Zadania 2

Napisz funkcję, której argumentem jest napis. Funkcja zwraca numer indeksu, na którym występuje ostatnia od lewej cyfra. W przypadku pustego napisu lub braku cyfry w napisie, funkcja powinna zwracać -1. W zadaniu nie korzystaj z funkcji bibliotecznych poza instrukcjami wejścia/wyjścia. Stwórz przypadek testowy.

```c
#include <stdio.h>

int ostatniaCyfra(char* napis) {
    int indeks = -1;
    int i = 0;

    while (napis[i] != '\0') {
        if (napis[i] >= '0' && napis[i] <= '9') {
            indeks = i;
        }
        i++;
    }

    return indeks;
}

int main() {
    char napis[] = "Przykładowy napis 123";
    int indeks = ostatniaCyfra(napis);

    if (indeks == -1) {
        printf("Nie znaleziono żadnej cyfry.\n");
    } else {
        printf("Ostatnia cyfra znaleziona na indeksie: %d\n", indeks);
    }

    return 0;
}
```

2 Napisz funkcję, której argumentem jest napis. Funkcja zwraca numer indeksu, na którym występuje pierwsza od lewej mała litera. W przypadku pustego napisu lub braku małych liter, funkcja powinna zwracać zero. W zadaniu nie korzystaj z funkcji bibliotecznych poza instrukcjami wejścia/wyjścia. Stwórz przypadek testowy.

```c
#include <stdio.h>

int pierwszaMalaLitera(char* napis) {
    int indeks = 0;
    int i = 0;

    while (napis[i] != '\0') {
        if (napis[i] >= 'a' && napis[i] <= 'z') {
            indeks = i + 1;
            break;
        }
        i++;
    }

    return indeks;
}

int main() {
    char napis[] = "Przykładowy Napis";
    int indeks = pierwszaMalaLitera(napis);

    if (indeks == 0) {
        printf("Nie znaleziono żadnej małej litery.\n");
    } else {
        printf("Pierwsza mała litera znaleziona na indeksie: %d\n", indeks);
    }

    return 0;
}
```

2 Napisz funkcję, której argumentem jest napis. Funkcja ma zwrócić ile znaków będących przecinkami jest w napisie. Stwórz przypadek testowy.

```c
#include <stdio.h>

int iloscPrzecinkow(char* napis) {
    int ilosc = 0;
    int i = 0;

    while (napis[i] != '\0') {
        if (napis[i] == ',') {
            ilosc++;
        }
        i++;
    }

    return ilosc;
}

int main() {
    char napis[] = "To jest, przykladowy, napis, z przecinkami.";
    int ilosc = iloscPrzecinkow(napis);

    printf("Liczba przecinkow w napisie: %d\n", ilosc);

    return 0;
}
```

2 Napisz funkcję, której argumentem jest napis. Funkcja ma zwrócić liczbę znaków cyfr nieparzystych występujących w napisie. Stwórz przypadek testowy.

```c
#include <stdio.h>

int liczbaCyfrNieparzystych(char* napis) {
    int liczba = 0;
    int i = 0;

    while (napis[i] != '\0') {
        if (napis[i] >= '0' && napis[i] <= '9') {
            int cyfra = napis[i] - '0';
            if (cyfra % 2 != 0) {
                liczba++;
            }
        }
        i++;
    }

    return liczba;
}

int main() {
    char napis[] = "Przykl4d0wy 7n4p1s z9 cyfram1";
    int liczba = liczbaCyfrNieparzystych(napis);

    printf("Liczba cyfr nieparzystych w napisie: %d\n", liczba);

    return 0;
}
```

2 Napisz funkcję, której argumentem jest napis. Funkcja ma za zadanie usunąć znaki z napisu o nieparzystych indeksach. Stwórz przypadek testowy. Przykład: Z napisu “ABCxyz” powinny pozostać napis “ACy”.

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

void usunZnakiNieparzyste(char* napis) {
    int dlugosc = strlen(napis);
    int i, j = 0;

    for (i = 0; i < dlugosc; i += 2) {
        napis[j] = napis[i];
        j++;
    }

    napis[j] = '\0';
}

int main() {
    char napis[] = "ABCxyz";

    printf("Przed modyfikacja: %s\n", napis);

    usunZnakiNieparzyste(napis);

    printf("Po modyfikacji: %s\n", napis);

    return 0;
}
```

2 Napisz funkcję, której argumentem jest napis. Funkcja ma za zadanie zwrócić nowy napis, który zawiera tylko co trzeci znak z oryginalnego napisu. Stwórz przypadek testowy. Przykład: Dla napisu "ABCXYZ" funkcja powinna zwrócić "AX".

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

char* coTrzeciZnak(char* napis) {
    int dlugosc = strlen(napis);
    int nowaDlugosc = (dlugosc / 3) + 1;
    char* nowyNapis = (char*) malloc(nowaDlugosc * sizeof(char));

    int i, j = 0;
    for (i = 0; i < dlugosc; i += 3) {
        nowyNapis[j] = napis[i];
        j++;
    }

    nowyNapis[j] = '\0';

    return nowyNapis;
}

int main() {
    char napis[] = "ABCXYZ";

    printf("Przed modyfikacja: %s\n", napis);

    char* nowyNapis = coTrzeciZnak(napis);

    printf("Po modyfikacji: %s\n", nowyNapis);

    free(nowyNapis);

    return 0;
}
```

2 Napisz funkcję, której argumentem jest napis. Funkcja powinna zwrócić ile znaków będących cyframi lub literami jest w danym napisie. W zadaniu nie korzystaj z funkcji bibliotecznych poza instrukcjami wejścia/wyjścia. Stwórz przypadek testowy.

```c
#include <stdio.h>

int liczbaZnakowAlfaNum(char* napis) {
    int liczba = 0;
    int i = 0;

    while (napis[i] != '\0') {
        if ((napis[i] >= 'a' && napis[i] <= 'z') || (napis[i] >= 'A' && napis[i] <= 'Z') || (napis[i] >= '0' && napis[i] <= '9')) {
            liczba++;
        }
        i++;
    }

    return liczba;
}

int main() {
    char napis[] = "Przykladowy123 napis!@#";
    int liczba = liczbaZnakowAlfaNum(napis);

    printf("Liczba znakow alfanumerycznych w napisie: %d\n", liczba);

    return 0;
}
```

2 Napisz funkcję, która przyjmuje jako argument napis. Funkcja powinna zwrócić liczbę znaków, które są małymi literami alfabetu angielskiego. Nie używaj funkcji bibliotecznych, z wyjątkiem funkcji wejścia/wyjścia. Stwórz przypadek testowy

```c
#include <stdio.h>

int liczbaMalychLiter(char* napis) {
    int liczba = 0;
    int i = 0;

    while (napis[i] != '\0') {
        if (napis[i] >= 'a' && napis[i] <= 'z') {
            liczba++;
        }
        i++;
    }

    return liczba;
}

int main() {
    char napis[] = "Przykladowy Napis z MALYMI LITERAMI";
    int liczba = liczbaMalychLiter(napis);

    printf("Liczba malych liter w napisie: %d\n", liczba);

    return 0;
}
```

2 Napisz funkcję, której argumentem jest napis. Funkcja ma za zadanie zwrócić liczbę znaków, które są cyframi od 0 do 4. W zadaniu nie korzystaj z funkcji bibliotecznych poza instrukcjami wejścia/wyjścia. Stwórz przypadek testowy.

```c
#include <stdio.h>

int liczbaCyfrDoCztery(char* napis) {
    int liczba = 0;
    int i = 0;

    while (napis[i] != '\0') {
        if (napis[i] >= '0' && napis[i] <= '4') {
            liczba++;
        }
        i++;
    }

    return liczba;
}

int main() {
    char napis[] = "Przykladowy tekst z cyframi 01234";
    int liczba = liczbaCyfrDoCztery(napis);

    printf("Liczba cyfr od 0 do 4 w napisie: %d\n", liczba);

    return 0;
}
```

2 Napisz funkcję, której argumentem są dwa napisy. Funkcja powinna zwrócić informację ile znaków będących cyframi jest w krótszym napisie. Jeśli napisy są równej długości, to funkcja powinna zwrócić liczbę znaków cyfr z pierwszego napisu. Stwórz przypadek testowy. W zadaniu nie korzystaj z funkcji bibliotecznych poza instrukcjami wejścia/wyjścia.

```c
#include <stdio.h>

int liczbaCyfrWKrotszym(char* napis1, char* napis2) {
    int dlugosc1 = 0;
    int dlugosc2 = 0;
    int i = 0;

    // Obliczamy długość pierwszego napisu
    while (napis1[i] != '\0') {
        dlugosc1++;
        i++;
    }

    i = 0;  // Resetujemy licznik

    // Obliczamy długość drugiego napisu
    while (napis2[i] != '\0') {
        dlugosc2++;
        i++;
    }

    // Porównujemy długości napisów
    if (dlugosc1 < dlugosc2) {
        return liczbaCyfr(napis1, dlugosc1);
    } else if (dlugosc1 > dlugosc2) {
        return liczbaCyfr(napis2, dlugosc2);
    } else {
        return liczbaCyfr(napis1, dlugosc1);
    }
}

int liczbaCyfr(char* napis, int dlugosc) {
    int liczba = 0;
    int i = 0;

    while (i < dlugosc) {
        if (napis[i] >= '0' && napis[i] <= '9') {
            liczba++;
        }
        i++;
    }

    return liczba;
}

int main() {
    char napis1[] = "123456";
    char napis2[] = "abc123";

    int wynik = liczbaCyfrWKrotszym(napis1, napis2);

    printf("Liczba cyfr w krótszym napisie: %d\n", wynik);

    return 0;
}
```

