }

int compute(int (*op[])(int, int), int n, int x, int y) {
    return op[n](x, y);
}
```

Come si chiamerebbe la funzione `compute` per eseguire un'operazione specifica (ad esempio, addizione, sottrazione, moltiplicazione, divisione) passando l'indice dell'operazione desiderata?

Dato il seguente codice:

```
int arr[3][4] = {{1,2,3,4}, {5,6,7,8}, {9,10,11,12}};
int *p = (int *)arr;
printf("%d", *(p+5));
```

Cosa viene stampato? Spiegare come funziona l'aritmetica dei puntatori con gli array multidimensionali

Funzioni

- Implementare una funzione con firma:

```
ListNode* alterna_liste(ListNode *l1, ListNode *l2);
```

che, date due liste concatenate l1 e l2, restituisca una nuova lista contenente alternativamente gli elementi di l1 e l2. Se una lista è più lunga dell'altra, gli elementi rimanenti vanno aggiunti in coda alla lista risultante.

Esempio: l1: 1 -> 3 -> 5 l2: 2 -> 4 -> 6 -> 7 -> 8 Output: 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8

- Scrivere una funzione ricorsiva con firma:

```
int cammino_somma(int *mat, int r, int c, int sum, int x, int y);
```

che, data una matrice mat di dimensioni r x c e un intero sum, determini se esiste un cammino dalla cella (0,0) alla cella (r-1, c-1) tale che la somma dei valori delle celle attraversate sia uguale a sum. È possibile muoversi solo in basso e a destra. La funzione deve restituire 1 se esiste un tale cammino, 0 altrimenti.

- Implementare una funzione con firma:

```
void elimina_dispari(TreeNode **root);
```

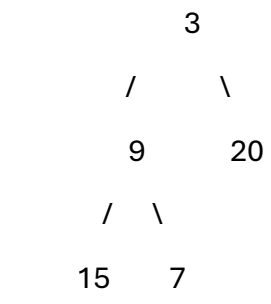
che, dato un albero binario di ricerca, elimini tutti i nodi contenenti valori dispari. L'albero risultante deve essere ancora un BST valido. Discutere la correttezza della funzione.

- Implementare una funzione con firma:

```
void zigzag_stampa(TreeNode *root);
```

che, dato un albero binario, stampi i suoi nodi in ordine "zig-zag": prima il livello 0 da sinistra a destra, poi il livello 1 da destra a sinistra, poi il livello 2 da sinistra a destra, e così via.

Esempio:



Output: 3 20 9 15 7

- Implementare una funzione ricorsiva con firma:

```
int somma_nodi(ListNode *head, int k);
```

che, data una lista concatenata head e un intero k, restituisca la somma di tutti i nodi che si trovano a distanza k dalla fine della lista.

Esempio:

Input: 1 -> 2 -> 3 -> 4 -> 5 -> 6, k = 2

Output: 9 (4 + 5, i nodi a distanza 2 dalla fine)

- Scrivere una funzione:

```
int cerca_sottostringa(char *str, char *sub)
```

che, date due stringhe `str` e `sub`, verifichi se `sub` è una sottostringa di `str`. La funzione deve restituire 1 se `sub` è presente in `str`, 0 altrimenti.

- Implementare una funzione

```
int verifica_anagramma(char *str1, char *str2)
```

che, date due stringhe `str1` e `str2`, verifichi se sono anagrammi (cioè se contengono le stesse lettere con le stesse frequenze). La funzione deve restituire 1 se le stringhe sono anagrammi, 0 altrimenti.

- Scrivi una funzione per verificare se un albero binario è un BST valido:

```
int verifica_bst(TreeNode *root, int min, int max);
```

Dato un albero binario con radice root, questa funzione deve verificare se l'albero è un BST valido. Può utilizzare i parametri min e max per tenere traccia dei limiti dei valori dei nodi durante la traversata dell'albero.

- Scrivi una funzione per trovare il numero di nodi in un livello specifico di un albero binario:

```
int conta_nodi_livello(TreeNode *root, int level);
```

Dato un albero binario con radice `root` e un intero `level`, questa funzione deve restituire il numero di nodi presenti al livello specificato dell'albero.