1. Esercizio sulle liste concatenate

Implementare una funzione ricorsiva chiamata `remove_even` che rimuova tutti gli elementi con valore pari da una lista concatenata di interi. La funzione deve restituire il puntatore alla nuova testa della lista.

Codice di base:

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    int value;
    struct node *nextPtr;
};

typedef struct node List;

List* remove_even(List *head);

// Implementare la funzione remove_even qui
```

3. Esercizio sulle matrici:

Traccia:
Implementare una funzione chiamata `rotate_matrix` che ruoti una matrice quadrata di 90 gradi in senso orario. La rotazione deve essere fatta in-place, senza utilizzare una matrice ausiliaria.

Codice di base:

```c
#include <stdio.h>

#define N 4  // Dimensione della matrice

void rotate_matrix(int matrix[N][N]);

// Implementare la funzione rotate_matrix qui
```

4. Esercizio sulla ricorsione:

Traccia:
Implementare una funzione ricorsiva chiamata `find_path` che trovi un percorso in un labirinto rappresentato da una matrice. Il labirinto è rappresentato da una matrice di 0 e 1, dove 0 rappresenta un passaggio libero e 1 rappresenta un muro. La funzione deve restituire 1 se esiste un percorso dal punto di partenza (0,0) al punto di arrivo (n-1, m-1), altrimenti 0.

Codice di base:

```
#include <stdio.h>

#define MAX 100

int find_path(int maze[MAX][MAX], int n, int m, int x, int y);

// Implementare la funzione find_path qui
```

5. Esercizio sulle stringhe e puntatori:

Traccia:
Implementare una funzione chiamata `reverse_words` che inverta l'ordine delle parole in una stringa. Le parole sono separate da uno spazio. La funzione deve modificare la stringa in-place, senza utilizzare array ausiliari.

Codice di base:

```
#include <stdio.h>
#include <string.h>

void reverse_words(char *str);

// Implementare la funzione reverse_words qui
```

6. Esercizio sulla ricorsione e backtracking:

Traccia: Implementare una funzione ricorsiva chiamata `generate_parentheses` che generi tutte le combinazioni valide di n coppie di parentesi. Una combinazione è considerata valida se le parentesi sono bilanciate.

Codice di base:

```
#include <stdio.h>
#include <stdlib.h>
```

```
void generate_parentheses(int n, char* current, int open, int close, int
index);

// Implementare la funzione generate_parentheses qui

int main() {
    int n = 3;  // Numero di coppie di parentesi
    char* current = (char*)malloc((2*n + 1) * sizeof(char));
    generate_parentheses(n, current, 0, 0, 0);
    free(current);
    return 0;
}
```

7. Esercizio su alberi binari

Implementare una funzione ricorsiva chiamata `is_symmetric` che determini se un albero binario è simmetrico (a specchio rispetto al suo centro).

```
#include <stdio.h>
#include <stdbool.h>

struct TreeNode {
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
};

bool is_symmetric(struct TreeNode* root);

// Implementare la funzione is_symmetric qui
```

8. Esercizio su array e ordinamento:

Traccia: Implementare una funzione chiamata `merge_sorted_arrays` che unisca due array ordinati in modo crescente in un unico array ordinato. La funzione deve gestire efficacemente array di lunghezze diverse.

Codice di base:

```
#include <stdio.h>
#include <stdlib.h>

int* merge_sorted_arrays(int* arr1, int size1, int* arr2, int size2, int*
result_size);
```

```
// Implementare la funzione merge_sorted_arrays qui
```