2 Napisz funkcję, której argumentami są dwa napisy. Funkcja powinna zwrócić informację ile znaków będących wielkimi literami jest w dłuższym napisie. Jeśli napisy są równej długości, to funkcja powinna zwrócić liczbę wielkich liter z drugiego napisu. Stwórz przypadek testowy. W zadaniu nie korzystaj z funkcji bibliotecznych poza instrukcjami wejścia/wyjścia.

```c
#include <stdio.h>

int liczbaWielkichLiterWDluzszym(char* napis1, char* napis2) {
    int dlugosc1 = 0;
    int dlugosc2 = 0;
    int i = 0;

    // Obliczamy długość pierwszego napisu
    while (napis1[i] != '\0') {
        dlugosc1++;
        i++;
    }

    i = 0;  // Resetujemy licznik

    // Obliczamy długość drugiego napisu
    while (napis2[i] != '\0') {
        dlugosc2++;
        i++;
    }

    // Porównujemy długości napisów
    if (dlugosc1 < dlugosc2) {
        return liczbaWielkichLiter(napis2, dlugosc2);
    } else if (dlugosc1 > dlugosc2) {
        return liczbaWielkichLiter(napis1, dlugosc1);
    } else {
        return liczbaWielkichLiter(napis2, dlugosc2);
    }
}

int liczbaWielkichLiter(char* napis, int dlugosc) {
    int liczba = 0;
    int i = 0;

    while (i < dlugosc) {
        if (napis[i] >= 'A' && napis[i] <= 'Z') {
            liczba++;
        }
        i++;



    }

    return liczba;
}

int main() {
    char napis1[] = "Przykladowy tekst";
    char napis2[] = "Inny tekst";

    int wynik = liczbaWielkichLiterWDluzszym(napis1, napis2);

    printf("Liczba wielkich liter w dłuższym napisie: %d\n", wynik);

    return 0;
}
```

2 Napisz funkcję, której argumentami są dwa napisy. Funkcja powinna zwrócić informację ile znaków będących małymi literami jest w krótszym napisie. Jeśli napisy są równej długości, to funkcja powinna zwrócić liczbę małych liter z pierwszego napisu. Stwórz przypadek testowy. W zadaniu nie korzystaj z funkcji bibliotecznych poza instrukcjami wejścia/wyjścia.

```c
#include <stdio.h>

int liczbaMalychLiterWKrotszym(char* napis1, char* napis2) {
    int dlugosc1 = 0;
    int dlugosc2 = 0;
    int i = 0;

    // Obliczamy długość pierwszego napisu
    while (napis1[i] != '\0') {
        dlugosc1++;
        i++;
    }

    i = 0;  // Resetujemy licznik

    // Obliczamy długość drugiego napisu
    while (napis2[i] != '\0') {
        dlugosc2++;
        i++;
    }

    // Porównujemy długości napisów
    if (dlugosc1 < dlugosc2) {
        return liczbaMalychLiter(napis1, dlugosc1);
    } else if (dlugosc1 > dlugosc2) {
        return liczbaMalychLiter(napis2, dlugosc2);
    } else {
        return liczbaMalychLiter(napis1, dlugosc1);
    }
}

int liczbaMalychLiter(char* napis, int dlugosc) {
    int liczba = 0;
    int i = 0;

    while (i < dlugosc) {
        if (napis[i] >= 'a' && napis[i] <= 'z') {
            liczba++;

        }
        i++;
    }

    return liczba;
}

int main() {
    char napis1[] = "Przykladowy tekst";
    char napis2[] = "Inny tekst";

    int wynik = liczbaMalychLiterWKrotszym(napis1, napis2);

    printf("Liczba małych liter w krótszym napisie: %d\n", wynik);

    return 0;
}
```

2 Napisz funkcję, której argumentami są dwa napisy. Funkcja powinna zwrócić informację ile znaków będących literami (bez względu na wielkość) jest w dłuższym napisie. Jeśli napisy są równej długości, to funkcja powinna zwrócić liczbę liter z drugiego napisu. Stwórz przypadek testowy. W zadaniu nie korzystaj z funkcji bibliotecznych poza instrukcjami wejścia/wyjścia.

```c
#include <stdio.h>

int liczbaLiterWDluzszym(char* napis1, char* napis2) {
    int dlugosc1 = 0;
    int dlugosc2 = 0;
    int i = 0;

    // Obliczamy długość pierwszego napisu
    while (napis1[i] != '\0') {
        dlugosc1++;
        i++;
    }

    i = 0;  // Resetujemy licznik

    // Obliczamy długość drugiego napisu
    while (napis2[i] != '\0') {
        dlugosc2++;
        i++;
    }

    // Porównujemy długości napisów
    if (dlugosc1 < dlugosc2) {
        return liczbaLiter(napis2, dlugosc2);
    } else if (dlugosc1 > dlugosc2) {
        return liczbaLiter(napis1, dlugosc1);
    } else {
        return liczbaLiter(napis2, dlugosc2);
    }
}

int liczbaLiter(char* napis, int dlugosc) {
    int liczba = 0;
    int i = 0;

    while (i < dlugosc) {
        if ((napis[i] >= 'a' && napis[i] <= 'z') || (napis[i] >= 'A' && napis[i] <= 'Z')) {
            liczba++;
        }
        i++;



    }

    return liczba;
}

int main() {
    char napis1[] = "Przykladowy tekst";
    char napis2[] = "Inny tekst";

    int wynik = liczbaLiterWDluzszym(napis1, napis2);

    printf("Liczba liter w dłuższym napisie: %d\n", wynik);

    return 0;
}
```

2 Napisz funkcję, której argumentem jest napis. Jeśli napis zawiera inne znaki niż cyfr, to funkcja ma zwracać zero. Jeśli napis zawiera tylko cyfry, funkcja ma zwrócić liczbę całkowitą powstałą z przepisania kolejno znaków cyfr. Załóż, że napis jest długości dokładnie 3. W zadaniu nie korzystaj z funkcji bibliotecznych poza instrukcjami wejścia/wyjścia. Stwórz przypadek testowy. Przykład. Dla napisu “345” funkcja ma zwrócić 345 (jako liczbę w typie całkowitoliczbowym).

```c
#include <stdio.h>

int zliczCyfry(char* napis) {
    int cyfry = 0;
    int i = 0;

    while (i < 3) {
        if (napis[i] >= '0' && napis[i] <= '9') {
            cyfry = cyfry * 10 + (napis[i] - '0');
        } else {
            return 0;
        }
        i++;
    }

    return cyfry;
}

int main() {
    char napis[] = "345";

    int wynik = zliczCyfry(napis);

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

    return 0;
}
```

2 Napisz funkcję, której argumentem jest napis. Funkcja zwraca numer indeksu, na którym występuje ostatnia mała litera. W przypadku pustego napisu lub braku małej litery w napisie, funkcja powinna zwracać -1. W zadaniu nie korzystaj z funkcji bibliotecznych poza instrukcjami wejścia/wyjścia. Stwórz przypadek testowy.

```c
#include <stdio.h>

int numerOstatniejMalejLitery(char* napis) {
    int i = 0;
    int indeks = -1;  // Inicjalizujemy wartość indeksu na -1

    while (napis[i] != '\0') {
        if (napis[i] >= 'a' && napis[i] <= 'z') {
            indeks = i;  // Zapisujemy aktualny indeks
        }
        i++;
    }

    return indeks;
}

int main() {
    char napis[] = "Przykladowy Tekst";

    int wynik = numerOstatniejMalejLitery(napis);

    printf("Numer ostatniej malej litery: %d\n", wynik);

    return 0;
}
```

2 Napisz funkcję, której argumentem jest napis. Funkcja powinna zwrócić ile znaków będących cyframi z systemu szesnastkowego jest w danym napisie. W zadaniu nie korzystaj z funkcji bibliotecznych poza instrukcjami wejścia/wyjścia. Stwórz przypadek testowy.

```c
#include <stdio.h>

int zliczCyfrySzesnastkowe(char* napis) {
    int cyfry = 0;
    int i = 0;

    while (napis[i] != '\0') {
        if ((napis[i] >= '0' && napis[i] <= '9') || (napis[i] >= 'A' && napis[i] <= 'F')) {
            cyfry++;
        }
        i++;
    }

    return cyfry;
}

int main() {
    char napis[] = "1A2B3C4D5E";

    int wynik = zliczCyfrySzesnastkowe(napis);

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

    return 0;
}
```

Zadania 3

Napisz funkcję, której argumentem jest dwuwymiarowa tablica tablic (zawierająca zmienne typu int) oraz jej wymiary n i m. Funkcja ma zwrócić największy element nieparzysty w tablicy (chodzi o wartości nieparzyste, a nie indeksy). W przypadku ich braku, zwróć -1. Stwórz przypadek testowy.

```c
#include <stdio.h>

int ZnajdzNajwiekszyNieparzysty( int n, int m, int tablica[][m]) {
    int najwiekszy_nieparzysty = -1;
    int znaleziono_nieparzysty = 0;

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (tablica[i][j] % 2 != 0 && tablica[i][j] > najwiekszy_nieparzysty) {
                najwiekszy_nieparzysty = tablica[i][j];
                znaleziono_nieparzysty = 1;
            }
        }
    }

    if (znaleziono_nieparzysty) {
        return najwiekszy_nieparzysty;
    } else {
        return -1;
    }
}

int main() {
    int tablica[][3] = {
        {2, 4, 6},
        {1, 3, 5},
        {8, 10, 12}
    };

    int n = sizeof(tablica) / sizeof(tablica[0]);
    int m = sizeof(tablica[0]) / sizeof(tablica[0][0]);

    int wynik = ZnajdzNajwiekszyNieparzysty(n, m,tablica);
    if (wynik != -1) {
        printf("Największy element nieparzysty: %d\n", wynik);
    } else {
        printf("Brak elementów nieparzystych w tablicy.\n");
    }

    return 0;

}
```

3 Napisz funkcję, której argumentem jest dwuwymiarowa tablica tablic (zawierająca zmienne typu int) oraz jej wymiary n i m. Funkcja ma zwrócić najmniejszy element parzysty w tablicy (chodzi o wartości parzyste, a nie indeksy). W przypadku ich braku, zwróć -1. Stwórz przypadek testowy.

```c
#include <stdio.h>

int ZnajdzNajmniejszyParzysty(int n, int m, int tablica[][m]) {
    int najmniejszy_parzysty = -1;
    int znaleziono_parzysty = 0;

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (tablica[i][j] % 2 == 0) {
                if (!znaleziono_parzysty || tablica[i][j] < najmniejszy_parzysty) {
                    najmniejszy_parzysty = tablica[i][j];
                    znaleziono_parzysty = 1;
                }
            }
        }
    }

    return najmniejszy_parzysty;
}

int main() {
    int tablica[][3] = {
        {1, 3, 5},
        {2, 4, 6},
        {7, 9, 11}
    };

    int n = sizeof(tablica) / sizeof(tablica[0]);
    int m = sizeof(tablica[0]) / sizeof(tablica[0][0]);

    int wynik = ZnajdzNajmniejszyParzysty(n, m,tablica);
    if (wynik != -1) {
        printf("Najmniejszy element parzysty: %d\n", wynik);
    } else {
        printf("Brak elementów parzystych w tablicy.\n");
    }

    return 0;
}
```

3 Napisz funkcję, której argumentem jest dwuwymiarowa kwadratowa tablica tablic (zawierająca zmienne typu int) oraz jej wymiar n. Funkcja ma odwrócić kolejność elementów w wierszach o parzystych indeksach. Stwórz przypadek testowy.

