Zadania 2

2 Napisz funkcję apply, która przyjmuje wskaźnik na funkcję int func(int), tablicę liczb całkowitych i jej rozmiar jako argumenty. Funkcja apply powinna zastosować funkcję, do której wskaźnik jest przekazany, do każdego elementu tablicy i zwrócić sumę wyników. Stwórz przypadek testowy.

```c
#include <stdio.h>

int func(int num) {
    // Przykładowa funkcja - podwaja wartość
    return 2 * num;
}

int apply(int (*funcPtr)(int), int arr[], int size) {
    int sum = 0;
    for (int i = 0; i < size; i++) {
        sum += funcPtr(arr[i]);
    }
    return sum;
}

int main() {
    // Przykładowa tablica
    int arr[] = {1, 2, 3, 4, 6};
    int size = sizeof(arr) / sizeof(arr[0]);

    // Wywołanie funkcji apply
    int result = apply(func, arr, size);

    // Wyświetlenie wyniku
    printf("Suma wyników: %d\n", result);

    return 0;
}
```

2 Napisz funkcję find_if, która przyjmuje tablicę liczb całkowitych, jej rozmiar i wskaźnik na funkcję int predicate(int) jako argumenty. Funkcja find_if powinna zwrócić wskaźnik na pierwszy element tablicy, dla którego funkcja predicate zwraca wartość różną od zera. Jeżeli nie ma takiego elementu, funkcja powinna zwrócić NULL. Stwórz przypadek testowy.

```c
#include <stdio.h>

int predicate(int num) {
    // Przykładowa funkcja - sprawdza, czy liczba jest parzysta
    return num % 2 == 0;
}

int* find_if(int arr[], int size, int (*predicatePtr)(int)) {
    for (int i = 0; i < size; i++) {
        if (predicatePtr(arr[i]) != 0) {
            return &arr[i];
        }
    }
    return NULL;
}

int main() {
    // Przykładowa tablica
    int arr[] = {1, 3, 4, 5, 7};
    int size = sizeof(arr) / sizeof(arr[0]);

    // Wywołanie funkcji find_if
    int* result = find_if(arr, size, predicate);

    // Wyświetlenie wyniku
    if (result != NULL) {
        printf("Znaleziono pierwszy element: %d\n", *result);
    } else {
        printf("Nie znaleziono elementu spełniającego warunek.\n");
    }

    return 0;
}
```

2 Napisz funkcję count_if, która przyjmuje tablicę liczb całkowitych, jej rozmiar i wskaźnik na funkcję int predicate(int) jako argumenty. Funkcja count_if powinna zwrócić liczbę elementów tablicy, dla których funkcja predicate zwraca wartość różną od zera. Stwórz przypadek testowy

```c
#include <stdio.h>

int predicate(int num) {
    // Przykładowa funkcja - sprawdza, czy liczba jest parzysta
    return num % 2 == 0;
}

int count_if(int arr[], int size, int (*predicatePtr)(int)) {
    int count = 0;
    for (int i = 0; i < size; i++) {
        if (predicatePtr(arr[i]) != 0) {
            count++;
        }
    }
    return count;
}

int main() {
    // Przykładowa tablica
    int arr[] = {1, 2, 3, 4, 5};
    int size = sizeof(arr) / sizeof(arr[0]);

    // Wywołanie funkcji count_if
    int result = count_if(arr, size, predicate);

    // Wyświetlenie wyniku
    printf("Liczba elementów spełniających warunek: %d\n", result);

    return 0;
}
```

2 Napisz funkcję sum_of_squares, która ma dwa argumenty. Pierwszym argumentem jest wskaźnik num1 na stałą wartość typu int, a drugim argumentem jest stały wskaźnik num2 na zmienną typu int. Funkcja sum_of_squares ma zwrócić liczbę całkowitą zawierającą sumę kwadratów wartości wskazywanej przez pierwszy i drugi wskaźnik. Stwórz przypadek testowy dla funkcji.

```c
#include <stdio.h>

int sum_of_squares(const int* num1, const int* num2) {
    int sum = (*num1) * (*num1) + (*num2) * (*num2);
    return sum;
}

int main() {
    // Przykładowe liczby
    int num1 = 2;
    int num2 = 3;

    // Wywołanie funkcji sum_of_squares
    int result = sum_of_squares(&num1, &num2);

    // Wyświetlenie wyniku
    printf("Suma kwadratów: %d\n", result);

    return 0;
}
```

2 Napisz bezargumentową funkcję init_block, która rezerwuje blok czterech zmiennych typu float. Funkcja ma ustawić kolejno w pamięci wartości 0.5, 1.5, 2.5 i 3.5. Na koniec funkcja powinna zwrócić wskaźnik na początkową zmienną z bloku. Stwórz przypadek testowy w main tak, aby wyświetlić na konsoli wartości zmiennych przechowywanych na bloku stworzonym wewnątrz funkcji.

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

float* init_block() {
    float* block = (float*)malloc(4 * sizeof(float));
    if (block == NULL) {
        printf("Błąd alokacji pamięci.\n");
        return NULL;
    }

    block[0] = 0.5;
    block[1] = 1.5;
    block[2] = 2.5;
    block[3] = 3.5;

    return block;
}

int main() {
    float* block = init_block();
    if (block != NULL) {
        for (int i = 0; i < 4; i++) {
            printf("Wartość zmiennej %d: %.1f\n", i + 1, block[i]);
        }
        free(block);
    }

    return 0;
}
```

2 Napisz funkcję, która otrzymuje trzy argumenty: dodatnią liczbę całkowitą m, liczbę całkowitą n ($n > 1$) oraz m-elementową tablicę tab o elementach typu int. Funkcja ma zwrócić iloczyn wartości elementów tablicy tab podzielnych przez n. W przypadku braku takich elementów zwróć 1. Stwórz dwa przypadki testowe dla funkcji.

```c
#include <stdio.h>

int calculate_product(int m, int n, int tab[]) {
    int product = 1;
    int found = 0;

    for (int i = 0; i < m; i++) {
        if (tab[i] % n == 0) {
            product *= tab[i];
            found = 1;
        }
    }

    if (found == 0) {
        return 1;
    } else {
        return product;
    }
}

int main() {
    // Przypadek testowy 1
    int m1 = 5;
    int n1 = 2;
    int tab1[] = {1, 2, 3, 4, 6};
    int result1 = calculate_product(m1, n1, tab1);
    printf("Iloczyn podzielnych przez %d: %d\n", n1, result1);

    // Przypadek testowy 2
    int m2 = 4;
    int n2 = 3;
    int tab2[] = {2, 4, 6, 8};
    int result2 = calculate_product(m2, n2, tab2);
    printf("Iloczyn podzielnych przez %d: %d\n", n2, result2);

    return 0;
}
```

2 Napisz funkcję, której argumentem jest napis. Funkcja ma za zadanie zwrócić liczbę znaków, które są cyframi z systemu szesnastkowego. W zadaniu nie korzystaj z funkcji bibliotecznych poza instrukcjami wejścia/wyjścia. Stwórz przypadek testowy.

```c
#include <stdio.h>

int count_hex_digits(const char* str) {
    int count = 0;
    int i = 0;

    while (str[i] != '\0') {
        if ((str[i] >= '0' && str[i] <= '9') || (str[i] >= 'A' && str[i] <= 'F') || (str[i] >= 'a' && str[i] <= 'f')) {
            count++;
        }
        i++;
    }

    return count;
}

int main() {
    const char* str = "AbCd1234EfGh";
    int digitCount = count_hex_digits(str);
    printf("Liczba znakow szesnastkowych w napisie: %d\n", digitCount);

    return 0;
}
```

2 Napisz funkcję, która dostaje w argumencie napis i zamienia wszystkie występujące w nim duże litery na znak '@'. Następnie usuń wszystkie znaki '@' z napisu. W zadaniu nie korzystaj z funkcji bibliotecznych poza instrukcjami wejścia/wyjścia. Stwórz przypadek testowy

```c
#include <stdio.h>

void replace_and_remove(char* str) {
    int i = 0;
    int j = 0;

    // Zamiana dużych liter na '@'
    while (str[i] != '\0') {
        if (str[i] >= 'A' && str[i] <= 'Z') {
            str[i] = '@';
        }
        i++;
    }

    // Usunięcie znaków '@'
    i = 0;
    while (str[i] != '\0') {
        if (str[i] != '@') {
            str[j] = str[i];
            j++;
        }
        i++;
    }
    str[j] = '\0';
}

int main() {
    char str[] = "HeLLo@WoRlD";
    printf("Przed zamiana i usunięciem: %s\n", str);
    replace_and_remove(str);
    printf("Po zamianie i usunięciu: %s\n", str);

    return 0;
}
```

2 Napisz funkcję, której argumentami są dwa napisy. Funkcja ma przepisać z pierwszego napisu znaki na parzystych indeksach do drugiego napisu. Stwórz przypadek testowy. Przykład: Napis pierwszy jest postaci "abcdef" to do drugiego napisu mają być przepisane znaki "ace".

```c
#include <stdio.h>

void copy_even_indices(const char* str1, char* str2) {
    int i, j;
    for (i = 0, j = 0; str1[i] != '\0'; i += 2, j++) {
        str2[j] = str1[i];
    }
    str2[j] = '\0';
}

int main() {
    const char* str1 = "abcdef";
    char str2[10];
    copy_even_indices(str1, str2);
    printf("Przepisane znaki: %s\n", str2);
    return 0;
}
```

2 Napisz funkcję w języku C, której argumentami są dwa napisy. Funkcja powinna przepisać z pierwszego napisu do drugiego napisu te znaki, które na swoich miejscach mają indeksy nieparzyste. Pamiętaj, aby drugi napis był odpowiednio duży, aby pomieścić przepisywane znaki. Stwórz przypadek testowy. Przykład: Jeśli pierwszy napis to “abcdef”, to do drugiego napisu powinny zostać przepisane znaki “bdf”.

```c
#include <stdio.h>

void copy_odd_indices(const char* str1, char* str2) {
    int i, j;
    for (i = 0, j = 0; str1[i] != '\0'; i++) {
        if (i % 2 != 0) {
            str2[j] = str1[i];
            j++;
        }
    }
    str2[j] = '\0';
}

int main() {
    const char* str1 = "abcdef";
    char str2[10];
    copy_odd_indices(str1, str2);
    printf("Przepisane znaki: %s\n", str2);
    return 0;
}
```

2 Napisz funkcję w języku C, której argumentami są dwa napisy. Funkcja powinna przepisać z pierwszego napisu do drugiego napisu te znaki, które są literami wielkimi. Zadbaj o to, aby drugi napis miał odpowiednią ilość miejsca na przepisywane znaki. Stwórz przypadek testowy. Przykład: Jeśli pierwszy napis to “AbCdEf”, to do drugiego napisu powinny zostać przepisane znaki “ACE”.

