Lista bez głowy:

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

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

int main()
{
   // stwórz listę bez głowy o elementach 5,7,-4
   struct element * lista = malloc(sizeof(struct element));
   lista->i = 5;
   lista->next = malloc(sizeof(struct element));
   lista->next->i = 7;
   lista->next->next = malloc(sizeof(struct element));
   lista->next->next->i = -4;
   lista->next->next->next = NULL;
   //wyswietl listę
   struct element * wsk = lista;
   while(wsk != NULL)
   {
      printf("%d\n", wsk->i);
      wsk = wsk->next;
   }
   return 0;
}
```

Lista z głową:

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

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

int main()
{
   // stwórz listę z głową o elementach 5,7,-4
   struct element * lista = malloc(sizeof(struct element));
   lista->next = malloc(sizeof(struct element));
   lista->next->i = 5;
   lista->next->next = malloc(sizeof(struct element));
   lista->next->next->i = 7;
   lista->next->next->next = malloc(sizeof(struct element));
   lista->next->next->next->i = -4;
   lista->next->next->next->next = NULL;
   //wyswietl listę
   struct element * wsk = lista->next;
   while(wsk != NULL)
   {
      printf("%d\n", wsk->i);
      wsk = wsk->next;
   }
   return 0;
}
```

Wskaźniki na funkcję:

```
typ_zwracany (*nazwa_wsk)(typ1 arg1, typ2 arg2);
```

- Pobranie wartości (dereferencja, wyłuskiwanie) - *
- Pobranie adresu wskaźnika - &

Wskaźnik na stałą wartość, a stały wskaźnik:

Wskaźnik na stałą wartość:

```
const int *a;
int const * a;
```

Stały wskaźnik:

```
int * const b;
```

Stały wskaźnik na stałą wartość:

```
int const * const c
```

3 zestawy PIOJAS

Zestaw 1

Zad.2. Napisz funkcję porównująca dwie tablice jednowymiarowe o takich samych rozmiarach o wartościach typu int. Funkcja ma zwrócić 1 jeśli liczba elementów nieparzystych z każdej tablic z osobna są sobie równe, oraz ma zwrócić 0 w przeciwnym wypadku. Stwórz dwa przypadki testowe dla funkcji.

```
int zad2(int tab_1[],int tab_2[],int n){
   int liczba_1 = 0;
   int liczba_2 = 0;
   for(int i =0;i<n;i++){
      if(tab_1[i]%2 != 0){
         liczba_1++;
      }
      if(tab_2[i]%2 != 0){
         liczba_2++;
      }
   }
   if(liczba_1 == liczba_2){
      return 1;
   }
   return 0;
}

int main()
{
   //Zadanie 2
   int tab_1[] = {1,1,1,1,2};
   int tab_2[] = {1,1,1,1,3};
   printf("%d\n",zad2(tab_1,tab_2,5));
   int tab1[] = {1,1,1,1,2};
   int tab2[] = {1,1,1,1,4};
   printf("%d\n",zad2(tab1,tab2,5));
}
```

Zad.3. Stwórz typ wyliczeniowy Kwiat przechowujący gatunki kwiatów. Następnie stwórz program zawierający tablicę 5 elementów typu Kwiat. Wypisz na konsoli zawartość tablicy używając instrukcji warunkowej i pętli.

```
enum Kwiat{
   Mak,
   Roza,
   Tulipan
};

int main()
{
   //Zadanie 3
   enum Kwiat tab[] = {Mak,Roza,Tulipan};
   for(int i =0;i<3;i++){
      switch(tab[i]){
      case Mak:
         printf("Mak\n");
         break;
      case Roza:
         printf("Roza\n");
         break;
      case Tulipan:
         printf("Tulipan\n");
         break;
      default:
         printf("Nieznany Kwiat\n");
      }
    }
}
```

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

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

Funkcja ma zwrócić napis utworzony z liter przechowywanych na liście. Stwórz przypadek testowy.