Przykład:

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \longrightarrow \begin{bmatrix} 4 & 3 & 2 & 1 \\ 5 & 6 & 7 & 8 \\ 12 & 11 & 10 & 9 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

```c
#include <stdio.h>

void odwrocWiersze(int macierz[][4], int n) {
    for (int i = 0; i < n; i++) {
        if (i % 2 == 0) {  // Sprawdzenie, czy indeks wiersza jest parzysty
            for (int j = 0, k = 3; j < k; j++, k--) {
                int temp = macierz[i][j];
                macierz[i][j] = macierz[i][k];
                macierz[i][k] = temp;
            }
        }
    }
}

void wypiszMacierz(int macierz[][4], int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%d ", macierz[i][j]);
        }
        printf("\n");
    }
}

int main() {
    int macierz[][4] = {
        {1, 2, 3, 4},
        {5, 6, 7, 8},
        {9, 10, 11, 12},
        {13, 14, 15, 16}
    };
    int n = sizeof(macierz) / sizeof(macierz[0]);

    printf("Macierz1:\n");
    wypiszMacierz(macierz, n);

    odwrocWiersze(macierz, n);

    printf("\nMacierz2:\n");
    wypiszMacierz(macierz, n);


    return 0;
}
```

3 Napisz funkcję, której argumentem jest dwuwymiarowa kwadratowa tablica tablic (zawierająca zmienne typu int) oraz jej wymiar n. Funkcja ma odwrócić kolejność elementów w kolumnach o parzystych indeksach. Stwórz przypadek testowy.

Przykład:

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \longrightarrow \begin{bmatrix} 13 & 2 & 15 & 4 \\ 9 & 6 & 11 & 8 \\ 5 & 10 & 7 & 12 \\ 1 & 14 & 3 & 16 \end{bmatrix}$$

```c
#include <stdio.h>

void odwrocKolumny(int tablica[][100], int n) {
    for (int i = 0; i < n; i++) {
        if (i % 2 == 0) { // Sprawdzanie parzystości indeksu kolumny
            int dol = 0; // Indeks dolnego elementu
            int gora = n - 1; // Indeks górnego elementu
            while (dol < gora) {
                int temp = tablica[dol][i]; // Zamiana elementów
                tablica[dol][i] = tablica[gora][i];
                tablica[gora][i] = temp;
                dol++;
                gora--;
            }
        }
    }
}

void wypiszMacierz(int tablica[][100], int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%d ", tablica[i][j]);
        }
        printf("\n");
    }
}

int main() {
    int macierz[][100] = {
        {1, 2, 3, 4},
        {5, 6, 7, 8},
        {9, 10, 11, 12},
        {13, 14, 15, 16}
    };
    int wymiar = 4;

    printf("Macierz1:\n");
    wypiszMacierz(macierz, wymiar);

    odwrocKolumny(macierz, wymiar);

    printf("\nOdwrócona macierz:\n");
    wypiszMacierz(macierz, wymiar);

    return 0;
}
```

3 Napisz funkcję, której argumentem jest dwuwymiarowa tablica tablic (zawierająca zmienne typu int) oraz jej wymiary n, $n > 1$ i m, $m > 1$. Funkcja ma zwrócić średnią elementów stojących na nieparzystych indeksach (oba mają być jednocześnie nieparzyste). Stwórz przypadek testowy.

```c
#include <stdio.h>

double sredniaNieparzystych(int tablica[][100], int n, int m) {
    int suma = 0;
    int licznik = 0;

    for (int i = 1; i < n; i += 2) {
        for (int j = 1; j < m; j += 2) {
            suma += tablica[i][j];
            licznik++;
        }
    }

    if (licznik > 0) {
        return (double)suma / licznik;
    } else {
        return -1; // Zwracanie -1 w przypadku braku elementów spełniających warunek
    }
}

int main() {
    int macierz[][100] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };
    int n = 3; // Wymiar n
    int m = 3; // Wymiar m

    double srednia = sredniaNieparzystych(macierz, n, m);

    if (srednia != -1) {
        printf("Średnia elementów na nieparzystych indeksach: %.2f\n", srednia);
    } else {
        printf("Brak elementów spełniających warunek\n");
    }

    return 0;
}
```

3 Napisz funkcję, której argumentem jest dwuwymiarowa tablica tablic (zawierająca zmienne typu int) oraz jej wymiary n, $n > 1$ i m, $m > 1$. Funkcja powinna zwrócić sumę elementów stojących na parzystych indeksach (zarówno indeksy wierszy jak i kolumn są parzyste). Stwórz przypadek testowy.

```c
#include <stdio.h>

int oblicz_sume_parzystych(int n, int m, int tablica[][m]) {
    int suma = 0;

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (i % 2 == 0 && j % 2 == 0) {
                suma += tablica[i][j];
            }
        }
    }

    return suma;
}

int main() {
    int tablica[][4] = {
        {1, 2, 3, 4},
        {5, 6, 7, 8},
        {9, 10, 11, 12},
        {13, 14, 15, 16}
    };

    int n = sizeof(tablica) / sizeof(tablica[0]);
    int m = sizeof(tablica[0]) / sizeof(tablica[0][0]);

    int wynik = oblicz_sume_parzystych(n, m, tablica);
    printf("Suma elementów na parzystych indeksach: %d\n", wynik);

    return 0;
}
```

3 Napisz funkcję, która przyjmuje jako argumenty dwuwymiarową tablicę tablic (zawierającą zmienne typu int) oraz jej wymiary n, $n > 1$ i m, $m > 1$. Funkcja ma zwrócić najmniejszy element stojący na nieparzystych indeksach (zarówno indeksy wierszy jak i kolumn są nieparzyste). Stwórz przypadek testowy.

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

int najmniejszyNieparzysty(int tablica[][100], int n, int m) {
    int najmniejszy = INT_MAX;

    for (int i = 1; i < n; i += 2) {
        for (int j = 1; j < m; j += 2) {
            if (tablica[i][j] < najmniejszy) {
                najmniejszy = tablica[i][j];
            }
        }
    }

    return najmniejszy;
}

int main() {
    int macierz[][100] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };
    int n = 3; // Wymiar n
    int m = 3; // Wymiar m

    int najmniejszy = najmniejszyNieparzysty(macierz, n, m);

    if (najmniejszy != INT_MAX) {
        printf("Najmniejszy element na nieparzystych indeksach: %d\n", najmniejszy);
    } else {
        printf("Brak elementów na nieparzystych indeksach.\n");
    }

    return 0;
}
```

3 Napisz funkcję, której argumentem jest dwuwymiarowa tablica tablic (zawierająca zmienne typu int) oraz jej wymiary n, $n > 1$ i m, $m > 1$. Funkcja ma zwrócić największy element stojący na parzystych indeksach (zarówno indeksy wierszy jak i kolumn są parzyste). Stwórz przypadek testowy.

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

int najwiekszyParzysty(int tablica[][100], int n, int m) {
    int najwiekszy = INT_MIN;

    for (int i = 0; i < n; i += 2) {
        for (int j = 0; j < m; j += 2) {
            if (tablica[i][j] > najwiekszy) {
                najwiekszy = tablica[i][j];
            }
        }
    }

    return najwiekszy;
}

int main() {
    int macierz[][100] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };
    int n = 3; // Wymiar n
    int m = 3; // Wymiar m

    int najwiekszy = najwiekszyParzysty(macierz, n, m);

    if (najwiekszy != INT_MIN) {
        printf("Największy element na parzystych indeksach: %d\n", najwiekszy);
    } else {
        printf("Brak elementów na parzystych indeksach.\n");
    }

    return 0;
}
```

3 Napisz funkcję, której argumentem jest dwuwymiarowa tablica tablic (zawierająca zmienne typu int) oraz jej wymiary n i m. Funkcja ma przestawić pierwszą i ostatnią kolumnę miedzy sobą. Jeśli tablica ma mniej niż dwie kolumny, funkcja ma nic nie robić. Stwórz przypadek testowy.

Przykład.

$$\begin{bmatrix} 2 & 3 & -3 \\ 1 & 4 & 7 \\ -3 & -6 & 11 \\ -2 & 8 & 23 \end{bmatrix} \longrightarrow \begin{bmatrix} -3 & 3 & 2 \\ 7 & 4 & 1 \\ 11 & -6 & -3 \\ 23 & 8 & -2 \end{bmatrix}$$

```c
#include <stdio.h>

void przestawKolumny(int tablica[][100], int n, int m) {
    if (m < 2) {
        return; // Nic nie robimy, jeśli tablica ma mniej niż dwie kolumny
    }

    for (int i = 0; i < n; i++) {
        int temp = tablica[i][0];
        tablica[i][0] = tablica[i][m - 1];
        tablica[i][m - 1] = temp;
    }
}

void wypiszTablice(int tablica[][100], int n, int m) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            printf("%d ", tablica[i][j]);
        }
        printf("\n");
    }
}

int main() {
    int macierz[][100] = {
        {2, 3, -3},
        {1, 4, 7},
        {-3, -6, 11},
        {-2, 8, 23}
    };
    int n = 4; // Wymiar n
    int m = 3; // Wymiar m

    printf("Macierz1:\n");
    wypiszTablice(macierz, n, m);

    przestawKolumny(macierz, n, m);

    printf("\nPo przestawieniu:\n");
    wypiszTablice(macierz, n, m);

    return 0;
}
```

3 Napisz funkcję, której argumentem jest dwuwymiarowa kwadratowa tablica tablic (zawierająca zmienne typu int) oraz jej wymiar n, $n > 1$. Funkcja ma zwrócić iloczyn elementów znajdujących się na przekątnej pobocznej (suma indeksów wiersza i kolumny jest równa $n - 1$). Stwórz przypadek testowy.

```c
#include <stdio.h>

int iloczynPrzekatnejPobocznej(int tablica[][100], int n) {
    int iloczyn = 1;

    for (int i = 0; i < n; i++) {
        iloczyn *= tablica[i][n - 1 - i];
    }

    return iloczyn;
}

int main() {
    int macierz[][100] = {
        {2, 3, -3},
        {1, 4, 7},
        {-3, -6, 11}
    };
    int n = 3; // Wymiar n

    int wynik = iloczynPrzekatnejPobocznej(macierz, n);

    printf("Iloczyn elementow na przekatnej pobocznej: %d\n", wynik);

    return 0;
}
```