```c
#include <stdio.h>

void copy_uppercase_letters(const char* str1, char* str2) {
    int i, j;
    for (i = 0, j = 0; str1[i] != '\0'; i++) {
        if (str1[i] >= 'A' && str1[i] <= 'Z') {
            str2[j] = str1[i];
            j++;
        }
    }
    str2[j] = '\0';
}

int main() {
    const char* str1 = "AbCdEf";
    char str2[10];
    copy_uppercase_letters(str1, str2);
    printf("Przepisane litery: %s\n", str2);
    return 0;
}
```

2 Napisz funkcję w języku C, która przyjmuje dwa napisy oraz dodatkowo liczbę całkowitą n. Funkcja powinna przepisać do drugiego napisu co n-ty znak z pierwszego napisu. Zadbaj o to, aby drugi napis miał wystarczającą ilość miejsca na przepisywanie znaków. Przykład: Dla napisu "abcdefghijklm", liczby n=3 oraz pustego napisu, po wykonaniu funkcji drugi napis powinien zawierać "cfil".

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

void copy_nth_character(const char* str1, char* str2, int n) {
    int length = strlen(str1);
    int j = 0;
    for (int i = n - 1; i < length; i += n) {
        str2[j] = str1[i];
        j++;
    }
    str2[j] = '\0';
}

int main() {
    const char* str1 = "abcdefghijklm";
    char str2[10];
    int n = 3;
    copy_nth_character(str1, str2, n);
    printf("Przepisane znaki: %s\n", str2);
    return 0;
}
```

2 Stwórz funkcję, która przyjmuje jako argument napis. Jeżeli napis zawiera cyfrę parzystą, funkcja powinna zwrócić długość napisu powiększoną o liczbę parzystych cyfr w napisie. W przypadku braku cyfr parzystych, funkcja powinna zwrócić normalną długość napisu. Stwórz przypadek testowy. Przykład: dla napisu “kod2468” funkcja powinna zwrócić 11, dla napisu “abc13” powinna zwrócić 5, dla napisu “2468” powinna zwrócić 8, dla napisu “wmii” powinna zwrócić 4.

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

int increase_length_if_even_digits(const char* str) {
    int length = strlen(str);
    int even_digits = 0;

    for (int i = 0; i < length; i++) {
        if (str[i] >= '0' && str[i] <= '9' && (str[i] - '0') % 2 == 0) {
            even_digits++;
        }
    }

    if (even_digits > 0) {
        return length + even_digits;
    } else {
        return length;
    }
}

int main() {
    const char* str1 = "kod2468";
    const char* str2 = "abc13";
    const char* str3 = "2468";
    const char* str4 = "wmii";

    int length1 = increase_length_if_even_digits(str1);
    int length2 = increase_length_if_even_digits(str2);
    int length3 = increase_length_if_even_digits(str3);
    int length4 = increase_length_if_even_digits(str4);

    printf("Długość napisu '%s': %d\n", str1, length1);
    printf("Długość napisu '%s': %d\n", str2, length2);
    printf("Długość napisu '%s': %d\n", str3, length3);
    printf("Długość napisu '%s': %d\n", str4, length4);

    return 0;
}
```

2 Zaimplementuj funkcję positive_count, która przyjmuje tablicę liczb całkowitych, jej rozmiar oraz wskaźnik na funkcję int is_positive(int). Funkcja positive_count powinna zwrócić liczbę elementów tablicy, dla których funkcja is_positive zwraca wartość większą od zera. Stwórz przypadek testowy.

```c
#include <stdio.h>

int is_positive(int num) {
    return num > 0;
}

int positive_count(int arr[], int size, int (*is_positive)(int)) {
    int count = 0;

    for (int i = 0; i < size; i++) {
        if (is_positive(arr[i])) {
            count++;
        }
    }

    return count;
}

int main() {
    int arr[] = {1, -2, 3, -4, 5};
    int size = sizeof(arr) / sizeof(arr[0]);

    int count = positive_count(arr, size, is_positive);

    printf("Liczba dodatnich elementów: %d\n", count);

    return 0;
}
```

2 Zaprojektuj funkcję even_count, która przyjmuje tablicę liczb całkowitych, jej rozmiar oraz wskaźnik na funkcję int is_even(int). Funkcja even_count powinna zwracać liczbę elementów w tablicy, dla których funkcja is_even zwraca wartość parzystą. Stwórz przypadek testowy.

```c
#include <stdio.h>

int is_even(int num) {
    return num % 2 == 0;
}

int even_count(int arr[], int size, int (*is_even)(int)) {
    int count = 0;

    for (int i = 0; i < size; i++) {
        if (is_even(arr[i])) {
            count++;
        }
    }

    return count;
}

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int size = sizeof(arr) / sizeof(arr[0]);

    int count = even_count(arr, size, is_even);

    printf("Liczba liczb parzystych: %d\n", count);

    return 0;
}
```

2 Stwórz rekurencyjną funkcję, która przyjmuje jako argument napis. Funkcja ma zwrócić liczbę znaków będących dużymi literami w napisie. W zadaniu nie korzystaj z funkcji bibliotecznych poza instrukcjami wejścia/wyjścia. Stwórz przypadek testowy. Uwaga: funkcja nierekurencyjna = 0pkt.

```c
#include <stdio.h>

int count_uppercase(const char *str) {
    if (*str == '\0') {
        return 0; // Warunek zakończenia rekurencji: koniec napisu
    }

    int count = 0;

    if (*str >= 'A' && *str <= 'Z') {
        count = 1; // Znaleziono duży znak
    }

    return count + count_uppercase(str + 1); // Rekurencyjne wywołanie dla kolejnego znaku
}

int main() {
    const char *str = "Hello World";

    int count = count_uppercase(str);

    printf("Liczba dużych liter: %d\n", count);

    return 0;
}
```

2 Stwórz rekurencyjną funkcję, która przyjmuje jako argument napis. Funkcja ma zwrócić liczbę znaków będących cyframi z systemu dziesiętnego w napisie. W zadaniu nie korzystaj z funkcji bibliotecznych poza instrukcjami wejścia/wyjścia. Stwórz przypadek testowy. Uwaga: funkcja nierekurencyjna = 0pkt.

```c
#include <stdio.h>

int count_digits_recursive(const char *str) {
    if (*str == '\0') {
        return 0; // Warunek zakończenia rekurencji: koniec napisu
    }

    int count = count_digits_recursive(str + 1); // Rekurencyjne wywołanie dla kolejnego znaku

    if (*str >= '0' && *str <= '9') {
        return count + 1; // Znaleziono cyfrę dziesiętną
    } else {
        return count; // Zwróć wynik bez inkrementacji licznika
    }
}


int main() {
    const char *str = "AbC123xyz45678";

    int count = count_digits_recursive(str);

    printf("Liczba cyfr: %d\n", count);

    return 0;
}
```

Zadania 3

Stwórz strukturę Kierowca o polach imie (napis) oraz przejechane_kilometry (typ całkowity). Następnie stwórz funkcję, której argumentami jest niepusta tablica struktur Kierowca oraz rozmiar tablicy. Funkcja ma zwrócić imię kierowcy, który przejechał najmniej kilometrów (w przypadku kilku równych wyników, ma zwrócić wynik ostatniego). Stwórz przypadek testowy.

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

#define MAX_NAME_LENGTH 100

// Struktura Kierowca
typedef struct {
    char imie[MAX_NAME_LENGTH];
    int przejechane_kilometry;
} Kierowca;

// Funkcja do znajdowania kierowcy o najmniejszej ilości przejechanych kilometrów
char* find_driver_with_least_distance(Kierowca* kierowcy, int rozmiar) {
    if (rozmiar <= 0) {
        return NULL; // Sprawdzenie poprawności rozmiaru tablicy
    }

    int najmniej_kilometrow = kierowcy[0].przejechane_kilometry;
    int indeks_najmniej_kilometrow = 0;

    for (int i = 1; i < rozmiar; i++) {
        if (kierowcy[i].przejechane_kilometry <= najmniej_kilometrow) {
            najmniej_kilometrow = kierowcy[i].przejechane_kilometry;
            indeks_najmniej_kilometrow = i;
        }
    }

    return kierowcy[indeks_najmniej_kilometrow].imie;
}

int main() {
    // Tworzenie tablicy kierowców
    Kierowca kierowcy[] = {
        {"Adam", 200},
        {"Barbara", 150},
        {"Cezary", 100},
        {"Dorota", 150},
        {"Edward", 250}
    };

    };

    int rozmiar = sizeof(kierowcy) / sizeof(kierowcy[0]);

    // Wywołanie funkcji find_driver_with_least_distance
    char* imie_najmniej_kilometrow = find_driver_with_least_distance(kierowcy, rozmiar);

    if (imie_najmniej_kilometrow != NULL) {
        printf("Kierowca z najmniejszą ilością przejechanych kilometrów: %s\n", imie_najmniej_kilometrow);
    } else {
        printf("Tablica kierowców jest pusta\n");
    }

    return 0;
}
```

3 Stwórz strukturę Kierowca o polach imie (napis) oraz liczba_wypadków (typ całkowity). Następnie stwórz funkcję, której argumentami jest niepusta tablica struktur Kierowca oraz rozmiar tablicy. Funkcja ma zwrócić kierowcę (jako strukturę), który miał najmniejszą liczbę wypadków (w przypadku kilku równych wyników, ma zwrócić wynik ostatniego). Stwórz przypadek testowy

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

#define MAX_NAME_LENGTH 100

// Struktura Kierowca
typedef struct {
    char imie[MAX_NAME_LENGTH];
    int liczba_wypadkow;
} Kierowca;

// Funkcja do znajdowania kierowcy o najmniejszej liczbie wypadków
Kierowca find_driver_with_least_accidents(Kierowca* kierowcy, int rozmiar) {
    if (rozmiar <= 0) {
        Kierowca kierowca_pusty;
        strcpy(kierowca_pusty.imie, "");
        kierowca_pusty.liczba_wypadkow = 0;
        return kierowca_pusty; // Zwracanie pustego kierowcy w przypadku błędnego rozmiaru
    }

    int najmniej_wypadkow = kierowcy[0].liczba_wypadkow;
    int indeks_najmniej_wypadkow = 0;

    for (int i = 1; i < rozmiar; i++) {
        if (kierowcy[i].liczba_wypadkow <= najmniej_wypadkow) {
            najmniej_wypadkow = kierowcy[i].liczba_wypadkow;
            indeks_najmniej_wypadkow = i;
        }
    }

    return kierowcy[indeks_najmniej_wypadkow];
}

int main() {
    // Tworzenie tablicy kierowców
    Kierowca kierowcy[] = {
        {"Adam", 2},
        {"Barbara", 1},
        {"Cezary", 3},
        {"Dorota", 1},
        {"Edward", 0}
    };

    int rozmiar = sizeof(kierowcy) / sizeof(kierowcy[0]);

    // Wywołanie funkcji find_driver_with_least_accidents
    Kierowca kierowca_najmniej_wypadkow = find_driver_with_least_accidents(kierowcy, rozmiar);

    if (strcmp(kierowca_najmniej_wypadkow.imie, "") != 0) {
        printf("Kierowca z najmniejszą liczbą wypadków: %s\n", kierowca_najmniej_wypadkow.imie);
    } else {
        printf("Tablica kierowców jest pusta\n");
    }

    return 0;
}
```

3 Stwórz typ wyliczeniowy Food przechowujący typy potraw. Następnie stwórz program zawierający tablicę 5 elementów typu Food. Wypisz na konsoli zawartość tablicy używając pętli i instrukcji warunkowej.