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

char * zad4(struct node* lista){
   int len = 0;
   struct node* new_list = lista->next;

   while(new_list != NULL){
      len++;
      new_list = new_list->next;
   }
   char * result = malloc((len+1) * sizeof(char));
   int i = 0;
   new_list = lista->next;
   while(new_list != NULL){
      result[i] = new_list->z;
      i++;
      new_list = new_list->next;

   }
   result[len] = '\0';
   return result;
}

int main()
{
   struct node* lista = malloc(sizeof(struct node));
   lista->next = malloc(sizeof(struct node));
   lista->next-> z = 'a';
   lista->next->next = malloc(sizeof(struct node));
   lista->next->next->z = 'n';
   lista->next->next->next = malloc(sizeof(struct node));
   lista->next->next->next->z = 'd';
   lista->next->next->next ->next= malloc(sizeof(struct node));
   lista->next->next->next ->next->z =  'r';
   lista->next->next->next ->next->next= malloc(sizeof(struct node));
   lista->next->next->next ->next->next->z = 'i';
   lista->next->next->next ->next->next->next = malloc(sizeof(struct node));
   lista->next->next->next ->next->next->next->z = 'i';
   lista->next->next->next ->next->next->next->next = NULL;

   char * napiss = zad4(lista);
   printf("%s\n",napiss);

   struct node*wsk =lista->next;
   while(wsk!= NULL){
      struct node* temp =wsk;
      wsk = wsk->next;
      free(temp);
   }
   free(lista);
   free(napiss);
   printf("Hello world!\n");
   return 0;
}
```

Zestaw 2

Zad.1. Dane są następujące wyrazy i znaki:
int void int fun w w2 * , * ( )
Ułóż je we właściwej kolejności, aby otrzymać nagłówek funkcji fun, której argumentami są
dwa wskaźniki. Następnie dodaj dowolną implementację funkcji i stwórz dla niej przypadek testowy.

```
void fun(int *wsk,int *wsk2){
   printf("%d\n",*wsk + *wsk2);
}

int main()
{
   //Zadani1
   int a =6;
   int b =8;
   fun(&a,&b);
}
```

Zad.2. Napisz funkcję, której argumentem jest liczba całkowita przekazana jako napis. Funkcja ma zwrócić sumę cyfr nieparzystych liczby przekazanej jako argument funkcji. Stwórz
przypadek testowy.
Przykład: dla “34821” ma być zwrócone 4.

```
int zad2(char * tab){
   int result =0;
   for(int i=0;tab[i] != '\0';i++){
         int cyfra =tab[i] - '0';
         if( cyfra %2 != 0){
            result += cyfra;
         }
   }
   return result;
}

int main()
{
   //Zadanie 2
   char napiss[] = "1216181";
   printf("%d\n",zad2(napiss));
}
```

Zad.3. Stwórz strukturę Kierowca o dwóch polach imie (napis) oraz rekord (dowolny typ całkowity). Następnie stwórz funkcję, której argumentami jest tablica struktur Kierowca oraz rozmiar tablicy. Funkcja ma zwrócić imię kierowcy z najgorszym (najmniejszym licz bowo) wynikiem (w przypadku kilku równych wyników, ma zwrócić wynik ostatniego). Stwórz przypadek testowy.

```
struct Kierowca{
   char *imie;
   int record;
};

char * zad3(struct Kierowca tab[],int n){
   struct Kierowca tmp = tab[0];
   int minn = tmp.record;
   char * wynik = NULL;
   for(int i=0;i<n;i++){
      struct Kierowca tmp = tab[i];
      if(tmp.record < minn){
         wynik = malloc(sizeof(char) * (strlen(tmp.imie)+1) );
         for(int j=0;tmp.imie[j] != '\0';j++){
            wynik[j] = tmp.imie[j];
         }
         wynik[strlen(tmp.imie)] = '\0';
      }
   }
   return wynik;
}