3 Napisz funkcję, której argumentem jest dwuwymiarowa kwadratowa tablica tablic (zawierająca zmienne typu int) oraz jej wymiar n, $n > 1$. Funkcja ma zwrócić iloczyn elementów znajdujących się na głównej przekątnej (oba indeksy pozycji są sobie równe). Stwórz przypadek testowy.

```c
#include <stdio.h>

int iloczynPrzekatnejGlownej(int tablica[][100], int n) {
    int iloczyn = 1;

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

    return iloczyn;
}

int main() {
    int macierz[][100] = {
        {2, 3, -3},
        {1, 4, 7},
        {-3, -6, 11}
    };
    int n = 3; // Wymiar n

    int wynik = iloczynPrzekatnejGlownej(macierz, n);

    printf("Iloczyn elementow na glownej przekatnej: %d\n", wynik);

    return 0;
}
```

3 Napisz funkcję, której argumentem jest dwuwymiarowa kwadratowa tablica tablic (zawierająca zmienne typu int) oraz jej wymiar n, $n > 1$. Funkcja powinna zwrócić największy element znajdujący się na przekątnej pobocznej (suma indeksów jest równa $n-1$). Stwórz przypadek testowy.

```c
#include <stdio.h>

int najwiekszyPrzekatnaPoboczna(int tablica[][100], int n) {
    int maxElement = tablica[0][n - 1]; // Przypisujemy pierwszy element na przekatnej pobocznej jako maksimum

    for (int i = 1; i < n; i++) {
        int element = tablica[i][n - 1 - i];
        if (element > maxElement) {
            maxElement = element;
        }
    }

    return maxElement;
}

int main() {
    int macierz[][100] = {
        {2, 3, -3},
        {1, 4, 7},
        {-3, -6, 11}
    };
    int n = 3; // Wymiar n

    int wynik = najwiekszyPrzekatnaPoboczna(macierz, n);

    printf("Najwiekszy element na przekatnej pobocznej: %d\n", wynik);

    return 0;
}
```

3 Napisz funkcję, której argumentem jest dwuwymiarowa tablica elementów (zawierająca zmienne typu int) oraz jej wymiary n i m. Funkcja ma przestawić pierwszy i ostatni wiersz między sobą. Jeśli tablica ma mniej niż dwa wiersze, funkcja ma nic nie robić. Stwórz przypadek testowy.

Przykład:

$$\begin{bmatrix} 2 & 3 & -3 \\ 1 & 4 & 7 \\ -3 & -6 & 11 \\ -2 & 8 & 23 \end{bmatrix} \longrightarrow \begin{bmatrix} -2 & 8 & 23 \\ 1 & 4 & 7 \\ -3 & -6 & 11 \\ 2 & 3 & -3 \end{bmatrix}$$

```c
#include <stdio.h>

void przestawWiersze(int tablica[][100], int n, int m) {
    if (n < 2) {
        return; // Jeśli mniej niż dwa wiersze, nic nie robimy
    }

    for (int j = 0; j < m; j++) {
        int temp = tablica[0][j]; // Przechowujemy pierwszy element

        tablica[0][j] = tablica[n - 1][j]; // Przypisujemy ostatni element na pierwsze miejsce
        tablica[n - 1][j] = temp; // Przypisujemy przechowywany element na ostatnie miejsce
    }
}

int main() {
    int macierz[][100] = {
        {2, 3, -3},
        {1, 4, 7},
        {-3, -6, 11},
        {-2, 8, 23}
    };
    int n = 4; // Liczba wierszy
    int m = 3; // Liczba kolumn

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

    przestawWiersze(macierz, n, m);

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

        printf("\n");
    }

    return 0;
}
```

3 Napisz funkcję, której argumentem jest dwuwymiarowa tablica elementów (zawierająca zmienne typu int) oraz jej wymiary n i m. Funkcja ma zamienić miejscami drugi wiersz z przedostatnim. Jeśli tablica ma mniej niż cztery wiersze, funkcja ma nic nie robić. Stwórz przypadek testowy.

Przykład:

$$\begin{bmatrix} 2 & 3 & -3 \\ 1 & 4 & 7 \\ -3 & -6 & 11 \\ -2 & 8 & 23 \end{bmatrix} \longrightarrow \begin{bmatrix} 2 & 3 & -3 \\ -3 & -6 & 11 \\ 1 & 4 & 7 \\ -2 & 8 & 23 \end{bmatrix}$$

```c
#include <stdio.h>

void zamienWiersze(int tablica[][100], int n, int m) {
    if (n < 4) {
        return; // Jeśli mniej niż cztery wiersze, nic nie robimy
    }

    for (int j = 0; j < m; j++) {
        int temp = tablica[1][j]; // Przechowujemy drugi wiersz

        tablica[1][j] = tablica[n - 2][j]; // Przypisujemy przedostatni wiersz na drugie miejsce
        tablica[n - 2][j] = temp; // Przypisujemy przechowywany drugi wiersz na przedostatnie miejsce
    }
}

int main() {
    int macierz[][100] = {
        {2, 3, -3},
        {1, 4, 7},
        {-3, -6, 11},
        {-2, 8, 23}
    };
    int n = 4; // Liczba wierszy
    int m = 3; // Liczba kolumn

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

    zamienWiersze(macierz, n, m);

    printf("\nMacierz po zamianie:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            printf("%d ", macierz[i][j]);

         }
         printf("\n");
     }

     return 0;
}
```

3 Napisz funkcję, której argumentem jest dwuwymiarowa tablica tablic (zawierająca zmienne typu int) oraz jej wymiary n i m. Funkcja ma zwrócić największy element nieparzysty w tablicy znajdujący się na głównej przekątnej (indeks wiersza i kolumny są sobie równe). W przypadku ich braku, zwróć -1. Stwórz przypadek testowy.

```c
#include <stdio.h>

int znajdzNajwiekszyNieparzysty(int tablica[][100], int n, int m) {
    int najwiekszyNieparzysty = -1; // Domyślna wartość, jeśli brak elementów nieparzystych na przekątnej

    for (int i = 0; i < n; i++) {
        if (tablica[i][i] % 2 != 0 && tablica[i][i] > najwiekszyNieparzysty) {
            najwiekszyNieparzysty = tablica[i][i];
        }
    }

    return najwiekszyNieparzysty;
}

int main() {
    int macierz[][100] = {
        {2, 3, -3},
        {1, 4, 7},
        {-3, -6, 11}
    };
    int n = 3; // Liczba wierszy
    int m = 3; // Liczba kolumn

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

    int najwiekszyNieparzysty = znajdzNajwiekszyNieparzysty(macierz, n, m);

    printf("\nNajwiekszy element nieparzysty na glownej przekatnej: %d\n", najwiekszyNieparzysty);

    return 0;
}
```

3 Napisz funkcję, której argumentem jest dwuwymiarowa tablica tablic (zawierająca zmienne typu int) oraz jej wymiary n i m. Funkcja ma zwrócić największy element dodatni w tablicy. W przypadku ich braku, zwróć -1. Stwórz przypadek testowy.

```c
#include <stdio.h>

int znajdzNajwiekszyDodatni(int n, int m, int tablica[][m]) {
    int najwiekszy = -1; // Inicjalizujemy najwiekszy element jako -1
    int znalezionoDodatni = 0; // Flaga informujaca, czy znaleziono dodatni element

    // Przeszukujemy tablice w poszukiwaniu najwiekszego elementu dodatniego
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (tablica[i][j] > najwiekszy && tablica[i][j] > 0) {
                najwiekszy = tablica[i][j];
                znalezionoDodatni = 1; // Ustawiamy flage, że znaleziono dodatni element
            }
        }
    }

    if (znalezionoDodatni){
        return najwiekszy;
    } else {
        return -1;
    }
}

int main() {
    int tablica[3][4] = {
        {-2, 5, -8, 3},
        {0, 7, 25, 9},
        {1, -6, 2, 11}
    };

    int n = 3; // liczba wierszy
    int m = 4; // liczba kolumn

    int wynik = znajdzNajwiekszyDodatni(n, m, tablica);
    printf("Najwiekszy element dodatni: %d\n", wynik);

    return 0;
}
```

Zadanie 4

Stwórz strukturę Osoba o trzech polach imie (tablica znaków o rozmiarze 20), wiek (int), waga (float). Następnie stwórz funkcję, której argumentami jest tablica struktur Osoba oraz rozmiar tablicy. Funkcja ma zwrócić “osobę” (jako strukturę) o największej wadze. Stwórz przypadek testowy

```c
#define MAX_IMIE 20
#define MAX_OSOBY 5

struct Osoba {
    char imie[MAX_IMIE];
    int wiek;
    float waga;
};

struct Osoba znajdzNajciezszaOsobe(struct Osoba osoby[], int rozmiar) {
    struct Osoba najciezszaOsoba = osoby[0];

    for (int i = 1; i < rozmiar; i++) {
        if (osoby[i].waga > najciezszaOsoba.waga) {
            najciezszaOsoba = osoby[i];
        }
    }

    return najciezszaOsoba;
}

int main() {
    struct Osoba osoby[MAX_OSOBY] = {
        {"Jan", 30, 75.5},
        {"Anna", 25, 68.2},
        {"Mateusz", 35, 82.1},
        {"Katarzyna", 28, 69.8},
        {"Michał", 32, 87.3}
    };

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

    struct Osoba najciezsza = znajdzNajciezszaOsobe(osoby, rozmiar);

    printf("Najciezsza osoba: %s, wiek: %d, waga: %.2f\n", najciezsza.imie, najciezsza.wiek, najciezsza.waga);

    return 0;
}
```

4 Napisz strukturę Pralka z polami model (tablica znaków długości 15) oraz licznikPrania (typu int). Następnie napisz funkcję zrobPranie, której argumentem jest wskaźnik do struktury typu Pralka. Funkcja ma dodać 1 do pola licznikPrania w przekazanym argumencie. Stwórz przypadek testowy.