```c
#include <stdio.h>

// Typ wyliczeniowy Food przechowujący typy potraw
typedef enum {
    PIZZA,
    BURGER,
    PASTA,
    SUSHI,
    SALAD
} Food;

int main() {
    // Tworzenie tablicy 5 elementów typu Food
    Food foods[5] = {PIZZA, BURGER, PASTA, SUSHI, SALAD};

    // Wypisanie zawartości tablicy za pomocą pętli i instrukcji warunkowej
    for (int i = 0; i < 5; i++) {
        printf("Element %d: ", i);

        switch (foods[i]) {
            case PIZZA:
                printf("Pizza\n");
                break;
            case BURGER:
                printf("Burger\n");
                break;
            case PASTA:
                printf("Pasta\n");
                break;
            case SUSHI:
                printf("Sushi\n");
                break;
            case SALAD:
                printf("Salad\n");
                break;
            default:
                printf("Unknown\n");
                break;
        }
    }

    return 0;
}
```

3 Napisz strukturę Samochod z polami marka (tablica znaków długości q5) oraz przebieg (typu int). Następnie napisz dwie funkcje i wywołaj każdą z nich co najmniej jeden raz:

a) initSamochod - funkcja przyjmuje dwa argumenty: markę i przebieg, i zwraca wskaźnik nowo-utworzoną strukturę ustawiającą składowe z przekazanych argumentów. Dodatkowo funkcja powinna sprawdzić, aby marka była napisem długości co najmniej 3 i przebieg był większy niż 1000. W przypadku nie spełnienia jednego z warunków, funkcja powinna zwracać NULL.

b) dodajKilometry - funkcja, której argumentem jest wskaźnik do struktury typu Samochod. Funkcja ma dodać 1000 do przebiegu w przekazanym argumencie. Upewnij się, że drugą funkcję możesz wywołać w main.

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

#define MAX_MARKA 25

// Struktura Samochod
typedef struct {
    char marka[MAX_MARKA];
    int przebieg;
} Samochod;

// Funkcja initSamochod
Samochod* initSamochod(const char* marka, int przebieg) {
    // Sprawdzenie warunków
    if (strlen(marka) < 3 || przebieg <= 1000) {
        return NULL;
    }

    // Alokacja pamięci dla struktury Samochod
    Samochod* samochod = (Samochod*)malloc(sizeof(Samochod));
    if (samochod == NULL) {
        return NULL;
    }

    // Ustawienie wartości składowych
    strncpy(samochod->marka, marka, MAX_MARKA);
    samochod->przebieg = przebieg;

    return samochod;
}

// Funkcja dodajKilometry
void dodajKilometry(Samochod* samochod) {
    samochod->przebieg += 1000;
}

int main() {
    // Inicjalizacja samochodu
    Samochod* samochod = initSamochod("Toyota", 5000);
    if (samochod != NULL) {
        printf("Marka: %s, Przebieg: %d\n", samochod->marka, samochod->przebieg);

        // Dodanie kilometrów
        dodajKilometry(samochod);
        printf("Po dodaniu kilometrów: Przebieg: %d\n", samochod->przebieg);

        // Zwolnienie pamięci
        free(samochod);
    } else {
        printf("Błąd inicjalizacji samochodu.\n");
    }

    return 0;
}
```

3 Napisz strukturę Komputer z polami model (tablica znaków długości 50) oraz moc (typu int). Następnie napisz dwie funkcje i wywołaj każdą z nich co najmniej jeden raz:

a) initKomputer - funkcja przyjmuje dwa argumenty: model i moc, i zwraca wskaźnik nowo-utworzoną strukturę ustawiającą składowe z przekazanych argumentów. Dodatkowo funkcja powinna sprawdzić, aby model był napisem długości co najmniej 4 i moc była większa niż 100. W przypadku nie spełnienia jednego z warunków, funkcja powinna zwracać NULL.

b) zwiekszMoc - funkcja, której argumentem jest wskaźnik do struktury typu Komputer. Funkcja ma dodać 50 do mocy w przekazanym argumencie. Upewnij się, że drugą funkcję możesz wywołać w main.

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

#define MAX_MODEL 50

// Struktura Komputer
typedef struct {
    char model[MAX_MODEL];
    int moc;
} Komputer;

// Funkcja initKomputer
Komputer* initKomputer(const char* model, int moc) {
    // Sprawdzenie warunków
    if (strlen(model) < 4 || moc <= 100) {
        return NULL;
    }

    // Alokacja pamięci dla struktury Komputer
    Komputer* komputer = (Komputer*)malloc(sizeof(Komputer));
    if (komputer == NULL) {
        return NULL;
    }

    // Ustawienie wartości składowych
    strncpy(komputer->model, model, MAX_MODEL);
    komputer->moc = moc;

    return komputer;
}

// Funkcja zwiekszMoc
void zwiekszMoc(Komputer* komputer) {
    komputer->moc += 50;
}

int main() {
    // Inicjalizacja komputera
    Komputer* komputer = initKomputer("Dell XPS", 500);
    if (komputer != NULL) {
        printf("Model: %s, Moc: %d\n", komputer->model, komputer->moc);

        // Zwiększenie mocy
        zwiekszMoc(komputer);
        printf("Po zwiększeniu mocy: Moc: %d\n", komputer->moc);

        // Zwolnienie pamięci
        free(komputer);
    } else {
        printf("Błąd inicjalizacji komputera.\n");
    }

    return 0;
}
```

3 Stwórz strukturę Student o dwóch polach imie (napis) oraz ocena (typ całkowity). Następnie stwórz funkcję, której argumentami jest niepusta tablica struktur Student oraz rozmiar tablicy. Funkcja ma zwrócić imię studenta z najniższą oceną (w przypadku kilku równych ocen, ma zwrócić imię ostatniego). Stwórz przypadek testowy.

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

#define MAX_IMIE 50

// Struktura Student
typedef struct {
    char imie[MAX_IMIE];
    int ocena;
} Student;

// Funkcja najnizszaOcena
char* najnizszaOcena(Student* students, int rozmiar) {
    if (rozmiar <= 0) {
        return NULL;
    }

    // Inicjalizacja imienia i oceny pierwszego studenta jako wartości początkowej
    char* imieNajnizszejOceny = students[0].imie;
    int najnizszaOcena = students[0].ocena;

    // Iteracja przez resztę studentów i aktualizacja imienia i oceny najniższej oceny
    for (int i = 1; i < rozmiar; i++) {
        if (students[i].ocena <= najnizszaOcena) {
            imieNajnizszejOceny = students[i].imie;
            najnizszaOcena = students[i].ocena;
        }
    }

    return imieNajnizszejOceny;
}

int main() {
    // Przykładowa tablica studentów
    Student students[] = {
        {"Jan", 4},
        {"Anna", 5},
        {"Marek", 3},
        {"Ewa", 4},
        {"Piotr", 3}
    };

    int rozmiar = sizeof(students) / sizeof(Student);

    // Wywołanie funkcji najnizszaOcena
    char* imie = najnizszaOcena(students, rozmiar);

    if (imie != NULL) {
        printf("Student z najniższą oceną: %s\n", imie);
    } else {
        printf("Nie znaleziono studenta.\n");
    }

    return 0;
}
```

3 Stwórz strukturę Programista o dwóch polach nazwisko (napis) oraz doświadczenie (typ całkowity). Następnie stwórz funkcję, której argumentami jest niepusta tablica struktur Programista oraz rozmiar tablicy. Funkcja ma zwrócić nazwisko programisty z największym doświadczeniem (w przypadku kilku równych wartości, ma zwrócić nazwisko ostatniego). Stwórz przypadek testowy.

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

#define MAX_NAZWISKO 50

// Struktura Programista
typedef struct {
    char nazwisko[MAX_NAZWISKO];
    int doswiadczenie;
} Programista;

// Funkcja najwiekszeDoswiadczenie
char* najwiekszeDoswiadczenie(Programista* programisci, int rozmiar) {
    if (rozmiar <= 0) {
        return NULL;
    }

    // Inicjalizacja nazwiska i doświadczenia pierwszego programisty jako wartości początkowej
    char* nazwiskoNajwiekszegoDoswiadczenia = programisci[0].nazwisko;
    int najwiekszeDoswiadczenie = programisci[0].doswiadczenie;

    // Iteracja przez resztę programistów i aktualizacja nazwiska i doświadczenia największego doświadczenia
    for (int i = 1; i < rozmiar; i++) {
        if (programisci[i].doswiadczenie >= najwiekszeDoswiadczenie) {
            nazwiskoNajwiekszegoDoswiadczenia = programisci[i].nazwisko;
            najwiekszeDoswiadczenie = programisci[i].doswiadczenie;
        }
    }

    return nazwiskoNajwiekszegoDoswiadczenia;
}
int main() {
    // Przykładowa tablica programistów
    Programista programisci[] = {
        {"Kowalski", 3},
        {"Nowak", 5},
        {"Lewandowski", 2},
        {"Wójcik", 5},
        {"Mazur", 4}
    };

    int rozmiar = sizeof(programisci) / sizeof(Programista);

    // Wywołanie funkcji najwiekszeDoswiadczenie
    char* nazwisko = najwiekszeDoswiadczenie(programisci, rozmiar);

    if (nazwisko != NULL) {
        printf("Programista z największym doświadczeniem: %s\n", nazwisko);
    } else {
        printf("Nie znaleziono programisty.\n");
    }

    return 0;
}
```

3 Zdefiniuj strukturę Komputer z polami model (tablica znaków długości 20) oraz czasUzytkowania (typu int). Następnie, stwórz dwie funkcje:

a) initKomputer - funkcja, która przyjmuje dwa argumenty: model i czas użytkowania, i zwraca wskaźnik na nowo utworzoną strukturę Komputer z polami ustawionymi na wartości przekazane jako argumenty. Funkcja powinna sprawdzać, czy model ma długość co najmniej 4 i czy czas użytkowania jest większy niż 100. W przypadku nie spełnienia tych warunków, funkcja powinna zwracać NULL.

b) zwiekszCzasUzytkowania - funkcja, której argumentem jest wskaźnik na strukturę Komputer. Funkcja powinna zwiększyć czas użytkowania komputera o 100. Wywołaj każdą funkcję co najmniej jeden raz.

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

#define MAX_MODEL 20

// Struktura Komputer
typedef struct {
    char model[MAX_MODEL];
    int czasUzytkowania;
} Komputer;

// Funkcja initKomputer
Komputer* initKomputer(const char* model, int czasUzytkowania
    if (strlen(model) < 4 || czasUzytkowania <= 100) {
        return NULL;
    }

    Komputer* komputer = (Komputer*)malloc(sizeof(Komputer));

    if (komputer != NULL) {
        strcpy(komputer->model, model);
        komputer->czasUzytkowania = czasUzytkowania;
    }

    return komputer;
}

// Funkcja zwiekszCzasUzytkowania
void zwiekszCzasUzytkowania(Komputer* komputer) {
    if (komputer != NULL) {
        komputer->czasUzytkowania += 100;
    }
}

int main() {
    // Inicjalizacja komputera za pomoca funkcji initKomputer
    Komputer* komputer = initKomputer("Dell", 200);

    if (komputer != NULL) {
        printf("Model komputera: %s\n", komputer->model);
        printf("Czas użytkowania: %d\n", komputer->czasUzytkowania);

        // Zwiększenie czasu użytkowania komputera za pomoca funkcji zwiekszCzasUzytkowania
        zwiekszCzasUzytkowania(komputer);

        printf("Po zwiększeniu czasu użytkowania: %d\n", komputer->czasUzytkowania);

        // Zwolnienie pamięci zaalokowanej dla komputera
        free(komputer);
    } else {
        printf("Nie można zainicjalizować komputera.\n");
    }

    return 0;
}
```

3 Napisz program, w którym, Napisz funkcję, której argumentem jest dwuwymiarowa tablica tablic (zawierająca zmienne typu int) oraz jej wymiary n i m. Funkcja ma odwrócić kolejność elementów w kolumnach o nieparzystych indeksach. Stwórz przypadek testowy.