int main()
{
   struct Kierowca tab[] = {{"Andrii",5},{"KillReal",3},{"Denys",1}};
   char * wynikk = zad3(tab,3);
   if(wynikk != NULL){
      printf("%s\n",wynikk);
      free(wynikk);   }
}
```

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

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

Funkcja ma zwrócić sumę elementów z drugiej listy większych niż maksimum pierwszej listy.
Stwórz przypadek testowy

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

int zad4(struct element * lista_1,struct element * lista_2){
   struct element* wsk1 = lista_1->next;
   int max = wsk1->x;
   while (wsk1 != NULL) {
      if(max < wsk1->x){
         max = wsk1->x;
      }
      wsk1 = wsk1->next;
   }
   int summa =0;
   struct element* wsk2 = lista_2->next;
   while (wsk2 != NULL) {
      if(wsk2->x > max){
         summa += wsk2->x;
      }
      wsk2 = wsk2->next;
   }

   return summa;
}

int main()
{
   struct element * lista_1 = malloc(sizeof(struct element));
   lista_1->next = malloc(sizeof(struct element));
   lista_1->next->x = 5;
   lista_1->next->next = malloc(sizeof(struct element));
   lista_1->next->next->x = 12;
   lista_1->next->next->next = NULL;

   struct element * lista_2 = malloc(sizeof(struct element));
   lista_2->next = malloc(sizeof(struct element));
   lista_2->next->x = 15;
   lista_2->next->next = malloc(sizeof(struct element));
   lista_2->next->next->x = -3;
   lista_2->next->next->next = malloc(sizeof(struct element));
   lista_2->next->next->next->x = 16;
   lista_2->next->next->next->next = NULL;

   printf("%d\n",zad4(lista_1,lista_2));
   return 0;
}
```

Zestaw 3

Zad.1. Dane są następujące wyrazy i znaki:
void int float const foo a b ( ) , *
Ułóż je we właściwej kolejności, aby otrzymać nagłówek funkcji foo, której argumentami są kolejno wskaźnik na stałą wartość int oraz liczba wymierna. Następnie dodaj dowolną implementację funkcji i stwórz dla niej przypadek testowy.

```
void foo(float a,const int * b){
   printf("%lf\n",a+*b);
}

int main()
{
   float a = 5.5;
   const int b =5;
   foo(a,&b);
}
```

Zad.2. Napisz funkcję porównująca dwie tablice jednowymiarowe o takich samych rozmiarach o wartościach typu int. Funkcja ma zwrócić 1 jeśli sumy elementów parzystych z każdej tablic z osobna są sobie równe, oraz ma zwrócić 0 w przeciwnym wypadku. Stwórz dwa przypadki testowe dla funkcji.

```
int zad2(int tab1[],int tab2[],int n){
   for(int i=0;i<n;i++){
      if(tab1[i] != tab2[i]){
         return 0;
      }
   }
   return 1;
}

int main()
{
   //Zadanie 2;
   int tab1[] = {1,2,3,4,5};
   int tab2[] = {1,2,3,4,5};
   int n =5;
   printf("%d\n",zad2(tab1,tab2,n));
}
```

Zad.3. Napisz strukturę Album z polami nazwa (tablica znaków długości 100) oraz liczba_utworow (typu int). Następnie napisz funkcje i wywołaj każdą z nich co najmniej jeden raz (upewniając się, że to możliwe):

a) initAlbum - funkcja przyjmuje dwa argumenty: nazwę albumu i liczbę utworów, i zwraca wskaźnik na nowo-utworzoną strukturę ustawiającą składowe z przekazanych argumen tów. Dodatkowo funkcja powinna sprawdzić, aby nazwa albumu była napisem długości co najmniej 3 i liczba utworów większa niż 1. W przypadku nie spełnienia jednego z warunków, funkcja powinna zwracać NULL.

b) dodajUtwory - funkcja, której argumentem jest wskaźnik do struktury typu Album. Funk cja ma dodać 5 do liczby utworów w przekazanym argumencie.

```
struct Album * initAlbum(char nazwa[],int liczba_utworow){
   if(strlen(nazwa) < 3 ||  liczba_utworow < 1){
      return NULL;
   }
   struct Album* t = malloc(sizeof(struct Album));
   for (int i = 0; i < strlen(nazwa); i++) {
      t->nazwa[i] = nazwa[i];
   }
   t->liczba_utworow = liczba_utworow;
   return t;
};
struct Album * dodajUtwory(struct Album * t){
   t->liczba_utworow += 5;
   return t;
};

int main()
{
   //Zadanie 3
   char nazwa[] = "Huesos";
   int liczba_utworow = 10;
   printf("%p\n",initAlbum(nazwa,liczba_utworow);
   struct Album t;
   for (int i = 0; i < strlen(nazwa); i++) {
      t.nazwa[i] = nazwa[i];
   }
   t.liczba_utworow = 4;
   printf("%p\n",dodajUtwory(&t));
   printf("%d\n",t.liczba_utworow);
   }
```

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

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

Funkcja ma połączyć obie listy (najpierw ma być pierwszy argument, potem drugi) zachowując głowę pierwszej listy. Funkcja ma zwrócić wskaźnik na nową listę. Stwórz przypadek testowy.
Przykład:

Lista1:
- głowa: adres 0400
- adres 0020, wartość:5
- adres 0060, wartość:12

Lista2:
- głowa: adres 0110
- adres 0130, wartość:54
- adres 0150, wartość:-3
- adres 0170, wartość:11

Lista po połączeniu:
- głowa: adres 0400
- adres 0020, wartość:5
- adres 0060, wartość:12
- adres 0130, wartość:54
- adres 0150, wartość:-3
- adres 0170, wartość:11

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


struct element* mergeLists(struct element* list1, struct element* list2) {
   if (list1 == NULL) {
      return list2;
   }
   if (list2 == NULL) {
      return list1;
   }
   struct element* current = list1;
   while (current->next != NULL) {
      current = current->next;
   }
   current->next = list2;
   return list1;
}

int main()
{
   //Zadanie 4
   struct element * lista_1 = malloc(sizeof(struct element));
   lista_1->next = malloc(sizeof(struct element));
   lista_1->next->a = 5;
   lista_1->next->next = malloc(sizeof(struct element));
   lista_1->next->next->a = 12;
   lista_1->next->next->next = NULL;

   struct element * lista_2 = malloc(sizeof(struct element));
   lista_2->next = malloc(sizeof(struct element));
   lista_2->next->a = 54;
   lista_2->next->next = malloc(sizeof(struct element));
   lista_2->next->next->a = -3;
   lista_2->next->next->next = malloc(sizeof(struct element));
   lista_2->next->next->next->a = 11;
   lista_2->next->next->next->next = NULL;

   struct element* mergedList = mergeLists(lista_1, lista_2);
   struct element* current = mergedList;
   while (current != NULL) {
      printf("%d ", current->a);
      current = current->next;
   }
   printf("\n");
}
```

ADDITION

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

// Структура для представления узла списка
struct ListNode {
    char value;
    struct ListNode* next;
};

// Функция для склеивания букв в слово
char* concatenate_letters(struct ListNode* head) {
    // Проверка на пустой список
    if (head == NULL) {
        return NULL;
    }

    // Вычисление размера списка
    int size = 0;
    struct ListNode* current = head;
    while (current != NULL) {
        size++;
        current = current->next;
    }

    // Выделение памяти для хранения слова
    char* word = (char*)malloc((size + 1) * sizeof(char));
    word[size] = '\0';  // Установка символа конца строки

    // Копирование букв из списка в слово
    current = head;
    int i = 0;
    while (current != NULL) {
        word[i] = current->value;
        i++;
        current = current->next;
    }

    return word;
}

int main() {
    // Создание списка из букв "H", "e", "l", "l", "o"
    struct ListNode* head = (struct ListNode*)malloc(sizeof(struct ListNode));
    head->value = 'H';

    struct ListNode* node2 = (struct ListNode*)malloc(sizeof(struct ListNode));
    node2->value = 'e';

    struct ListNode* node3 = (struct ListNode*)malloc(sizeof(struct ListNode));
    node3->value = 'l';

    struct ListNode* node4 = (struct ListNode*)malloc(sizeof(struct ListNode));
    node4->value = 'l';

    struct ListNode* node5 = (struct ListNode*)malloc(sizeof(struct ListNode));
    node5->value = 'o';

    head->next = node2;
    node2->next = node3;
    node3->next = node4;
    node4->next = node5;
    node5->next = NULL;

    // Склеивание букв в слово
    char* result = concatenate_letters(head);
    printf("%s\n", result);  // Выводит "Hello"

    // Освобождение памяти
    free(result);
    free(node5);
    free(node4);
    free(node3);
    free(node2);
    free(head);

    return 0;
}
```

Добавление элемента в начало списка без головы в int main()

```
struct element * lista = malloc(sizeof (struct element));
   lista->i = 5;
   lista->next = malloc(sizeof (struct element));
   lista->next->i = 7;
   lista->next->next = malloc(sizeof (struct element));
   lista->next->next->i = -4;
   lista->next->next->next = NULL;
   struct element * temp = lista;
   while (temp != NULL)
   {
      printf("%d\n", temp->i);
      temp = temp->next;
   }
   printf("dodanie\n");
   struct element * wsk = malloc(sizeof (struct element));
   wsk->i = 11;
   wsk->next = lista;
   lista = wsk;
   temp = lista;
   while (temp != NULL)
   {
      printf("%d\n", temp->i);
      temp = temp->next;
   }
```