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

struct Pralka {
    char model[15];
    int licznikPrania;
};

void zrobPranie(struct Pralka *pralka) {
    pralka->licznikPrania++;
}

int main() {
    struct Pralka pralka1;
    strcpy(pralka1.model, "ABC123");
    pralka1.licznikPrania = 0;

    printf("Przed praniem: Model: %s, Licznik prania: %d\n", pralka1.model, pralka1.licznikPrania);

    zrobPranie(&pralka1);

    printf("Po praniu: Model: %s, Licznik prania: %d\n", pralka1.model, pralka1.licznikPrania);

    return 0;
}
```

4 Stwórz strukturę Samochod o dwóch polach marka (napis) oraz cena (dowolny typ całkowity). Następnie stwórz funkcję, której argumentami jest niepusta tablica struktur Samochod oraz rozmiar tablicy. Funkcja ma zwrócić najdroższą cenę za samochód z tablicy. Stwórz przypadek testowy.

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

struct Samochod {
    char marka[20];
    int cena;
};

int znajdzNajdrozszySamochod(struct Samochod samochody[], int rozmiar) {
    int najdrozszy = samochody[0].cena;

    for (int i = 1; i < rozmiar; i++) {
        if (samochody[i].cena > najdrozszy) {
            najdrozszy = samochody[i].cena;
        }
    }

    return najdrozszy;
}

int main() {
    struct Samochod samochody[3];
    strcpy(samochody[0].marka, "Audi");
    samochody[0].cena = 50000;
    strcpy(samochody[1].marka, "BMW");
    samochody[1].cena = 60000;
    strcpy(samochody[2].marka, "Mercedes");
    samochody[2].cena = 70000;

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

    int najdrozszySamochod = znajdzNajdrozszySamochod(samochody, rozmiar);
    printf("Najdrozszy samochod: %d\n", najdrozszySamochod);

    return 0;
}
```

4 Stwórz strukturę Osoba o trzech polach imie (tablica znaków o rozmiarze 20), wiek (int), wzrost (float). Następnie stwórz funkcję, której argumentami jest tablica struktur Osoba oraz rozmiar tablicy. Funkcja ma zwrócić “osobę” (jako strukturę) o najmłodszym wieku. Stwórz przypadek testowy

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

struct Osoba {
    char imie[20];
    int wiek;
    float wzrost;
};

struct Osoba znajdzNajmlodszaOsobe(struct Osoba osoby[], int rozmiar) {
    struct Osoba najmlodsza = osoby[0];

    for (int i = 1; i < rozmiar; i++) {
        if (osoby[i].wiek < najmlodsza.wiek) {
            najmlodsza = osoby[i];
        }
    }

    return najmlodsza;
}

int main() {
    struct Osoba osoby[3];
    strcpy(osoby[0].imie, "Jan");
    osoby[0].wiek = 25;
    osoby[0].wzrost = 180.5;
    strcpy(osoby[1].imie, "Anna");
    osoby[1].wiek = 30;
    osoby[1].wzrost = 165.0;
    strcpy(osoby[2].imie, "Piotr");
    osoby[2].wiek = 22;
    osoby[2].wzrost = 175.2;

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

    struct Osoba najmlodszaOsoba = znajdzNajmlodszaOsobe(osoby, rozmiar);
    printf("Najmlodsza osoba: %s, %d lat, %.2f cm wzrostu\n", najmlodszaOsoba.imie, najmlodszaOsoba.wiek, najmlodszaOsoba.wzrost);

    return 0;
}
```

4 Stwórz strukturę Samolot o dwóch polach model (napis) oraz liczba_silnikow (dowolny typ całkowity). Następnie stwórz funkcję, której argumentami jest niepusta tablica struktur Samolot oraz rozmiar tablicy. Funkcja ma zwrócić najmniejszą liczbę silników. Stwórz przypadek testowy.

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

struct Samolot {
    char model[20];
    int liczba_silnikow;
};

int znajdzNajmniejszaLiczbeSilnikow(struct Samolot samoloty[], int rozmiar) {
    int najmniejsza = samoloty[0].liczba_silnikow;

    for (int i = 1; i < rozmiar; i++) {
        if (samoloty[i].liczba_silnikow < najmniejsza) {
            najmniejsza = samoloty[i].liczba_silnikow;
        }
    }

    return najmniejsza;
}

int main() {
    struct Samolot samoloty[3];
    strcpy(samoloty[0].model, "Boeing 747");
    samoloty[0].liczba_silnikow = 4;
    strcpy(samoloty[1].model, "Airbus A320");
    samoloty[1].liczba_silnikow = 2;
    strcpy(samoloty[2].model, "Cessna 172");
    samoloty[2].liczba_silnikow = 1;

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

    int najmniejszaLiczbaSilnikow = znajdzNajmniejszaLiczbeSilnikow(samoloty, rozmiar);
    printf("Najmniejsza liczba silnikow: %d\n", najmniejszaLiczbaSilnikow);

    return 0;
}
```

4 Stwórz strukturę Komputer o dwóch polach marka (napis) oraz liczba_rdzeni (dowolny typ całkowity). Następnie stwórz funkcję, której argumentami jest niepusta tablica struktur Komputer oraz rozmiar tablicy. Funkcja ma zwrócić największą liczbę rdzeni. Stwórz przypadek testowy.

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

struct Komputer {
    char marka[20];
    int liczba_rdzeni;
};

int znajdzNajwiekszaLiczbeRdzeni(struct Komputer komputery[], int rozmiar) {
    int najwieksza = komputery[0].liczba_rdzeni;

    for (int i = 1; i < rozmiar; i++) {
        if (komputery[i].liczba_rdzeni > najwieksza) {
            najwieksza = komputery[i].liczba_rdzeni;
        }
    }

    return najwieksza;
}

int main() {
    struct Komputer komputery[3];
    strcpy(komputery[0].marka, "Dell");
    komputery[0].liczba_rdzeni = 4;
    strcpy(komputery[1].marka, "HP");
    komputery[1].liczba_rdzeni = 8;
    strcpy(komputery[2].marka, "Lenovo");
    komputery[2].liczba_rdzeni = 6;

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

    int najwiekszaLiczbaRdzeni = znajdzNajwiekszaLiczbeRdzeni(komputery, rozmiar);
    printf("Najwieksza liczba rdzeni: %d\n", najwiekszaLiczbaRdzeni);

    return 0;
}
```

4 Stwórz strukturę Samochod o dwóch polach model (napis) oraz moc_silnika (typ całkowity). Następnie stwórz funkcję, której argumentami jest niepusta tablica struktur Samochod oraz rozmiar tablicy. Funkcja powinna zwrócić najmniejszą moc silnika. Stwórz przypadek testowy.

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

struct Samochod {
    char model[20];
    int moc_silnika;
};

int znajdzNajmniejszaMocSilnika(struct Samochod samochody[], int rozmiar) {
    int najmniejsza = samochody[0].moc_silnika;

    for (int i = 1; i < rozmiar; i++) {
        if (samochody[i].moc_silnika < najmniejsza) {
            najmniejsza = samochody[i].moc_silnika;
        }
    }

    return najmniejsza;
}

int main() {
    struct Samochod samochody[3];
    strcpy(samochody[0].model, "Opel Astra");
    samochody[0].moc_silnika = 100;
    strcpy(samochody[1].model, "Volkswagen Golf");
    samochody[1].moc_silnika = 120;
    strcpy(samochody[2].model, "Ford Focus");
    samochody[2].moc_silnika = 90;

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

    int najmniejszaMocSilnika = znajdzNajmniejszaMocSilnika(samochody, rozmiar);
    printf("Najmniejsza moc silnika: %d\n", najmniejszaMocSilnika);

    return 0;
}
```

4 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>

enum Food {
    PIZZA,
    BURGER,
    PASTA,
    SUSHI,
    SALAD
};

int main() {
    enum Food foodArray[5] = {PIZZA, BURGER, PASTA, SUSHI, SALAD};

    for (int i = 0; i < 5; i++) {
        switch (foodArray[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 food\n");
                break;
        }
    }

    return 0;
}
```

4 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>

enum Food {
    PIZZA,
    BURGER,
    PASTA,
    SUSHI,
    SALAD
};

int main() {
    enum Food foodArray[5] = {PIZZA, BURGER, PASTA, SUSHI, SALAD};

    for (int i = 0; i < 5; i++) {
        if (foodArray[i] == PIZZA) {
            printf("Pizza\n");
        } else if (foodArray[i] == BURGER) {
            printf("Burger\n");
        } else if (foodArray[i] == PASTA) {
            printf("Pasta\n");
        } else if (foodArray[i] == SUSHI) {
            printf("Sushi\n");
        } else if (foodArray[i] == SALAD) {
            printf("Salad\n");
        }
    }

    return 0;
}
```

4 Stwórz typ wyliczeniowy Vehicle przechowujący rodzaje pojazdów. Następnie stwórz program zawierający tablicę 3 elementów typu Vehicle. Wypisz na konsoli zawartość tablicy używając pętli i instrukcji warunkowej.

```c
#include <stdio.h>

enum Vehicle {
    CAR,
    BIKE,
    BUS
};

int main() {
    enum Vehicle vehicleArray[3] = {CAR, BIKE, BUS};

    for (int i = 0; i < 3; i++) {
        if (vehicleArray[i] == CAR) {
            printf("Vehicle: Car\n");
        } else if (vehicleArray[i] == BIKE) {
            printf("Vehicle: Bike\n");
        } else if (vehicleArray[i] == BUS) {
            printf("Vehicle: Bus\n");
        }
    }

    return 0;
}
```

4 Napisz strukturę Laptop z polami model (tablica znaków długości 30) oraz pojemnoscDysku (typu int). Następnie napisz funkcję rozszerzPojemnosc, której argumentem jest wskaźnik do struktury typu Laptop. Funkcja ma dodać 500 do pola pojemnoscDysku w przekazanym argumencie. Stwórz przypadek testowy.