Przykład 1:	Zamienia się w przykład 2:
{2, 3, -3}	{2, 8, -3}
{1, 4, 7}	{1, -6, 7}
{-3, -6, 11}	{-3, 4, 11}
{-2, 8, 23}	{-2, 3, 23}

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

void odwroc_kolumny(int** tab, int n, int m) {
    for (int j = 1; j < m; j += 2) { // iterujemy po nieparzystych indeksach kolumn
        int i = 0, k = n - 1; // ustawiamy wskaźniki na początku i końcu kolumny
        while (i < k) {
            int temp = tab[i][j];
            tab[i][j] = tab[k][j];
            tab[k][j] = temp;
            i++;
            k--;
        }
    }
}

void wyswietl_tablice(int** tab, int n, int m) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            printf("%d ", tab[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}

int main() {
    int n = 4; // liczba wierszy
    int m = 3; // liczba kolumn

    int tablica[][3] = {
        {2, 3, -3},
        {1, 4, 7},
        {-3, -6, 11},
        {-2, 8, 23}
    };

    int** tab = (int**)malloc(n * sizeof(int*));
    for (int i = 0; i < n; i++) {
        tab[i] = (int*)malloc(m * sizeof(int));
        for (int j = 0; j < m; j++) {
            tab[i][j] = tablica[i][j];
        }
    }

    printf("Przed odwróceniem:\n");
    wyswietl_tablice(tab, n, m);

    odwroc_kolumny(tab, n, m);

    printf("Po odwróceniu:\n");
    wyswietl_tablice(tab, n, m);

    // Zwolnienie pamięci
    for (int i = 0; i < n; i++) {
        free(tab[i]);
    }
    free(tab);

    return 0;
}
```

3 Stwórz strukturę Telefon z polami marka (tablica znaków długości 15) oraz liczbaPołączeń (typu int). Następnie, napisz dwie funkcje:

a) initTelefon - funkcja, która przyjmuje dwa argumenty: markę i liczbę połączeń, i zwraca wskaźnik na nowo utworzoną strukturę Telefon z polami ustawionymi na wartości przekazane jako argumenty. Funkcja powinna sprawdzić, czy marka ma długość co najmniej 3 i czy liczba połączeń jest większa niż 50. Jeżeli jeden z tych warunków nie jest spełniony, funkcja powinna zwracać NULL.

b) zwiekszLiczbePolaczen - funkcja, której argumentem jest wskaźnik na strukturę Telefon. Funkcja powinna zwiększyć liczbę połączeń o 10.

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

#define MAX_MARKA_LENGTH 15

// Struktura Telefon
typedef struct {
    char marka[MAX_MARKA_LENGTH];
    int liczbaPolaczen;
} Telefon;

// Funkcja initTelefon
Telefon* initTelefon(const char* marka, int liczbaPolaczen) {
    if (strlen(marka) < 3 || liczbaPolaczen <= 50) {
        return NULL;
    }

    Telefon* telefon = (Telefon*)malloc(sizeof(Telefon));
    strncpy(telefon->marka, marka, MAX_MARKA_LENGTH - 1);
    telefon->marka[MAX_MARKA_LENGTH - 1] = '\0';
    telefon->liczbaPolaczen = liczbaPolaczen;

    return telefon;
}

// Funkcja zwiekszLiczbePolaczen
void zwiekszLiczbePolaczen(Telefon* telefon) {
    telefon->liczbaPolaczen += 10;
}

int main() {
    // Inicjalizacja telefonu
    Telefon* telefon = initTelefon("Samsung", 100);

    if (telefon != NULL) {
        printf("Marka telefonu: %s\n", telefon->marka);
        printf("Liczba polaczen: %d\n", telefon->liczbaPolaczen);


        printf("Po zwiekszeniu liczby polaczen: %d\n", telefon->liczbaPolaczen);
    } else {
        printf("Nieprawidlowe dane telefonu.\n");
    }

    // Zwolnienie pamięci
    free(telefon);

    return 0;
}
```

3 Zdefiniuj strukturę Komputer z polami model (tablica znaków długości 20) oraz czasUzytkowania (typu int). Następnie, stwórz dwie funkcje:

a) initKomputer - funkcja, która przyjmuje dwa argumenty: model i czas użytkowania, i zwraca nowo utworzoną strukturę Komputer (jako wartość, nie wskaźnik) z polami ustawionymi na wartości przekazane jako argumenty. Funkcja powinna sprawdzać, czy model ma długość co najmniej 4 i czy czas użytkowania jest większy niż 100. W przypadku nie spełnienia tych warunków, funkcja powinna strukturę z modelem ustawionym jako "DEFAULT" i czas użytkowania równym 200.

b) zwiekszCzasUzytkowania - funkcja, której argumentem jest wskaźnik na strukturę Komputer. Funkcja powinna zwiększyć czas użytkowania komputera o 100. Wywołaj każdą funkcję co najmniej jeden raz

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

#define MAX_MODEL_LENGTH 20

// Struktura Komputer
typedef struct {
    char model[MAX_MODEL_LENGTH];
    int czasUzytkowania;
} Komputer;

// Funkcja initKomputer
Komputer initKomputer(const char* model, int czasUzytkowania) {
    Komputer komputer;

    if (strlen(model) >= 4 && czasUzytkowania > 100) {
        strncpy(komputer.model, model, MAX_MODEL_LENGTH - 1);
        komputer.model[MAX_MODEL_LENGTH - 1] = '\0';
        komputer.czasUzytkowania = czasUzytkowania;
    } else {
        strcpy(komputer.model, "DEFAULT");
        komputer.czasUzytkowania = 200;
    }

    return komputer;
}

// Funkcja zwiekszCzasUzytkowania
void zwiekszCzasUzytkowania(Komputer* komputer) {
    komputer->czasUzytkowania += 100;
}

int main() {
    // Inicjalizacja komputera
    Komputer komputer = initKomputer("DELL", 150);

    printf("Model komputera: %s\n", komputer.model);
    printf("Czas użytkowania: %d\n", komputer.czasUzytkowania);

    // Zwiększenie czasu użytkowania
    zwiekszCzasUzytkowania(&komputer);

    printf("Po zwiększeniu czasu użytkowania: %d\n", komputer.czasUzytkowania);

    return 0;
}
```

3 Napisz funkcję, której argumentem jest dwuwymiarowa tablica tablic (zawierająca zmienne typu int) oraz jej wymiary n i m. Funkcja ma zwrócić średnią elementów na głównej przekątnej. Stwórz przypadek testowy.

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

double srednia_przekatnej(int** tab, int n, int m) {
    int suma = 0;
    int licznik = 0;

    // Sprawdzamy, czy tablica jest kwadratowa
    if (n != m) {
        printf("Tablica nie jest kwadratowa.\n");
        return 0.0;
    }

    // Obliczamy sume elementów na głównej przekatnej
    for (int i = 0; i < n; i++) {
        suma += tab[i][i];
        licznik++;
    }

    // Obliczamy średnia
    double srednia = (double)suma / licznik;

    return srednia;
}

int main() {
    int n = 3; // liczba wierszy
    int m = 3; // liczba kolumn

    int** tablica = (int**)malloc(n * sizeof(int*));
    for (int i = 0; i < n; i++) {
        tablica[i] = (int*)malloc(m * sizeof(int));
    }
    // Inicjalizacja tablicy
    tablica[0][0] = 2;
    tablica[0][1] = 3;
    tablica[0][2] = -3;
    tablica[1][0] = 1;
    tablica[1][1] = 4;
    tablica[1][2] = 7;
    tablica[2][0] = -3;
    tablica[2][1] = -6;
    tablica[2][2] = 11;

    printf("Tablica:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            printf("%d ", tablica[i][j]);
        }
        printf("\n");
    }
    printf("\n");

    double srednia = srednia_przekatnej(tablica, n, m);
    printf("Średnia elementów na głównej przekatnej: %.2f\n", srednia);

    // Zwolnienie pamieci
    for (int i = 0; i < n; i++) {
        free(tablica[i]);
    }
    free(tablica);

    return 0;
}
```

3 Napisz funkcję, której argumentem jest dwuwymiarowa kwadratowa tablica tablic (zawierająca elementy typu int) i jej wymiar n, $n > 0$. Funkcja ma sumę indeksów najmniejszego elementu w tablicy. W przypadku kilku najmniejszych elementów, ma być to najmniejsza możliwa suma. Stwórz przypadek testowy.

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

void suma_indeksow_min(int** tab, int n) {
    int min = tab[0][0];
    int min_suma_indeksow = 0;

    // Szukamy najmniejszego elementu
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (tab[i][j] < min) {
                min = tab[i][j];
            }
        }
    }

    // Obliczamy sumę indeksów najmniejszego elementu
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (tab[i][j] == min) {
                min_suma_indeksow += (i + j);
            }
        }
    }

    printf("Najmniejszy element: %d\n", min);
    printf("Suma indeksów najmniejszego elementu: %d\n", min_suma_indeksow);
}

int main() {
    int n = 4; // wymiar tablicy

    int** tablica = (int**)malloc(n * sizeof(int*));
    for (int i = 0; i < n; i++) {
        tablica[i] = (int*)malloc(n * sizeof(int));
    }

int main() {
    int n = 4; // wymiar tablicy

    int** tablica = (int**)malloc(n * sizeof(int*));
    for (int i = 0; i < n; i++) {
        tablica[i] = (int*)malloc(n * sizeof(int));
    }

    // Inicjalizacja tablicy
    tablica[0][0] = 2;
    tablica[0][1] = 3;
    tablica[0][2] = -3;
    tablica[0][3] = 4;
    tablica[1][0] = 1;
    tablica[1][1] = 4;
    tablica[1][2] = 7;
    tablica[1][3] = -2;
    tablica[2][0] = -3;
    tablica[2][1] = -6;
    tablica[2][2] = 11;
    tablica[2][3] = 5;
    tablica[3][0] = -2;
    tablica[3][1] = 8;
    tablica[3][2] = 23;
    tablica[3][3] = 0;

    printf("Tablica:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%d ", tablica[i][j]);
        }
        printf("\n");
    }
    printf("\n");

    suma_indeksow_min(tablica, n);