Создание указателя на функцию через typedef

```
#define PTR INT int*

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

typedef int (*PTR_EVEN) (int);

int main(void)
{
   PTR_EVEN func_even = is_even;
   printf("%d\n", func_even (2)) ;
   printf ("%d\n", func_even (3)) ;
   return 0;
}
```

Реверс слова без его замены и использования библиотек

```
char * reverseString(char * a, int size){
   char * b = malloc(sizeof(char)*size);

   int j = PIDOR;

   for(int i = size-1; i >= 0; i--){
      b[j] = a[i];
      j++;
   }

   return b;
}
   char word[] = "hello";
   int size = 5;
   printf("%s\n", reverseString(word, size));
   return 0;
}
```

Удаление четных элементов из списка без головы

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

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

// Function to remove even elements from the list
void remove_even(elem **start) {
   elem *current = *start;
   elem *previous = NULL;

   while (current != NULL) {
      if (current->x % 2 == 0) {
         if (previous == NULL) {
            // If the first element is even
            *start = current->next;
            free(current);
            current = *start;
         } else {
            // If an even element is in the middle of the list
            previous->next = current->next;
            free(current);
            current = previous->next;
         }
      } else {
         previous = current;
         current = current->next;
      }
   }
}


int main() {
   elem *list = malloc(sizeof(elem));
   list->x = 5;
   list->next = malloc(sizeof(elem));
   list->next->x = 12;
   list->next->next = malloc(sizeof(elem));
   list->next->next->x = 13;
   list->next->next->next = malloc(sizeof(elem));
   list->next->next->next->x = 16;
   list->next->next->next->next = NULL;

   // Call the function to remove even elements
   remove_even(&list);

   // Print the remaining elements of the list
   elem *current = list;
   while (current != NULL) {
      printf("%d ", current->x);
      current = current->next;
   }

   return 0;
}
```

Удаление отрицательных элементов из списка без головы

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

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

void remove_minus(struct node **lista) {
   struct node *current = *lista;
   struct node *previous = NULL;

   while (current != NULL) {
      if (current->value < 0) {
         if (previous == NULL) {
            *lista = current->next;
            free(current);
            current = *lista;
         } else {
            previous->next = current->next;
            free(current);
            current = previous->next;
         }
      } else {
         previous = current;
         current = current->next;
      }
   }
}

int main()
{
   struct node * lista = malloc(sizeof(struct node));
   lista -> value = 5;
   lista -> next = malloc(sizeof(struct node));
   lista -> next -> value = 6;
   lista -> next -> next = malloc(sizeof(struct node));
   lista -> next -> next -> value = -3;
   lista -> next -> next -> next = malloc(sizeof(struct node));
   lista -> next -> next -> next -> value = -2;
   lista -> next -> next -> next -> next = NULL;

   remove_minus(&lista);

   struct node * current = lista;
   while (current != NULL) {
      printf("%f  ", current->value);
      current = current->next;
   }

   return 0;
}
```

Zad.3. Napisz funkcję swapColumns, która przyjmuje jako argumenty dwuwymiarową tablicę tablic liczb całkowitych, jej wymiary oraz dwa indeksy kolumn **do** zamiany miejscami. Funkcja powinna przestawić wskazane kolumny i zwrócić zmodyfikowaną tablicę. Uwzględnij sytuację, jeśli podane indeksy są nieprawidłowe - wtedy funkcja ma nic nie robić. Stwórz przypadek testowy dla funkcji.

```
// Funkcja do zamiany dwóch kolumn w tablicy dwuwymiarowej
void swapColumns(int **array, int rows, int cols, int col1, int col2) {

    for (int i = 0; i < rows; i++) {
        int temp = array[i][col1];
        array[i][col1] = array[i][col2];
        array[i][col2] = temp;
    }
}

int main(void)
{




    int rows = 3;
    int cols = 4;

    // Tworzenie przykładowej tablicy dwuwymiarowej
    int **array = (int **)malloc(rows * sizeof(int *));
    for (int i = 0; i < rows; i++) {
        array[i] = (int *)malloc(cols * sizeof(int));
        for (int j = 0; j < cols; j++) {
            array[i][j] = i * cols + j + 1;
        }
    }

    // Wyświetlenie oryginalnej tablicy
    printf("Oryginalna tablica:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%2d ", array[i][j]);
        }
        printf("\n");
    }

    int col1 = 1; // Pierwsza kolumna do zamiany
    int col2 = 3; // Druga kolumna do zamiany

    // Wywołanie funkcji swapColumns
    swapColumns(array, rows, cols, col1, col2);

    // Wyświetlenie zmodyfikowanej tablicy po zamianie kolumn
    printf("\nTablica po zamianie kolumn %d i %d:\n", col1, col2);
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%2d ", array[i][j]);
        }
        printf("\n");
    }

    // Zwolnienie pamięci
    for (int i = 0; i < rows; i++) {
        free(array[i]);
    }
    free(array);
    return 0;
}
```