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

struct Laptop {
    char model[30];
    int pojemnoscDysku;
};

void rozszerzPojemnosc(struct Laptop* laptop) {
    laptop->pojemnoscDysku += 500;
}

int main() {
    struct Laptop laptop;
    strcpy(laptop.model, "Dell XPS 15");
    laptop.pojemnoscDysku = 1000;

    printf("Przed rozszerzeniem: Model: %s, Pojemność dysku: %d\n", laptop.model, laptop.pojemnoscDysku);

    rozszerzPojemnosc(&laptop);

    printf("Po rozszerzeniu: Model: %s, Pojemność dysku: %d\n", laptop.model, laptop.pojemnoscDysku);

    return 0;
}
```

4 Napisz strukturę Ksiazka z polami tytul (tablica znaków długości 50) oraz liczbaStron (typu int). Następnie napisz funkcję dodajRozdzial, której argumentem jest wskaźnik do struktury typu Ksiazka. Funkcja ma dodać 10 do pola liczbaStron w przekazanym argumencie. Stwórz przypadek testowy.

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

struct Ksiazka {
    char tytul[50];
    int liczbaStron;
};

void dodajRozdzial(struct Ksiazka* ksiazka) {
    ksiazka->liczbaStron += 10;
}

int main() {
    struct Ksiazka ksiazka;
    strcpy(ksiazka.tytul, "Wzorce projektowe");
    ksiazka.liczbaStron = 200;

    printf("Przed dodaniem rozdziału: Tytuł: %s, Liczba stron: %d\n", ksiazka.tytul, ksiazka.liczbaStron);

    dodajRozdzial(&ksiazka);

    printf("Po dodaniu rozdziału: Tytuł: %s, Liczba stron: %d\n", ksiazka.tytul, ksiazka.liczbaStron);

    return 0;
}
```

4 Napisz strukturę Komorka z polami numer (tablica znaków długości 15) oraz stanBaterii (typu int). Następnie napisz funkcję ladujBaterie, której argumentem jest wskaźnik do struktury typu Komorka. Funkcja ma dodać 20 do pola stanBaterii w przekazanym argumencie. Jeżeli po dodaniu wartość przekracza 100, stan baterii powinien być ustawiony na 100. Stwórz przypadek testowy.

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

struct Komorka {
    char numer[15];
    int stanBaterii;
};

void ladujBaterie(struct Komorka* komorka) {
    komorka->stanBaterii += 20;
    if (komorka->stanBaterii > 100) {
        komorka->stanBaterii = 100;
    }
}

int main() {
    struct Komorka komorka;
    strcpy(komorka.numer, "123456789");
    komorka.stanBaterii = 80;

    printf("Przed ładowaniem baterii: Numer: %s, Stan baterii: %d\n", komorka.numer, komorka.stanBaterii);

    ladujBaterie(&komorka);

    printf("Po ładowaniu baterii: Numer: %s, Stan baterii: %d\n", komorka.numer, komorka.stanBaterii);

    return 0;
}
```

4 Napisz strukturę Pralka z polami model (tablica znaków długości 30) oraz licznikPrania (typu int). Następnie napisz funkcję zrobPranie, której argumentem jest wskaźnik do struktury typu Pralka. Funkcja ma dodać 1 do pola licznikPrania w przekazanym argumencie. Stwórz przypadek testowy.

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

struct Pralka {
    char model[30];
    int licznikPrania;
};

void zrobPranie(struct Pralka* pralka) {
    pralka->licznikPrania += 1;
}

int main() {
    struct Pralka pralka;
    strcpy(pralka.model, "ABC123");
    pralka.licznikPrania = 0;

    printf("Przed praniem: Model: %s, Licznik prania: %d\n", pralka.model, pralka.licznikPrania);

    zrobPranie(&pralka);

    printf("Po praniu: Model: %s, Licznik prania: %d\n", pralka.model, pralka.licznikPrania);

    return 0;
}
```

4 Napisz strukturę KontoBankowe z polami numerKonta (tablica znaków długości 26) oraz saldo (typu double). Następnie napisz funkcję wplacPieniadze, której argumentem jest wskaźnik do struktury typu KontoBankowe oraz kwota, którą chcemy wpłacić. Funkcja ma dodać przekazaną kwotę do pola saldo w przekazanym argumencie. Stwórz przypadek testowy.

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

#define MAX_DLUGOSC_KONTA 26

struct KontoBankowe {
    char numerKonta[MAX_DLUGOSC_KONTA];
    double saldo;
};

void wplacPieniadze(struct KontoBankowe* konto, double kwota) {
    konto->saldo += kwota;
}

int main() {
    struct KontoBankowe konto;
    strcpy(konto.numerKonta, "12345678901234567890123456");
    konto.saldo = 1000.0;

    printf("Przed wpłata: Numer konta: %s, Saldo: %.2lf\n", konto.numerKonta, konto.saldo);

    double wplata = 500.0;
    wplacPieniadze(&konto, wplata);

    printf("Po wpłacie %.2lf: Numer konta: %s, Saldo: %.2lf\n", wplata, konto.numerKonta, konto.saldo);

    return 0;
}
```

4 Stwórz strukturę Osoba o trzech polach imie (napis), wiek (int), waga (float). Następnie stwórz funkcję, której argumentami jest tablica struktur Osoba oraz rozmiar tablicy. Funkcja ma zwrócić “osobę” (jako strukturę) o najmniejszym wieku. Stwórz przypadek testowy.

```c
#define MAX_DLUGOSC_IMIENIA 20
struct Osoba {
    char imie[MAX_DLUGOSC_IMIENIA];
    int wiek;
    float waga;
};
struct Osoba znajdzNajmlodszaOsobe(struct Osoba osoby[], int rozmiar) {
    struct Osoba najmlodsza = osoby[0];

    for (int i = 1; i < rozmiar; i++) {
        if (osoby[i].wiek < najmlodsza.wiek) {
            najmlodsza = osoby[i];
        }
    }

    return najmlodsza;
}
int main() {
    struct Osoba osoby[3];

    strcpy(osoby[0].imie, "Adam");
    osoby[0].wiek = 25;
    osoby[0].waga = 70.5;

    strcpy(osoby[1].imie, "Ewa");
    osoby[1].wiek = 32;
    osoby[1].waga = 65.2;

    strcpy(osoby[2].imie, "Jan");
    osoby[2].wiek = 20;
    osoby[2].waga = 80.0;

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

    struct Osoba najmlodszaOsoba = znajdzNajmlodszaOsobe(osoby, rozmiar);

    printf("Najmlodsza osoba: Imie: %s, Wiek: %d, Waga: %.2f\n", najmlodszaOsoba.imie, najmlodszaOsoba.wiek, najmlodszaOsoba.waga);

    return 0;
}
```

4 Napisz strukturę Pralka z polami model (tablica znaków długości 15) oraz licznikPrania (typu int). Następnie napisz funkcję zrobPranie, której argumentem jest wskaźnik do struktury typu Pralka. Funkcja ma dodać 1 do pola licznikPrania w przekazanym argumencie. Stwórz przypadek testowy.

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

#define MAX_DLUGOSC_MODEL 15

struct Pralka {
    char model[MAX_DLUGOSC_MODEL];
    int licznikPrania;
};

void zrobPranie(struct Pralka *pralka) {
    pralka->licznikPrania++;
}

int main() {
    struct Pralka pralka;
    strcpy(pralka.model, "ABC123");
    pralka.licznikPrania = 0;

    printf("Licznik prania przed: %d\n", pralka.licznikPrania);
    zrobPranie(&pralka);
    printf("Licznik prania po: %d\n", pralka.licznikPrania);

    return 0;
}
```

5. Napisz funkcję, która przyjmuje jako argument dwie listy z głową o elementach typu:

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

Funkcja zwraca 1 jeśli suma elementów na obu listach jest taka sama oraz 0 w przeciwnym wypadku. Stwórz przypadek testowy.

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

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

int sprawdzSumy(struct node *lista1, struct node *lista2) {
    int suma1 = 0, suma2 = 0;

    struct node *aktualny = lista1;
    while (aktualny != NULL) {
        suma1 += aktualny->t;
        aktualny = aktualny->next;
    }

    aktualny = lista2;
    while (aktualny != NULL) {
        suma2 += aktualny->t;
        aktualny = aktualny->next;
    }

    return (suma1 == suma2) ? 1 : 0;
}

void dodajElement(struct node **lista, int wartosc) {
    struct node *nowy = (struct node *)malloc(sizeof(struct node));
    nowy->t = wartosc;
    nowy->next = NULL;

    if (*lista == NULL) {
        *lista = nowy;
    } else {
        struct node *aktualny = *lista;
        while (aktualny->next != NULL) {
            aktualny = aktualny->next;
        }
        aktualny->next = nowy;
    }
}
```

```
}

void wypiszListe(struct node *lista) {
    struct node *aktualny = lista;
    while (aktualny != NULL) {
        printf("%d ", aktualny->t);
        aktualny = aktualny->next;
    }
    printf("\n");
}

void zwolnijListe(struct node *lista) {
    struct node *aktualny = lista;
    while (aktualny != NULL) {
        struct node *nastepny = aktualny->next;
        free(aktualny);
        aktualny = nastepny;
    }
}

int main() {
    struct node *lista1 = NULL;
    struct node *lista2 = NULL;

    dodajElement(&lista1, 1);
    dodajElement(&lista1, 2);
    dodajElement(&lista1, 3);

    dodajElement(&lista2, 4);
    dodajElement(&lista2, 5);
    dodajElement(&lista2, 6);

    wypiszListe(lista1);
    wypiszListe(lista2);

    int wynik = sprawdzSumy(lista1, lista2);

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

    zwolnijListe(lista1);
    zwolnijListe(lista2);

    return 0;
}
```

5. Napisz funkcję, która przyjmuje jako argument dwie listy bez głowy o elementach typu:

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

Funkcja zwraca 1 jeśli suma elementów na obu listach jest taka sama oraz 0 w przeciwnym wypadku. Stwórz przypadek testowy.

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

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

int sprawdzSumy(struct elem *lista1, struct elem *lista2) {
    int suma1 = 0, suma2 = 0;

    struct elem *aktualny1 = lista1;
    while (aktualny1 != NULL) {
        suma1 += aktualny1->t;
        aktualny1 = aktualny1->next;
    }

    struct elem *aktualny2 = lista2;
    while (aktualny2 != NULL) {
        suma2 += aktualny2->t;
        aktualny2 = aktualny2->next;
    }

    return (suma1 == suma2) ? 1 : 0;
}

void dodajElement(struct elem **lista, int wartosc) {
    struct elem *nowy = (struct elem *)malloc(sizeof(struct elem));
    nowy->t = wartosc;
    nowy->next = NULL;

    if (*lista == NULL) {
        *lista = nowy;
    } else {
        struct elem *aktualny = *lista;
        while (aktualny->next != NULL) {
            aktualny = aktualny->next;
        }
        aktualny->next = nowy;
    }
```

```
}

void wypiszListe(struct elem *lista) {
    struct elem *aktualny = lista;
    while (aktualny != NULL) {
        printf("%d ", aktualny->t);
        aktualny = aktualny->next;
    }
    printf("\n");
}

void zwolnijListe(struct elem *lista) {
    struct elem *aktualny = lista;
    while (aktualny != NULL) {
        struct elem *nastepny = aktualny->next;
        free(aktualny);
        aktualny = nastepny;
    }
}

int main() {
    struct elem *lista1 = NULL;
    struct elem *lista2 = NULL;

    dodajElement(&lista1, 1);
    dodajElement(&lista1, 2);
    dodajElement(&lista1, 3);

    dodajElement(&lista2, 4);
    dodajElement(&lista2, 5);
    dodajElement(&lista2, 6);

    wypiszListe(lista1);
    wypiszListe(lista2);

    int wynik = sprawdzSumy(lista1, lista2);

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


    zwolnijListe(lista1);
    zwolnijListe(lista2);

    return 0;
}
```

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

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

Funkcja ma zwiększyć o 5 każdy element znajdujący się na liście (dodać do pola **x** wartość 5). Stwórz przypadek testowy.