    // Zwolnienie pamięci
    for (int i = 0; i < n; i++) {
        free(tablica[i]);
    }
    free(tablica);

    return 0;
}
```

3 Zaprojektuj strukturę Ksiazka posiadającą dwa pola: tytul (napis) oraz isbn (dowolny typ całkowity). Następnie stwórz funkcję, której argumentami są tablica struktur Ksiazka oraz rozmiar tablicy. Funkcja powinna zwrócić numer isbn książki o najkrótszym tytule (przy kilku takich pozycjach, numer pierwszego). Stwórz przypadek testowy.

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

#define MAX_TITLE_LENGTH 50

struct Ksiazka {
    char tytul[MAX_TITLE_LENGTH];
    int isbn;
};

int znajdzKsiazkeONajkrotszymTytule(struct Ksiazka tablica[], int rozmiar) {
    int indeksNajkrotszego = 0;
    int najkrotszaDlugosc = strlen(tablica[0].tytul);

    for (int i = 1; i < rozmiar; i++) {
        int dlugoscAktualna = strlen(tablica[i].tytul);
        if (dlugoscAktualna < najkrotszaDlugosc) {
            najkrotszaDlugosc = dlugoscAktualna;
            indeksNajkrotszego = i;
        }
    }

    return tablica[indeksNajkrotszego].isbn;
}

int main() {
    struct Ksiazka biblioteka[3];
    strcpy(biblioteka[0].tytul, "Ksiazka1");
    biblioteka[0].isbn = 12345;
    strcpy(biblioteka[1].tytul, "Ksiazka2");
    biblioteka[1].isbn = 67890;
    strcpy(biblioteka[2].tytul, "Ksiazka3");
    biblioteka[2].isbn = 54321;

    int rozmiar = sizeof(biblioteka) / sizeof(biblioteka[0]);

    int numerISBN = znajdzKsiazkeONajkrotszymTytule(biblioteka, rozmiar);

    printf("Numer ISBN ksiazki o najkrotszym tytule: %d\n", numerISBN);

    return 0;
}
```

3 Zaprojektuj funkcję, której argumentem jest dwuwymiarowa kwadratowa tablica tablic (zawierająca elementy typu int) oraz jej wymiar n, $n > 0$. Funkcja powinna zwracać różnicę między sumą indeksów najmniejszego a sumą indeksów największego elementu w tablicy. W przypadku kilku elementów o najmniejszej lub największej wartości, powinny to być najmniejsze możliwe sumy indeksów. Przeprowadź testy na przykładowych danych.

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

#define N 3

int znajdzRozniceSumyIndeksow(int** tablica, int n) {
    int minWartosc = tablica[0][0];
    int maxWartosc = tablica[0][0];
    int minSumaIndeksow = 2 * n;
    int maxSumaIndeksow = 0;

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (tablica[i][j] < minWartosc) {
                minWartosc = tablica[i][j];
                minSumaIndeksow = i + j;
            } else if (tablica[i][j] > maxWartosc) {
                maxWartosc = tablica[i][j];
                maxSumaIndeksow = i + j;
            }
        }
    }

    return maxSumaIndeksow - minSumaIndeksow;
}

int main() {
    int tablica[N][N] = {
        {11, 3, -6},
        {1, 4, 7},
        {-3, -8, 11}
    };

    int** dynamicznaTablica = (int**)malloc(N * sizeof(int*));
    for (int i = 0; i < N; i++) {
        dynamicznaTablica[i] = (int*)malloc(N * sizeof(int));
    }

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            dynamicznaTablica[i][j] = tablica[i][j];
        }
    }

    int roznicaSumyIndeksow = znajdzRozniceSumyIndeksow(dynamicznaTablica, N);

    printf("Roznica sumy indeksow: %d\n", roznicaSumyIndeksow);

    for (int i = 0; i < N; i++) {
        free(dynamicznaTablica[i]);
    }
    free(dynamicznaTablica);

    return 0;
}
```

3 Zdefiniuj strukturę Telefon z polami marka (tablica znaków długości 30) oraz iloscPołączeń (typu int). Stwórz napisz funkcję initTelefon, która przyjmuje dwa argumenty: markę i ilość połączeń, i zwraca nowo utworzoną strukturę Telefon (jako wartość, nie wskaźnik) z polami ustawionymi na wartości przekazane jako argumenty. Funkcja powinna sprawdzać, czy marka ma długość co najmniej 3 i czy ilość połączeń jest większa niż 50. W przypadku nie spełnienia co najmniej jednego z tych warunków, funkcja powinna zwrócić strukturę z marką ustawioną jako "NIEZNANY" i ilością połączeń równą 100. Stwórz dwa przypadki testowe

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

#define MAX_MARKA 30

// Struktura Telefon
struct Telefon {
    char marka[MAX_MARKA];
    int iloscPolaczen;
};

// Funkcja inicjalizujaca strukture Telefon
struct Telefon initTelefon(const char* marka, int iloscPolaczen) {
    struct Telefon telefon;

    // Sprawdzenie warunków
    if (strlen(marka) >= 3 && iloscPolaczen > 50) {
        strcpy(telefon.marka, marka);
        telefon.iloscPolaczen = iloscPolaczen;
    } else {
        strcpy(telefon.marka, "NIEZNANY");
        telefon.iloscPolaczen = 100;
    }

    return telefon;
}

int main() {
    // Przypadki testowe
    struct Telefon telefon1 = initTelefon("Samsung", 80);
    struct Telefon telefon2 = initTelefon("iPhone", 40);

    // Wyświetlenie rezultatów
    printf("Telefon 1:\n");
    printf("Marka: %s\n", telefon1.marka);
    printf("Ilość połączeń: %d\n", telefon1.iloscPolaczen);
    printf("\n");

    printf("Telefon 2:\n");
    printf("Marka: %s\n", telefon2.marka);
    printf("Ilość połączeń: %d\n", telefon2.iloscPolaczen);

    return 0;
}
```

3 Zdefiniuj strukturę Samochod z polami model (tablica znaków długości 25) oraz przebieg (typu int). Napisz funkcję initSamochod, która przyjmuje dwa argumenty: model i przebieg, i zwraca nowo utworzoną strukturę Samochod (jako wartość, nie wskaźnik) z polami ustawionymi na wartości przekazane jako argumenty. Funkcja powinna sprawdzać, czy model rozpoczyna się od dużej litery i czy przebieg jest nie większy niż 500. W przypadku nie spełnienia co najmniej jednego z tych warunków, funkcja powinna zwrócić strukturę z modelem ustawionym jako "DEFAULT" i przebiegiem równym 1000. Stwórz dwa przypadki testowe.

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

#define MAX_MODEL 25

// Struktura Samochod
struct Samochod {
    char model[MAX_MODEL];
    int przebieg;
};

// Funkcja inicjalizująca strukturę Samochod
struct Samochod initSamochod(const char* model, int przebieg) {
    struct Samochod samochod;

    // Sprawdzenie warunków
    if (isupper(model[0]) && przebieg <= 500) {
        strcpy(samochod.model, model);
        samochod.przebieg = przebieg;
    } else {
        strcpy(samochod.model, "DEFAULT");
        samochod.przebieg = 1000;
    }

    return samochod;
}

int main() {
    // Przypadki testowe
    struct Samochod samochod1 = initSamochod("BMW", 300);
    struct Samochod samochod2 = initSamochod("ford", 600);

    // Wyświetlenie rezultatów
    printf("Samochod 1:\n");
    printf("Model: %s\n", samochod1.model);
    printf("Przebieg: %d\n", samochod1.przebieg);
    printf("\n");

    printf("Samochod 2:\n");
    printf("Model: %s\n", samochod2.model);
    printf("Przebieg: %d\n", samochod2.przebieg);

    return 0;
}
```

Zadania 4

Zad.4. Napisz funkcję, która otrzymuje jako argument listę bez głowy o elementach typu:

```
struct element {
    float value;
    struct element * next;
};
```

Funkcja ma zamienić wartości każdego elementu na jego wartość bezwzględną. Dla listy pustej funkcja ma nic nie robić. Stwórz przypadek testowy.

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

struct element {
    float value;
    struct element* next;
};

void wypiszListeBezwzgledne(struct element* lista) {
    struct element* curr = lista;

    while (curr != NULL) {
        printf("%.2f ", fabs(curr->value));
        curr = curr->next;
    }

    printf("\n");
}

void usunListe(struct element* lista) {
    struct element* curr = lista;
    struct element* temp;

    while (curr != NULL) {
        temp = curr->next;
        free(curr);
        curr = temp;
    }
}

int main() {
    // Tworzenie przykładowej listy
    struct element* lista = (struct element*)malloc(sizeof(struct element));
    struct element* elem1 = (struct element*)malloc(sizeof(struct element));
    struct element* elem2 = (struct element*)malloc(sizeof(struct element));
    struct element* elem3 = (struct element*)malloc(sizeof(struct element));

    lista->value = -2.5;
    lista->next = elem1;

    elem1->value = 3.7;
    elem1->next = elem2;

    elem2->value = -1.2;
    elem2->next = elem3;

    elem3->value = 4.9;
    elem3->next = NULL;

    // Wyświetlanie wartości bezwzględnych elementów listy
    printf("Wartości bezwzględne: ");
    wypiszListeBezwzgledne(lista);

    // Usuwanie listy
    usunListe(lista);

    return 0;
}
```

Zad.4. Napisz funkcję, która otrzymuje jako argument listę z głową o elementach typu:

```
struct node {
    int val;
    struct node * next;
};
```

Funkcja ma zamienić wartości każdego elementu na jego wartość bezwzględną. Dla listy pustej funkcja ma nic nie robić. Stwórz przypadek testowy.

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

struct node {
    int val;
    struct node* next;
};

void zamienNaBezwzgledne(struct node* head) {
    struct node* curr = head->next;

    while (curr != NULL) {
        curr->val = abs(curr->val);
        curr = curr->next;
    }
}

void wypiszListe(struct node* head) {
    struct node* curr = head->next;

    while (curr != NULL) {
        printf("%d ", curr->val);
        curr = curr->next;
    }

    printf("\n");
}

void dodajElement(struct node* head, int value) {
    struct node* newNode = (struct node*)malloc(sizeof(struct node));
    newNode->val = value;
    newNode->next = head->next;
    head->next = newNode;
}

void usunListe(struct node* head) {
    struct node* curr = head->next;
    struct node* temp;

    while (curr != NULL) {
        temp = curr->next;
        free(curr);
        curr = temp;
    }

    head->next = NULL;
}

int main() {
    // Inicjalizacja listy z głową
    struct node* head = (struct node*)malloc(sizeof(struct node));
    head->next = NULL;

    // Dodawanie elementów do listy
    dodajElement(head, -5);
    dodajElement(head, 3);
    dodajElement(head, -2);
    dodajElement(head, 7);

    // Wyświetlanie początkowej listy
    printf("Początkowa lista: ");
    wypiszListe(head);

    // Zamiana wartości na wartości bezwzględne
    zamienNaBezwzgledne(head);

    // Wyświetlanie listy po zamianie
    printf("Lista po zamianie na wartości bezwzględne: ");
    wypiszListe(head);

    // Usuwanie listy
    usunListe(head);
    free(head);

    return 0;
}
```

Zad.4. Napisz funkcję, która przyjmuje jako argument listę z głową o elementach typu:

```
struct node {
    int x;
    struct node * next;
};
```

oraz liczbę całkowitą `d`. Funkcja ma zwrócić ile na liście jest elementów równych `d`. Stwórz jeden przypadek testowy.

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

struct node {
    int x;
    struct node* next;
};

int liczElementy(struct node* head, int d) {
    int count = 0;
    struct node* curr = head->next;

    while (curr != NULL) {
        if (curr->x == d) {
            count++;
        }
        curr = curr->next;
    }

    return count;
}

void dodajElement(struct node* head, int value) {
    struct node* newNode = (struct node*)malloc(sizeof(struct node));
    newNode->x = value;
    newNode->next = head->next;
    head->next = newNode;
}

void usunListe(struct node* head) {
    struct node* curr = head->next;
    struct node* temp;

    while (curr != NULL) {
        temp = curr->next;
        free(curr);
        curr = temp;
    }

    head->next = NULL;

}

int main() {
    // Inicjalizacja listy z głową
    struct node* head = (struct node*)malloc(sizeof(struct node));
    head->next = NULL;

    // Dodawanie elementów do listy
    dodajElement(head, 2);
    dodajElement(head, 4);
    dodajElement(head, 6);
    dodajElement(head, 2);
    dodajElement(head, 8);

    // Wyświetlanie listy
    printf("Lista: ");
    struct node* curr = head->next;
    while (curr != NULL) {
        printf("%d ", curr->x);
        curr = curr->next;
    }
    printf("\n");

    // Liczenie wystąpień elementów równych 2
    int d = 2;
    int count = liczElementy(head, d);

    // Wyświetlanie wyniku
    printf("Liczba wystąpień elementów równych %d: %d\n", d, count);

    // Usuwanie listy
    usunListe(head);
    free(head);

    return 0;
}
```

Zad.4. Napisz funkcję, która przyjmuje jako argument listę z głową o elementach typu:

```
struct node {
    int x;
    struct node * next;
};
```

Funkcja ma zwrócić sumę elementów nieparzystych z listy. Stwórz jeden przypadek testowy.

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

struct node {
    int x;
    struct node* next;
};

int sumaNieparzystych(struct node* head) {
    int sum = 0;
    struct node* curr = head->next;

    while (curr != NULL) {
        if (curr->x % 2 != 0) {
            sum += curr->x;
        }
        curr = curr->next;
    }

    return sum;
}

void dodajElement(struct node* head, int value) {
    struct node* newNode = (struct node*)malloc(sizeof(struct node));
    newNode->x = value;
    newNode->next = head->next;
    head->next = newNode;
}

void usunListe(struct node* head) {
    struct node* curr = head->next;
    struct node* temp;

    while (curr != NULL) {
        temp = curr->next;
        free(curr);
        curr = temp;
    }

    head->next = NULL;

}

int main() {
    // Inicjalizacja listy z głową
    struct node* head = (struct node*)malloc(sizeof(struct node));
    head->next = NULL;

    // Dodawanie elementów do listy
    dodajElement(head, 2);
    dodajElement(head, 3);
    dodajElement(head, 6);
    dodajElement(head, 5);
    dodajElement(head, 8);

    // Wyświetlanie listy
    printf("Lista: ");
    struct node* curr = head->next;
    while (curr != NULL) {
        printf("%d ", curr->x);
        curr = curr->next;
    }
    printf("\n");

    // Obliczanie sumy elementów nieparzystych
    int sum = sumaNieparzystych(head);

    // Wyświetlanie wyniku
    printf("Suma elementów nieparzystych: %d\n", sum);

    // Usuwanie listy
    usunListe(head);
    free(head);

    return 0;
}
```

Zad.4. Napisz funkcję, która przyjmuje jako argument listę bez głowy o elementach typu:

```
struct elem {
    int x;
    struct elem * next;
};
```

Funkcja ma podwoić wszystkie elementy dodatnie na liście (o ile istnieją). Stwórz jeden przypadek testowy.

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

struct elem {
    int x;
    struct elem* next;
};

void podwojDodatnie(struct elem* head) {
    struct elem* curr = head->next;

    while (curr != NULL) {
        if (curr->x > 0) {
            struct elem* newNode = (struct elem*)malloc(sizeof(struct elem));
            newNode->x = curr->x;
            newNode->next = curr->next;
            curr->next = newNode;

            curr = newNode->next; // Przesuwamy się do następnego elementu, który nie jest podwojony
        } else {
            curr = curr->next;
        }
    }
}

void dodajElement(struct elem* head, int value) {
    struct elem* newNode = (struct elem*)malloc(sizeof(struct elem));
    newNode->x = value;
    newNode->next = head->next;
    head->next = newNode;
}

void usunListe(struct elem* head) {
    struct elem* curr = head->next;
    struct elem* temp;

    while (curr != NULL) {
        temp = curr->next;
        free(curr);
        curr = temp;
    }

    head->next = NULL;
}

void wyswietlListe(struct elem* head) {
    struct elem* curr = head->next;

    printf("Lista: ");
    while (curr != NULL) {
        printf("%d ", curr->x);
        curr = curr->next;
    }
    printf("\n");
}

int main() {
    // Inicjalizacja listy bez głowy
    struct elem* head = (struct elem*)malloc(sizeof(struct elem));
    head->next = NULL;

    // Dodawanie elementów do listy
    dodajElement(head, -2);
    dodajElement(head, 3);
    dodajElement(head, -6);
    dodajElement(head, 5);
    dodajElement(head, 8);
    dodajElement(head, 10);
    dodajElement(head, -8);

    // Wyświetlanie listy przed podwojeniem
    printf("Przed podwojeniem: ");
    wyswietlListe(head);

    // Podwojenie dodatnich elementów
    podwojDodatnie(head);

    // Wyświetlanie listy po podwojeniu
    printf("Po podwojeniu: ");
    wyswietlListe(head);

    // Usuwanie listy
    usunListe(head);
    free(head);

    return 0;
}
```

Zad.4. Napisz funkcję, która przyjmuje jako argument listę z głową o elementach typu:

```
struct node {
    int y;
    struct node * next;
};
```

oraz dwie liczby całkowite `a` i `b`. Funkcja ma dodać na początek listy dwa nowe elementy i ich wartości ustawić odpowiednio z podanych argumentów. Stwórz jeden przypadek testowy.

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

struct node {
    int y;
    struct node* next;
};

void dodajElementy(struct node* head, int a, int b) {
    struct node* newNode1 = (struct node*)malloc(sizeof(struct node));
    newNode1->y = a;
    newNode1->next = head->next;
    head->next = newNode1;

    struct node* newNode2 = (struct node*)malloc(sizeof(struct node));
    newNode2->y = b;
    newNode2->next = head->next;
    head->next = newNode2;
}

void dodajElement(struct node* head, int value) {
    struct node* newNode = (struct node*)malloc(sizeof(struct node));
    newNode->y = value;
    newNode->next = head->next;
    head->next = newNode;
}

void usunListe(struct node* head) {
    struct node* curr = head->next;
    struct node* temp;

    while (curr != NULL) {
        temp = curr->next;
        free(curr);
        curr = temp;
    }

    head->next = NULL;
}

void wyswietlListe(struct node* head) {
    struct node* curr = head->next;

    printf("Lista: ");
    while (curr != NULL) {
        printf("%d ", curr->y);
        curr = curr->next;
    }
    printf("\n");
}

int main() {
    // Inicjalizacja listy z głową
    struct node* head = (struct node*)malloc(sizeof(struct node));
    head->next = NULL;

    // Dodawanie elementów do listy
    dodajElement(head, 5);
    dodajElement(head, 3);
    dodajElement(head, 1);

    // Wyświetlanie listy przed dodaniem nowych elementów
    printf("Przed dodaniem: ");
    wyswietlListe(head);

    // Dodawanie nowych elementów na początek listy
    dodajElementy(head, 10, 7);

    // Wyświetlanie listy po dodaniu nowych elementów
    printf("Po dodaniu: ");
    wyswietlListe(head);

    // Usuwanie listy
    usunListe(head);
    free(head);

    return 0;
}
```

Zad.4. Napisz funkcję, która przyjmuje jako argument listę bez głowy o elementach typu:

```
struct node {
    int a;
    struct node * next;
};
```

oraz dwie liczby całkowite `a` i `b`. Funkcja ma dodać na początek listy dwa nowe elementy i ich wartości ustawić odpowiednio z podanych argumentów. Stwórz jeden przypadek testowy.

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

struct node {
    int a;
    struct node* next;
};

void dodajElementy(struct node** head, int a, int b) {
    struct node* newNode1 = (struct node*)malloc(sizeof(struct node));
    newNode1->a = a;
    newNode1->next = *head;
    *head = newNode1;

    struct node* newNode2 = (struct node*)malloc(sizeof(struct node));
    newNode2->a = b;
    newNode2->next = *head;
    *head = newNode2;
}

void wyswietlListe(struct node* head) {
    struct node* curr = head;

    printf("Lista: ");
    while (curr != NULL) {
        printf("%d ", curr->a);
        curr = curr->next;
    }
    printf("\n");
}

void zwolnijListe(struct node* head) {
    struct node* curr = head;
    struct node* temp;

    while (curr != NULL) {
        temp = curr->next;
        free(curr);
        curr = temp;
    }

}

int main() {
    // Inicjalizacja listy bez głowy
    struct node* head = NULL;

    // Dodawanie elementów do listy
    dodajElementy(&head, 5, 10);
    dodajElementy(&head, 3, 8);

    // Wyświetlanie listy
    wyswietlListe(head);

    // Zwolnienie pamięci zajmowanej przez listę
    zwolnijListe(head);

    return 0;
}
```

Zad.4. Napisz funkcję, która porównuje dwie listy z głową o elementach typu:

```
struct node {
    int i;
    struct node * next;
};
```

i zwraca 1 jeśli ostatnie elementy na liście są równe oraz 0 w pozostałych przypadkach (także wtedy gdy któraś z list lub obie są puste). Stwórz jeden przypadek testowy.