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

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

zwraca adres ostatniej wartości ujemnej na liście. W przypadku pustej listy lub braku elementów ujemnych, funkcja ma zwrócić NULL. Stwórz jeden przypadek testowy.

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

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

float zad4(struct node * lista, int n){
    float result = -1;
    struct node * wsk = lista;
    while (wsk!=NULL) {
        if(wsk->value >= 0){
            result = wsk->value;
        }
        wsk = wsk->next;
    }
    if(result < 0){
        return 0;
    }
    return result;
}

int main(void)
{
    struct node *lista = malloc(sizeof(struct node));
    lista->value = 10.5;
    lista->next = malloc(sizeof(struct node));
    lista->next->value = 11;
    lista->next->next = malloc(sizeof(struct node));
    lista->next->next->value = -10.3;
    lista->next->next->next = malloc(sizeof(struct node));
    lista->next->next->next->value = 10.77;

    printf("%lf\n", zad4(lista, 4));
    return 0;
}
```

Нахождение самого маленького числа стоящего на нечетном индексе двухмерной таблицы

```
int foo(int tab[][100], int n, int m){
    int min_element = tab[1][1];
    for(int i = 1; i < n; i+=2){
        for(int j = 1; j < m; j+=2){
            if(min_element > tab[i][j]){
                min_element = tab[i][j];
            }
        }
    }
    return min_element;
}

int main()
{
    int n = 5; // Rows
    int m = 4; // Columns
    int array[5][100] = {
        {10, 23, 5, 18},
        {7, 12, 9, 14},
        {30, 8, 17, 6},
        {11, 22, 13, 20},
        {15, 25, 4, 19}
    };
    int result = foo(array, n, m);
    printf("The smallest element on odd indices is: %d\n", result);
    return 0;
}
```

Использования указателя на функцию и удаление последнего элемента в масссиве удолетворяющего условию второй функции

```
int findWithCondition(int arr[], int size, int (*condition)(int)) {
    int index = -1;  // Индекс найденного элемента, -1 если не найдено

    for (int i = 0; i < size; i++) {
        if (condition(arr[i]) == 1) {
            index = i;  // Запоминаем индекс, если элемент удовлетворяет условию
        }
    }

    return index;
}

// Пример функции condition: возвращает 1, если число четное, иначе 0
int condition(int num) {
    return num % 2 == 0 ? 1 : 0;
}

int main() {
    int array[] = {1, 3, 4, 6, 7, 9, 10};
    int size = sizeof(array) / sizeof(array[0]);

    int result = findWithCondition(array, size, condition);

    if (result != -1) {
        printf("last index: %d\n", result);
    } else {
        printf("nothing.\n");
    }

    return 0;
}
```

Вознесение в квадрат первой переменной если вторая равна 0

```
int copy_squared_value_protected(const int *a, int *b) {
    if (*b == 0) {
        *b = *a * *a;
        return *b;
    } else {
        return *a;
    }
}

int main() {
    const int a = 66;
    int b = 0;
    printf("result: %d", copy_squared_value_protected(&a, &b));
    return 0;
}
```

Typ weliczyniowy (enum) который выводит себя рекурсивно

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

enum Day {
    pn,
    wt,
    sr,
    cht,
    pt,
    sb,
    ws
};

void print_days(enum Day day, int n) {
    if (n <= 0) {
        return;
    }
    const char *day_names[] = {
        "pn",
        "wt",
        "sr",
        "cht",
        "pt",
        "sb",
        "ws"
    };

    printf("runnin thru days: %s\n", day_names[day]);
    enum Day next_day = (day + 1) % 7;
    print_days(next_day, n - 1);
}

int main() {
    enum Day starting_day = sr;
    int n = 33;
    print_days(starting_day, n);
    return 0;
}
```