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

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

void zwiekszListe(struct elem *lista) {
    struct elem *aktualny = lista;
    while (aktualny != NULL) {
        aktualny->x += 5;
        aktualny = aktualny->next;
    }
}

void dodajElement(struct elem **lista, float wartosc) {
    struct elem *nowy = (struct elem *)malloc(sizeof(struct elem));
    nowy->x = wartosc;
    nowy->next = *lista;
    *lista = nowy;
}

void wypiszListe(struct elem *lista) {
    struct elem *aktualny = lista;
    while (aktualny != NULL) {
        printf("%.2f ", aktualny->x);
        aktualny = aktualny->next;
    }
    printf("\n");
}

void zwolnijListe(struct elem *lista) {
    struct elem *aktualny = lista;
    while (aktualny != NULL) {
        struct elem *nastepny = aktualny->next;
        free(aktualny);
        aktualny = nastepny;
    }
}

void zwolnijListe(struct elem *lista) {
    struct elem *aktualny = lista;
    while (aktualny != NULL) {
        struct elem *nastepny = aktualny->next;
        free(aktualny);
        aktualny = nastepny;
    }
}

int main() {
    struct elem *lista = NULL;

    dodajElement(&lista, 1.0);
    dodajElement(&lista, 2.5);
    dodajElement(&lista, 3.7);

    printf("Przed zwiekszeniem: ");
    wypiszListe(lista);

    zwiekszListe(lista);

    printf("Po zwiekszeniu: ");
    wypiszListe(lista);

    zwolnijListe(lista);

    return 0;
}
```

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

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

Funkcja ma podnieść do kwadratu każdy element znajdujący się na liście (podnieść pole a do kwadratu). Stwórz przypadek testowy.

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

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

void podniesDoKwadratu(struct elem *lista) {
    struct elem *aktualny = lista;
    while (aktualny != NULL) {
        aktualny->a = aktualny->a * aktualny->a;
        aktualny = aktualny->next;
    }
}

void dodajElement(struct elem **lista, int wartosc) {
    struct elem *nowy = (struct elem *)malloc(sizeof(struct elem));
    nowy->a = wartosc;
    nowy->next = *lista;
    *lista = nowy;
}

void wypiszListe(struct elem *lista) {
    struct elem *aktualny = lista;
    while (aktualny != NULL) {
        printf("%d ", aktualny->a);
        aktualny = aktualny->next;
    }
    printf("\n");
}

void zwolnijListe(struct elem *lista) {
    struct elem *aktualny = lista;
    while (aktualny != NULL) {
        struct elem *nastepny = aktualny->next;
        free(aktualny);
        aktualny = nastepny;
    }
}

int main() {
    struct elem *lista = NULL;

    dodajElement(&lista, 2);
    dodajElement(&lista, 5);
    dodajElement(&lista, 7);

    printf("Przed podniesieniem do kwadratu: ");
    wypiszListe(lista);

    podniesDoKwadratu(lista);

    printf("Po podniesieniu do kwadratu: ");
    wypiszListe(lista);

    zwolnijListe(lista);

    return 0;
}
```

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

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

oraz liczbę całkowitą a. Funkcja ma pomnożyć elementy nieparzyste na liście przez a. Stwórz jeden przypadek testowy.

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

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

void pomnozElementyNieparzyste(struct element *lista, int a) {
    struct element *aktualny = lista;
    while (aktualny != NULL) {
        if (aktualny->value % 2 != 0) {
            aktualny->value *= a;
        }
        aktualny = aktualny->next;
    }
}

void dodajElement(struct element **lista, int wartosc) {
    struct element *nowy = (struct element *)malloc(sizeof(struct element));
    nowy->value = wartosc;
    nowy->next = *lista;
    *lista = nowy;
}

void wypiszListe(struct element *lista) {
    struct element *aktualny = lista;
    while (aktualny != NULL) {
        printf("%d ", aktualny->value);
        aktualny = aktualny->next;
    }
    printf("\n");
}

void zwolnijListe(struct element *lista) {
    struct element *aktualny = lista;
    while (aktualny != NULL) {
        struct element *nastepny = aktualny->next;
        free(aktualny);
        aktualny = nastepny;

    }
}

int main() {
    struct element *lista = NULL;

    dodajElement(&lista, 1);
    dodajElement(&lista, 2);
    dodajElement(&lista, 3);
    dodajElement(&lista, 4);
    dodajElement(&lista, 5);

    printf("Przed mnożeniem przez a: ");
    wypiszListe(lista);

    int a = 3;
    pomnozElementyNieparzyste(lista, a);

    printf("Po mnożeniu przez a: ");
    wypiszListe(lista);

    zwolnijListe(lista);

    return 0;
}
```

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

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

oraz liczbę całkowitą a. Funkcja ma pomnożyć elementy nieparzyste na liście przez a. Stwórz jeden przypadek testowy.

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

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

void pomnozElementyNieparzyste(struct node *lista, int a) {
    struct node *aktualny = lista;
    while (aktualny != NULL) {
        if (aktualny->s % 2 != 0) {
            aktualny->s *= a;
        }
        aktualny = aktualny->next;
    }
}

void dodajElement(struct node **lista, int wartosc) {
    struct node *nowy = (struct node *)malloc(sizeof(struct node));
    nowy->s = wartosc;
    nowy->next = *lista;
    *lista = nowy;
}

void wypiszListe(struct node *lista) {
    struct node *aktualny = lista;
    while (aktualny != NULL) {
        printf("%d ", aktualny->s);
        aktualny = aktualny->next;
    }
    printf("\n");
}

void zwolnijListe(struct node *lista) {
    struct node *aktualny = lista;
    while (aktualny != NULL) {
        struct node *nastepny = aktualny->next;
        free(aktualny);
        aktualny = nastepny;

    }
}

int main() {
    struct node *lista = NULL;

    dodajElement(&lista, 1);
    dodajElement(&lista, 2);
    dodajElement(&lista, 3);
    dodajElement(&lista, 4);
    dodajElement(&lista, 5);

    printf("Przed mnożeniem przez a: ");
    wypiszListe(lista);

    int a = 3;
    pomnozElementyNieparzyste(lista, a);

    printf("Po mnożeniu przez a: ");
    wypiszListe(lista);

    zwolnijListe(lista);

    return 0;
}
```

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

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

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

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

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

int ileElementowRownych(struct node *lista, int b) {
    struct node *aktualny = lista;
    int licznik = 0;
    while (aktualny != NULL) {
        if (aktualny->val == b) {
            licznik++;
        }
        aktualny = aktualny->next;
    }
    return licznik;
}

void dodajElement(struct node **lista, int wartosc) {
    struct node *nowy = (struct node *)malloc(sizeof(struct node));
    nowy->val = wartosc;
    nowy->next = *lista;
    *lista = nowy;
}

void wypiszListe(struct node *lista) {
    struct node *aktualny = lista;
    while (aktualny != NULL) {
        printf("%d ", aktualny->val);
        aktualny = aktualny->next;
    }
    printf("\n");
}

void zwolnijListe(struct node *lista) {
    struct node *aktualny = lista;
    while (aktualny != NULL) {
        struct node *nastepny = aktualny->next;

        free(aktualny);
        aktualny = nastepny;
    }
}

int main() {
    struct node *lista = NULL;

    dodajElement(&lista, 1);
    dodajElement(&lista, 2);
    dodajElement(&lista, 3);
    dodajElement(&lista, 2);
    dodajElement(&lista, 4);

    printf("Lista: ");
    wypiszListe(lista);

    int b = 2;
    int ilosc = ileElementowRownych(lista, b);

    printf("Ilosc elementow rownych %d: %d\n", b, ilosc);

    zwolnijListe(lista);

    return 0;
}
```

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

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

oraz liczbę całkowitą `c`. Funkcja ma dodać do elementów ujemnych wartość `c`. Stwórz jeden przypadek testowy.

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

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

void dodajDoUjemnych(struct node *lista, int c) {
    struct node *aktualny = lista;
    while (aktualny != NULL) {
        if (aktualny->z < 0) {
            aktualny->z += c;
        }
        aktualny = aktualny->next;
    }
}

void dodajElement(struct node **lista, double wartosc) {
    struct node *nowy = (struct node *)malloc(sizeof(struct node));
    nowy->z = wartosc;
    nowy->next = *lista;
    *lista = nowy;
}

void wypiszListe(struct node *lista) {
    struct node *aktualny = lista;
    while (aktualny != NULL) {
        printf("%.2lf ", aktualny->z);
        aktualny = aktualny->next;
    }
    printf("\n");
}

void zwolnijListe(struct node *lista) {
    struct node *aktualny = lista;
    while (aktualny != NULL) {
        struct node *nastepny = aktualny->next;
        free(aktualny);
        aktualny = nastepny;

    }
}

int main() {
    struct node *lista = NULL;

    dodajElement(&lista, -1.5);
    dodajElement(&lista, 2.0);
    dodajElement(&lista, -3.7);
    dodajElement(&lista, 4.2);
    dodajElement(&lista, -5.9);

    printf("Lista przed: ");
    wypiszListe(lista);

    int c = 10;
    dodajDoUjemnych(lista, c);

    printf("Lista po: ");
    wypiszListe(lista);

    zwolnijListe(lista);

    return 0;
}
```

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

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

oraz liczbę całkowitą `d`. Funkcja ma odjąć od elementów dodatnich wartość `d`. Stwórz jeden przypadek testowy.