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

struct node {
    int i;
    struct node* next;
};

int porownajListy(struct node* head1, struct node* head2) {
    if (head1 == NULL && head2 == NULL) {
        // Obie listy są puste
        return 0;
    }

    struct node* curr1 = head1;
    struct node* curr2 = head2;

    // Przesunięcie na ostatni element pierwszej listy
    while (curr1->next != NULL) {
        curr1 = curr1->next;
    }

    // Przesunięcie na ostatni element drugiej listy
    while (curr2->next != NULL) {
        curr2 = curr2->next;
    }

    // Porównanie ostatnich elementów
    if (curr1->i == curr2->i) {
        return 1;
    } else {
        return 0;
    }
}

int main() {
    // Inicjalizacja listy 1
    struct node* head1 = NULL;
    struct node* elem1 = (struct node*)malloc(sizeof(struct node));
    elem1->i = 5;
    elem1->next = NULL;
    head1 = elem1;

    // Inicjalizacja listy 2
    struct node* head2 = NULL;
    struct node* elem2 = (struct node*)malloc(sizeof(struct node));
    elem2->i = 10;
    elem2->next = NULL;
    head2 = elem2;

    // Porównanie list
    int wynik = porownajListy(head1, head2);

    // Wyświetlenie wyniku
    printf("Wynik: %d\n", wynik);

    // Zwolnienie pamięci zajmowanej przez listy
    free(elem1);
    free(elem2);

    return 0;
}
```

Zad.4. Napisz funkcję, która porównuje dwie listy bez głowy o elementach typu:

```
struct node {
    int a;
    struct node * next;
};
```

i zwraca 1 jeśli listy są takiej samej długości oraz 0 w pozostałych przypadkach (także wtedy, gdy któraś z list lub obie są puste). Stwórz przypadek testowy.

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

struct node {
    int a;
    struct node* next;
};

int porownajListy(struct node* head1, struct node* head2) {
    struct node* temp1 = head1;
    struct node* temp2 = head2;

    // Liczenie długości listy 1
    int count1 = 0;
    while (temp1 != NULL) {
        count1++;
        temp1 = temp1->next;
    }

    // Liczenie długości listy 2
    int count2 = 0;
    while (temp2 != NULL) {
        count2++;
        temp2 = temp2->next;
    }

    // Porównanie długości list
    if (count1 == count2) {
        return 1;
    } else {
        return 0;
    }
}

int main() {
    // Inicjalizacja listy 1
    struct node* head1 = NULL;
    struct node* elem1 = (struct node*)malloc(sizeof(struct node));
    elem1->a = 5;
    elem1->next = NULL;

    head1 = elem1;

    // Inicjalizacja listy 2
    struct node* head2 = NULL;
    struct node* elem2 = (struct node*)malloc(sizeof(struct node));
    elem2->a = 10;
    elem2->next = NULL;
    head2 = elem2;

    // Dodanie dodatkowego elementu do listy 2, wtedy wynik da 0, żeby było 1 trzeba usunąć ten element
    struct node* elem3 = (struct node*)malloc(sizeof(struct node));
    elem3->a = 15;
    elem3->next = NULL;
    elem2->next = elem3;

    // Porównanie długości list
    int wynik = porownajListy(head1, head2);

    // Wyświetlenie wyniku
    printf("Wynik: %d\n", wynik);

    // Zwolnienie pamięci zajmowanej przez listy
    free(elem1);
    free(elem2);
    free(elem3);

    return 0;
}
```

Zad.4. Napisz funkcję, która porównuje dwie listy z głową o elementach typu:

```
struct element {
    int i;
    struct element * next;
};
```

i zwraca 1 jeśli listy są takiej samej długości oraz 0 w pozostałych przypadkach (także wtedy, gdy któraś z list lub obie są puste). Stwórz przypadek testowy.

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

struct element {
    int i;
    struct element* next;
};

int porownajListy(struct element* head1, struct element* head2) {
    struct element* temp1 = head1;
    struct element* temp2 = head2;

    // Liczenie długości listy 1
    int count1 = 0;
    while (temp1 != NULL) {
        count1++;
        temp1 = temp1->next;
    }

    // Liczenie długości listy 2
    int count2 = 0;
    while (temp2 != NULL) {
        count2++;
        temp2 = temp2->next;
    }

    // Porównanie długości list
    if (count1 == count2) {
        return 1;
    } else {
        return 0;
    }
}

int main() {
    // Inicjalizacja listy 1
    struct element* head1 = NULL;
    struct element* elem1 = (struct element*)malloc(sizeof(struct element));
    elem1->i = 5;
    elem1->next = NULL;
    head1 = elem1;

    // Inicjalizacja listy 2
    struct element* head2 = NULL;
    struct element* elem2 = (struct element*)malloc(sizeof(struct element));
    elem2->i = 10;
    elem2->next = NULL;
    head2 = elem2;

    // Porównanie długości list
    int wynik = porownajListy(head1, head2);

    // Wyświetlenie wyniku
    printf("Wynik: %d\n", wynik);

    // Zwolnienie pamięci zajmowanej przez listy
    free(elem1);
    free(elem2);

    return 0;
}
```

Zad.4. Napisz funkcję, która przyjmuje dwie listy z głową o elementach typu:

```
struct node {
    int value;
    struct node * next;
};
```

i zwraca 1 jeśli suma wszystkich elementów nieparzystych w obu listach jest taka sama oraz 0 w przeciwnym przypadku (także wtedy, gdy któraś z list lub obie są puste). Stwórz przypadek testowy.

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

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

int porownajSumyNieparzyste(struct node* head1, struct node* head2) {
    int sum1 = 0;
    int sum2 = 0;

    struct node* temp1 = head1;
    struct node* temp2 = head2;

    // Obliczanie sumy elementów nieparzystych w liście 1
    while (temp1 != NULL) {
        if (temp1->value % 2 != 0) {
            sum1 += temp1->value;
        }
        temp1 = temp1->next;
    }

    // Obliczanie sumy elementów nieparzystych w liście 2
    while (temp2 != NULL) {
        if (temp2->value % 2 != 0) {
            sum2 += temp2->value;
        }
        temp2 = temp2->next;
    }

    // Porównanie sum
    if (sum1 == sum2) {
        return 1;
    } else {
        return 0;
    }
}

int main() {
    // Inicjalizacja listy 1
    struct node* head1 = NULL;
    struct node* elem1 = (struct node*)malloc(sizeof(struct node));
    elem1->value = 3;
    elem1->next = NULL;
    head1 = elem1;

    // Inicjalizacja listy 2
    struct node* head2 = NULL;
    struct node* elem2 = (struct node*)malloc(sizeof(struct node));
    elem2->value = 5;
    elem2->next = NULL;
    head2 = elem2;

    // Porównanie sum elementów nieparzystych
    int wynik = porownajSumyNieparzyste(head1, head2);

    // Wyświetlenie wyniku
    printf("Wynik: %d\n", wynik);

    // Zwolnienie pamięci zajmowanej przez listy
    free(elem1);
    free(elem2);

    return 0;
}
```

Zad.4. Napisz funkcję, która przyjmuje dwie listy bez głowy o elementach typu:

```
struct node {
    int w;
    struct node * next;
};
```

i zwraca 1 jeśli suma wszystkich elementów parzystych w obu listach jest taka sama oraz 0 w przeciwnym przypadku (także wtedy, gdy któraś z list lub obie są puste). Stwórz przypadek testowy.

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

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

int porownajSumyNieparzyste(struct node* head1, struct node* head2) {
    int sum1 = 0;
    int sum2 = 0;

    struct node* temp1 = head1;
    struct node* temp2 = head2;

    // Obliczanie sumy elementów nieparzystych w liście 1
    while (temp1 != NULL) {
        if (temp1->value % 2 != 0) {
            sum1 += temp1->value;
        }
        temp1 = temp1->next;
    }

    // Obliczanie sumy elementów nieparzystych w liście 2
    while (temp2 != NULL) {
        if (temp2->value % 2 != 0) {
            sum2 += temp2->value;
        }
        temp2 = temp2->next;
    }

    // Porównanie sum
    if (sum1 == sum2) {
        return 1;
    } else {
        return 0;
    }
}

int main() {
    // Inicjalizacja listy 1
    struct node* head1 = NULL;
    struct node* elem1 = (struct node*)malloc(sizeof(struct node));
    elem1->value = 3;
    elem1->next = NULL;
    head1 = elem1;

    // Inicjalizacja listy 2
    struct node* head2 = NULL;
    struct node* elem2 = (struct node*)malloc(sizeof(struct node));
    elem2->value = 5;
    elem2->next = NULL;
    head2 = elem2;

    // Porównanie sum elementów nieparzystych
    int wynik = porownajSumyNieparzyste(head1, head2);

    // Wyświetlenie wyniku
    printf("Wynik: %d\n", wynik);

    // Zwolnienie pamięci zajmowanej przez listy
    free(elem1);
    free(elem2);

    return 0;
}
```

Zad.4. Napisz funkcję, która otrzymuje jako argument listę bez głowy o elementach typu:

```
struct node {
    int i;
    struct node * next;
};
```

Funkcja ma wyświetlić na konsoli w kolejnych wierszach wartości elementów na liście będących kwadratami liczb całkowitych. Stwórz przypadek testowy.

Przykład. Jeśli lista składa się z elementów 4,5,6,-34,0,25,11, to ma być wyświetlone w kolejnych wierszach: 4,0,25.

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

struct node {
    int i;
    struct node* next;
};

void wyswietlKwadraty(struct node* head) {
    struct node* current = head;

    while (current != NULL) {
        int value = current->i;

        // Sprawdzenie czy wartość jest kwadratem liczby całkowitej
        int sqrt_value = sqrt(value);
        if (sqrt_value * sqrt_value == value) {
            printf("%d\n", value);
        }

        current = current->next;
    }
}

int main() {
    // Inicjalizacja listy
    struct node* head = NULL;
    struct node* elem1 = (struct node*)malloc(sizeof(struct node));
    struct node* elem2 = (struct node*)malloc(sizeof(struct node));
    struct node* elem3 = (struct node*)malloc(sizeof(struct node));
    struct node* elem4 = (struct node*)malloc(sizeof(struct node));
    struct node* elem5 = (struct node*)malloc(sizeof(struct node));
    struct node* elem6 = (struct node*)malloc(sizeof(struct node));
    struct node* elem7 = (struct node*)malloc(sizeof(struct node));


    elem1->i = 4;
    elem1->next = elem2;
    elem2->i = 5;
    elem2->next = elem3;
    elem3->i = 6;
    elem3->next = elem4;
    elem4->i = -34;
    elem4->next = elem5;
    elem5->i = 0;
    elem5->next = elem6;
    elem6->i = 25;
    elem6->next = elem7;
    elem7->i = 11;
    elem7->next = NULL;

    head = elem1;

    // Wyświetlanie kwadratów liczb całkowitych
    wyswietlKwadraty(head);

    // Zwolnienie pamięci zajmowanej przez listę
    free(elem1);
    free(elem2);
    free(elem3);
    free(elem4);
    free(elem5);
    free(elem6);
    free(elem7);

    return 0;
}
```

Zad.4. Napisz funkcję, która przyjmuje jako argument listę z głową o elementach typu:

```
struct node {
    float value;
    struct node * next;
};
```

i zwraca największą z liczb znajdujących się na liście. W przypadku pustej listy, funkcja ma zwrócić zero. Stwórz jeden przypadek testowy.

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

struct node {
    float value;
    struct node* next;
};

float findMax(struct node* head) {
    if (head == NULL) {
        return 0; // Pusta lista, zwracamy zero
    }

    float max = head->value;
    struct node* current = head->next;

    while (current != NULL) {
        if (current->value > max) {
            max = current->value;
        }
        current = current->next;
    }

    return max;
}

int main() {
    // Tworzenie przykładowej listy: 1.5 -> 3.2 -> 2.8 -> NULL
    struct node* head = (struct node*)malloc(sizeof(struct node));
    struct node* second = (struct node*)malloc(sizeof(struct node));
    struct node* third = (struct node*)malloc(sizeof(struct node));

    head->value = 1.5;
    head->next = second;
    second->value = 3.2;
    second->next = third;
    third->value = 2.8;
    third->next = NULL;

    // Wywołanie funkcji findMax i wyświetlenie wyniku
    float max = findMax(head);
    printf("Najwieksza liczba na liscie: %.2f\n", max);

    // Zwolnienie pamięci
    free(head);
    free(second);
    free(third);

    return 0;
}
```

Zad.4. Napisz funkcję, która przyjmuje jako argument listę bez głowy o elementach typu:

```
struct node {
    double x;
    struct node * next;
};
```

i zwraca najmniejszą z liczb znajdujących się na liście. W przypadku pustej listy, funkcja ma zwrócić zero. Stwórz jeden przypadek testowy.

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

struct node {
    double x;
    struct node* next;
};

double findMin(struct node* head) {
    if (head == NULL) {
        return 0; // Pusta lista, zwracamy zero
    }

    double min = head->x;
    struct node* current = head->next;

    while (current != NULL) {
        if (current->x < min) {
            min = current->x;
        }
        current = current->next;
    }

    return min;
}

int main() {
    // Tworzenie przykładowej listy: 1.2 -> 2.8 -> 0.5 -> NULL
    struct node* head = (struct node*)malloc(sizeof(struct node));
    struct node* second = (struct node*)malloc(sizeof(struct node));
    struct node* third = (struct node*)malloc(sizeof(struct node));

    head->x = 1.2;
    head->next = second;
    second->x = 2.8;
    second->next = third;
    third->x = 0.5;
    third->next = NULL;

    // Wywołanie funkcji findMin i wyświetlenie wyniku
    double min = findMin(head);
    printf("Najmniejsza liczba na liscie: %.2f\n", min);

    // Zwolnienie pamięci
    free(head);
    free(second);
    free(third);

    return 0;
}
```

Zad.4. Napisz funkcję, która otrzymuje jako argument listę bez głowy o elementach typu:

```
struct node {
    int i;
    struct node * next;
};
```

Funkcja ma zdublować wartość ostatniego elementu, o ile lista jest nie pusta. W przypadku pustej listy, funkcja ma nic nie robić. Stwórz przypadek testowy.

Przykład: Dla listy 3,4,5 ma być zdublowana 5, więc po modyfikacji lista ma być postaci 3,4,5,5.

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

struct node {
    int i;
    struct node* next;
};

void duplicateLastElement(struct node* head) {
    if (head == NULL) {
        return; // Pusta lista, nic nie robimy
    }

    struct node* current = head;

    while (current->next != NULL) {
        current = current->next;
    }

    // Utworzenie nowego węzła z zdublowaną wartością
    struct node* new_node = (struct node*)malloc(sizeof(struct node));
    new_node->i = current->i;
    new_node->next = NULL;

    // Podłączenie nowego węzła na końcu listy
    current->next = new_node;
}

int main() {
    // Tworzenie przykładowej listy: 3 -> 4 -> 5 -> NULL
    struct node* head = (struct node*)malloc(sizeof(struct node));
    struct node* second = (struct node*)malloc(sizeof(struct node));
    struct node* third = (struct node*)malloc(sizeof(struct node));

    head->i = 3;
    head->next = second;
    second->i = 4;
    second->next = third;
    third->i = 5;
    third->next = NULL;

    // Wywołanie funkcji duplicateLastElement
    duplicateLastElement(head);

    // Wyświetlenie listy po modyfikacji
    struct node* current = head;
    while (current != NULL) {
        printf("%d ", current->i);
        current = current->next;
    }
    printf("\n");

    // Zwolnienie pamięci
    struct node* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }

    return 0;
}
```

Zad.4. Napisz funkcję, która otrzymuje jako argument listę z głową o elementach typu:

```
struct elem {
    int a;
    struct elem * next;
};
```

Funkcja ma zdublować wartość ostatniego elementu, o ile lista jest nie pusta. W przypadku pustej listy, funkcja ma nic nie robić. Stwórz przypadek testowy.

Przykład: Dla listy 3,4,5 ma być zdublowana 5, więc po modyfikacji lista ma być postaci 3,4,5,5.

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

struct elem {
    int a;
    struct elem* next;
};

void duplicateLastElement(struct elem* head) {
    if (head == NULL) {
        return; // Pusta lista, nic nie robimy
    }

    struct elem* current = head;

    while (current->next != NULL) {
        current = current->next;
    }

    // Utworzenie nowego węzła z zdublowaną wartością
    struct elem* new_node = (struct elem*)malloc(sizeof(struct elem));
    new_node->a = current->a;
    new_node->next = NULL;

    // Podłączenie nowego węzła na końcu listy
    current->next = new_node;
}

int main() {
    // Tworzenie przykładowej listy: 3 -> 4 -> 5 -> NULL
    struct elem* head = (struct elem*)malloc(sizeof(struct elem));
    struct elem* second = (struct elem*)malloc(sizeof(struct elem));
    struct elem* third = (struct elem*)malloc(sizeof(struct elem));

    head->a = 3;
    head->next = second;
    second->a = 4;
    second->next = third;
    third->a = 5;
    third->next = NULL;

    // Wywołanie funkcji duplicateLastElement
    duplicateLastElement(head);

    // Wyświetlenie listy po modyfikacji
    struct elem* current = head;
    while (current != NULL) {
        printf("%d ", current->a);
        current = current->next;
    }
    printf("\n");

    // Zwolnienie pamięci
    struct elem* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }

    return 0;
}
```

Zadania 1

Dane są następujące wyrazy i znaki: int char char * * * ( ) napis1 napis2 , foo Ułóż je we właściwej kolejności, aby otrzymać nagłówek funkcji foo, która dostaje jako argumenty dwa napisy oraz zwraca wskaźnik na int. Następnie dodaj dowolną implementację funkcji i stwórz dla niej przypadek testowy.

```c
#include <stdio.h>

int *foo(char *napis1, char *napis2);

int main() {
    char *napis1 = "Hello";
    char *napis2 = "World";

    int *result = foo(napis1, napis2);
    printf("Wynik: %d\n", *result);

    return 0;
}

int *foo(char *napis1, char *napis2) {
    printf("Pierwszy napis: %s\n", napis1);
    printf("Drugi napis: %s\n", napis2);

    // Przykładowa implementacja
    int *result = (int *)malloc(sizeof(int));
    *result = 42;

    return result;
}
```

1 Dane są następujące wyrazy i znaki: float float int () [ ] [ ] 10 , tab a fun Ułóż je we właściwej kolejności, aby otrzymać nagłówek funkcji fun, która dostaje jako argumenty dwuwymiarową tablicę elementów wymiaru 10x10 oraz liczbę całkowitą. Następnie dodaj dowolną implementację funkcji i stwórz dla niej przypadek testowy.

```c
#include <stdio.h>

void fun(float tab[][10], int num);

int main() {
    float tab[10][10] = {
        {1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8, 9.9, 10.10},
        {11.11, 12.12, 13.13, 14.14, 15.15, 16.16, 17.17, 18.18, 19.19, 20.20},
        // reszta wierszy tablicy...
    };
    int num = 42;

    fun(tab, num);

    return 0;
}

void fun(float tab[][10], int num) {
    printf("Liczba: %d\n", num);

    printf("Tablica:\n");
    for (int i = 0; i < 10; i++) {
        for (int j = 0; j < 10; j++) {
            printf("%.2f ", tab[i][j]);
        }
        printf("\n");
    }

    // Przykładowa implementacja
    // ...

    return;
}
```

1 Dane są następujące wyrazy i znaki: char void int int foo a b tab a [ ] [ ] ( ) , , * Ułóż je we właściwej kolejności (zachowując krotność), aby otrzymać nagłówek funkcji foo, która dostaje jako argumenty napis, liczbę całkowitą oraz dwuwymiarową tablicę elementów. Następnie dodaj dowolną implementację funkcji i stwórz dla niej przypadek testowy.

```c
#include <stdio.h>

void foo(char* str, int num, int arr[][10]);

int main() {
    char str[] = "Hello, world!";
    int num = 42;
    int arr[5][10] = {
        {1, 2, 3, 4, 5, 6, 7, 8, 9, 10},
        {11, 12, 13, 14, 15, 16, 17, 18, 19, 20},
        // reszta wierszy tablicy...
    };

    foo(str, num, arr);

    return 0;
}

void foo(char* str, int num, int arr[][10]) {
    printf("Napis: %s\n", str);
    printf("Liczba: %d\n", num);

    printf("Tablica:\n");
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 10; j++) {
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }

    // Przykładowa implementacja
    // ...

    return;
}
```

1 Dane są następujące wyrazy i znaki: int char float sum a b ( ) * , * Ułóż je we właściwej kolejności (zachowując krotność), aby otrzymać nagłówek funkcji sum, która dostaje jako argumenty napis i wskaźnik na liczbę zmiennoprzecinkową. Następnie dodaj dowolną implementację funkcji i stwórz dla niej przypadek testowy.

```c
#include <stdio.h>

float sum(char* str, float* num);

int main() {
    char str[] = "Hello";
    float num = 3.14;

    float result = sum(str, &num);
    printf("Wynik: %.2f\n", result);

    return 0;
}

float sum(char* str, float* num) {
    printf("Napis: %s\n", str);
    printf("Liczba: %.2f\n", *num);

    // Przykładowa implementacja
    float sum = 0.0;
    for (int i = 0; str[i] != '\0'; i++) {
        sum += str[i];
    }

    *num += sum;

    return sum;
}
```

1 Dane są następujące wyrazy i znaki: int char char * * * ( ) napis1 napis2 , foo Ułóż je we właściwej kolejności, aby otrzymać nagłówek funkcji foo, która dostaje jako argumenty dwa napisy oraz zwraca wskaźnik na int. Następnie dodaj dowolną implementację funkcji i stwórz dla niej przypadek testowy.

```c
#include <stdio.h>

int* foo(char* str1, char* str2);

int main() {
    char str1[] = "Hello";
    char str2[] = "World";

    int* result = foo(str1, str2);
    printf("Wynik: %d\n", *result);

    return 0;
}

int* foo(char* str1, char* str2) {
    printf("Napis 1: %s\n", str1);
    printf("Napis 2: %s\n", str2);

    // Przykładowa implementacja
    int* ptr = (int*)malloc(sizeof(int));
    *ptr = 10;

    return ptr;
}
```

1 Dane są następujące wyrazy i znaki: void int int const const ( ) * * , fun a b Ułóż je we właściwej kolejności, aby otrzymać nagłówek funkcji fun, której argumentami są dwa wskaźniki na stałą wartość. Następnie dodaj dowolną implementację funkcji i stwórz dla niej przypadek testowy.

```c
#include <stdio.h>

void fun(const int* ptr1, const int* ptr2);

int main() {
    int a = 5;
    int b = 10;

    fun(&a, &b);

    return 0;
}

void fun(const int* ptr1, const int* ptr2) {
    printf("Wartość 1: %d\n", *ptr1);
    printf("Wartość 2: %d\n", *ptr2);
    // Przykładowa implementacja funkcji
    int sum = *ptr1 + *ptr2;
    printf("Suma: %d\n", sum);
}
```

1 Dane są następujące wyrazy i znaki: void int int int foo fun x ( ) ( ) ( ) , * Ułóż je we właściwej kolejności, aby otrzymać nagłówek funkcji fun, której argumentami są wskaźnik na funkcję i liczba całkowita. Następnie dodaj dowolną implementację funkcji i stwórz dla niej przypadek testowy.

```c
#include <stdio.h>

void fun(int (*ptr)(void), int num);

int foo(void);
int bar(void);

int main() {
    fun(foo, 5);
    fun(bar, 10);

    return 0;
}

void fun(int (*ptr)(void), int num) {
    printf("Wynik funkcji: %d\n", ptr());
    printf("Liczba: %d\n", num);
}

int foo(void) {
    return 42;
}

int bar(void) {
    return 99;
}
```