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

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

void odejmijOdDodatnich(struct node *lista, int d) {
    struct node *aktualny = lista;
    while (aktualny != NULL) {
        if (aktualny->num > 0) {
            aktualny->num -= d;
        }
        aktualny = aktualny->next;
    }
}

void dodajElement(struct node **lista, int wartosc) {
    struct node *nowy = (struct node *)malloc(sizeof(struct node));
    nowy->num = wartosc;
    nowy->next = *lista;
    *lista = nowy;
}

void wypiszListe(struct node *lista) {
    struct node *aktualny = lista;
    while (aktualny != NULL) {
        printf("%d ", aktualny->num);
        aktualny = aktualny->next;
    }
    printf("\n");
}

void zwolnijListe(struct node *lista) {
    struct node *aktualny = lista;
    while (aktualny != NULL) {
        struct node *nastepny = aktualny->next;
        free(aktualny);
        aktualny = nastepny;
    }
}

int main() {
    struct node *lista = NULL;

    dodajElement(&lista, -5);
    dodajElement(&lista, 3);
    dodajElement(&lista, -8);
    dodajElement(&lista, 7);
    dodajElement(&lista, 2);

    printf("Lista przed: ");
    wypiszListe(lista);

    int d = 4;
    odejmijOdDodatnich(lista, d);

    printf("Lista po: ");
    wypiszListe(lista);

    zwolnijListe(lista);

    return 0;
}
```

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

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

Funkcja ma wyświetlić w kolejnych wierszach te elementy listy, które są większe od wartości pierwszego elementu. W przypadku gdy lista ma mniej niż dwa elementy, to funkcja ma nic nie wyświetlać. Stwórz przypadek testowy.

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

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

void dodajElement(struct node **lista, int wartosc) {
    struct node *nowy = (struct node *)malloc(sizeof(struct node));
    nowy->val = wartosc;
    nowy->next = NULL;

    if (*lista == NULL) {
        *lista = nowy;
    } else {
        struct node *temp = *lista;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = nowy;
    }
}

void wypiszWieksze(struct node *lista) {
    if (lista == NULL || lista->next == NULL) {
        return;
    }

    int pierwszy = lista->val;
    struct node *aktualny = lista->next;
    while (aktualny != NULL) {
        if (aktualny->val > pierwszy) {
            printf("%d\n", aktualny->val);
        }
        aktualny = aktualny->next;
    }
}

void zwolnijListe(struct node *lista) {
    struct node *temp;
    while (lista != NULL) {
        temp = lista;
        lista = lista->next;
        free(temp);
    }
}

int main() {
    struct node *lista = NULL;

    dodajElement(&lista, 5);
    dodajElement(&lista, 3);
    dodajElement(&lista, 8);
    dodajElement(&lista, 2);
    dodajElement(&lista, 6);

    printf("Elementy większe od pierwszego:\n");
    wypiszWieksze(lista);

    zwolnijListe(lista);

    return 0;
}
```

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

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

Funkcja ma wyświetlić w kolejnych wierszach te elementy listy, które są większe niż pierwszy element. W przypadku gdy lista ma mniej niż dwa elementy, to funkcja ma nic nie wyświetlać. Stwórz przypadek testowy.

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

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

void printGreaterElements(struct elem* head) {
    if (head == NULL || head->next == NULL) {
        return; // Lista ma mniej niż dwa elementy, nie ma co wyświetlać
    }

    int firstElement = head->val;
    struct elem* current = head->next;

    while (current != NULL) {
        if (current->val > firstElement) {
            printf("%d\n", current->val);
        }
        current = current->next;
    }
}

int main() {
    // Tworzenie listy testowej
    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->val = 5;
    head->next = second;

    second->val = 8;
    second->next = third;

    third->val = 6;
    third->next = NULL;

    // Wywołanie funkcji i wyświetlenie wyników
    printGreaterElements(head);

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

    return 0;
}
```

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

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

Funkcja ma wyświetlić w kolejnych wierszach te elementy listy, które są mniejsze niż pierwszy element. W przypadku gdy lista ma mniej niż dwa elementy, to funkcja ma nic nie wyświetlać. Stwórz przypadek testowy.

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

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

void printSmallerElements(struct node* head) {
    if (head == NULL || head->next == NULL) {
        return; // Lista ma mniej niż dwa elementy, nie ma co wyświetlać
    }

    int firstElement = head->val;
    struct node* current = head->next;

    while (current != NULL) {
        if (current->val < firstElement) {
            printf("%d\n", current->val);
        }
        current = current->next;
    }
}

int main() {
    // Tworzenie listy testowej
    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->val = 5;
    head->next = second;

    second->val = 3;
    second->next = third;

    third->val = 7;
    third->next = NULL;

    // Wywołanie funkcji i wyświetlenie wyników
    printSmallerElements(head);

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

    return 0;
}
```

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

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

oraz liczbę całkowitą a. Funkcja wyświetlić te elementy listy, które są większa niż a. W przypadku braku takich elementów, funkcja ma nic nie robić. Stwórz jeden przypadek testowy.

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

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

void printGreaterElements(struct element* head, int a) {
    struct element* current = head->next;
    int found = 0; // Flaga informujaca, czy znaleziono elementy wieksze od a

    while (current != NULL) {
        if (current->value > a) {
            printf("%d\n", current->value);
            found = 1; // Ustawienie flagi, że znaleziono elementy wieksze od a
        }
        current = current->next;
    }

    if (!found) {
        printf("Brak elementów większych od %d\n", a);
    }
}

int main() {
    // Tworzenie listy testowej
    struct element* head = (struct element*)malloc(sizeof(struct element));
    struct element* second = (struct element*)malloc(sizeof(struct element));
    struct element* third = (struct element*)malloc(sizeof(struct element));

    head->next = second;

    second->value = 5;
    second->next = third;

    third->value = 7;
    third->next = NULL;

    // Wywołanie funkcji i wyświetlenie wyników
    int a = 4;
    printGreaterElements(head, a);

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

    return 0;
}
```

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

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

oraz liczbę całkowitą a. Funkcja wyświetlić te elementy listy, które są większa niż a. W przypadku braku takich elementów, funkcja ma nic nie robić. Stwórz jeden przypadek testowy.

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

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

void printGreaterElements(struct element* head, int a) {
    struct element* current = head;
    int found = 0; // Flaga informujaca, czy znaleziono elementy wieksze od a

    while (current != NULL) {
        if (current->value > a) {
            printf("%d\n", current->value);
            found = 1; // Ustawienie flagi, że znaleziono elementy wieksze od a
        }
        current = current->next;
    }

    if (!found) {
        printf("Brak elementów wiekszych od %d\n", a);
    }
}

int main() {
    // Tworzenie listy testowej
    struct element* head = NULL;
    struct element* current = NULL;

    // Dodawanie elementów do listy
    for (int i = 1; i <= 5; i++) {
        struct element* newElement = (struct element*)malloc(sizeof(struct element));
        newElement->value = i;
        newElement->next = NULL;

        if (head == NULL) {
            head = newElement;
            current = head;
        } else {
            current->next = newElement;
            current = current->next;
        }
    }

    // Wywołanie funkcji i wyświetlenie wyników
    int a = 3;
    printGreaterElements(head, a);

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

    return 0;
}
```

5. Napisz funkcję, która przyjmuje jako argument dwie listy z głową o elementach typu:

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

Funkcja zwraca 1 jeśli suma elementów parzystych na obu listach jest taka sama oraz 0 w przeciwnym wypadku. Stwórz przypadek testowy.

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

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

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

    // Obliczanie sumy elementów parzystych z listy 1
    struct node* current1 = head1;
    while (current1 != NULL) {
        if (current1->a % 2 == 0) {
            sum1 += current1->a;
        }
        current1 = current1->next;
    }

    // Obliczanie sumy elementów parzystych z listy 2
    struct node* current2 = head2;
    while (current2 != NULL) {
        if (current2->a % 2 == 0) {
            sum2 += current2->a;
        }
        current2 = current2->next;
    }

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

```c
int main() {
    // Tworzenie list testowych
    struct node* head1 = NULL;
    struct node* head2 = NULL;

    // Dodawanie elementów do listy 1
    for (int i = 1; i <= 5; i++) {
        struct node* newNode = (struct node*)malloc(sizeof(struct node));
        newNode->a = i;
        newNode->next = NULL;

        if (head1 == NULL) {
            head1 = newNode;
        } else {
            struct node* current = head1;
            while (current->next != NULL) {
                current = current->next;
            }
            current->next = newNode;
        }
    }

    // Dodawanie elementów do listy 2
    for (int i = 2; i <= 6; i++) {
        struct node* newNode = (struct node*)malloc(sizeof(struct node));
        newNode->a = i;
        newNode->next = NULL;

        if (head2 == NULL) {
            head2 = newNode;
        } else {
            struct node* current = head2;
            while (current->next != NULL) {
                current = current->next;
            }
            current->next = newNode;
        }
    }


    // Wywołanie funkcji i wyświetlenie wyniku
    int result = areEvenSumsEqual(head1, head2);
    printf("Czy suma elementów parzystych jest taka sama: %s\n", result ? "Tak" : "Nie");

    // Zwolnienie pamięci
    struct node* current = head1;
    while (current != NULL) {
        struct node* temp = current;
        current = current->next;
        free(temp);
    }

    current = head2;
    while (current != NULL) {
        struct node* temp = current;
        current = current->next;
        free(temp);
    }

    return 0;
}
```

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

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

oraz liczbę całkowitą a. Funkcja wyświetlić te elementy listy, które są mniejsze niż a. W przypadku braku takich elementów, funkcja ma nic nie robić. Stwórz jeden przypadek testowy.

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

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

void displayElementsLessThanA(struct element* head, int a) {
    int found = 0;

    struct element* current = head;
    while (current != NULL) {
        if (current->value < a) {
            printf("%d\n", current->value);
            found = 1;
        }
        current = current->next;
    }

    if (!found) {
        printf("Brak elementów mniejszych niż %d\n", a);
    }
}

int main() {
    // Tworzenie listy testowej
    struct element* head = NULL;

    // Dodawanie elementów do listy
    for (int i = 1; i <= 5; i++) {
        struct element* newElement = (struct element*)malloc(sizeof(struct element));
        newElement->value = i;
        newElement->next = NULL;

        if (head == NULL) {
            head = newElement;
        } else {
            struct element* current = head;
            while (current->next != NULL) {
                current = current->next;
            }
            current->next = newElement;
        }
    }

    // Wywołanie funkcji dla liczby a=3
    int a = 3;
    displayElementsLessThanA(head, a);

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

    return 0;
}
```