Проверка стринга на наличие int и конвертация стринга в int

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

int suma(int a, char tab[][a]) {
    int result = 0;
    for (int i = 0; i < a; i++) {
        for (int j = 0; j < a; j++) {
            if (isdigit(tab[i][j]) == 1) {
                result += tab[i][j]  - '0';
            }
        }
    }
    return result;
}

int main(int argc, const char * argv[]) {
    char tab[3][3] = {{'1', '1', '1'}, {'1', '1', '1'}, {'1', '1', '1'}};
    int a = 3;
    printf("%d\n", suma(a, tab));
    return 0;
}
```

Нахождение подслова в другом слове

```
int findSubstring(char *str, char *sub) {
    for (int i = 0; str[i] != '\0'; i++) {
        if (str[i] == sub[0]) {
            int k = 0;
            int j = i;  // Используем отдельную переменную j для обхода подстроки
            while (str[j] == sub[k] && sub[k] != '\0') {
                j++;
                k++;
            }
            if (sub[k] == '\0') {
                return 1;  // Подстрока найдена
            }
        }
    }
    return 0;  // Подстрока не найдена
}
int main(int argc, const char * argv[]) {
    char str[] = "Hello, world!";
    char sub[] = "world";

    int found = findSubstring(str, sub);

    if (found == 1) {
        printf("Substring found.\n");
    } else if (found == 0) {
        printf("Substring not found.\n");
    } else {
        printf("Error occurred during search.\n");
    }

    return 0;
}
```

Переворот нечетных columns

```
void foo(int n, int m, int arr[n][m]){
    for(int i = 1; i < n; i++){
        int start = 0;
        int end = m-1;
}
        while (start < end) {
            // Обмен элементов
            int temp = arr[start][i];
            arr[start][i] = arr[end][i];
            arr[end][i] = temp;

            // Переход к следующей паре элементов
            start++;
            end--;
        }
    }
}
int main(int argc, const char * argv[]) {
    int rows = 5;
    int cols = 5;
    int table[5][5] = {
        {1, 2, 3, 4, 5},
        {6, 7, 8, 9, 10},
        {11, 12, 13, 14, 15},
        {16, 17, 18, 19, 20},
        {21, 22, 23, 24, 25}
    };
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%2d ", table[i][j]);
        }
        printf("\n");
    }
    printf("\n");
    foo(rows, cols, table);
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%2d ", table[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Резервация 4 int, и цикл с конца по таблице

```
int * init_block_int(void){
    int * tab = malloc(sizeof(sizeof(int)*4));
    tab[0]=1;
    tab[1]=2;
    tab[2]=3;
    tab[3]=25;
    return &tab[3];
}

int main(void) {
    int a = 9;
    int result = foo(a, fun);
    printf("%d\n", result);
    int * tab = init_block_int();
    for(int i = 3; i >= 0; i--){
        printf("%d\n", *(tab -i));
    }
    return 0;
}
```

Фунцкия добавления элемента в начало в листе без головы

```
struct element* dodaj (struct element*Lista, int a)
{
    struct element * wsk = malloc (sizeof(struct element));
    wsk-›i=a;
    wsk-›next=Lista;
    return wsk;
}
```

Использование двух указателей на функцию и сравнение их результатов

```
int cc(double x){
    int temp = 0;
    while (temp < x) {
        temp++;
    }
    return temp-1;
}

int foo(double(*f1)(double), double(*f2)(double), double x){
    if(x<1)
        return -1;
    int n = cc(x);
    for(int i = 0; i < n; i++){
        if (f1(i) != f2(i)*f2(i)*f2(i)){
            return -1;
        }
    }
    return 1;
}

double pom1(double x){
    return x*x*x;
}

double pom2(double x){
    return ((int)x)%3;
}

int main(void) {
    printf("%d\n", cc(2));
    printf("%d\n", foo(pom1, pom2, 2));
    printf("%d\n", foo(pom1, pom2, 5));
    return 0;
}
```

Пример инициализации структуры и работа с ней

```
struct album{
   char nazwa[100];
   int liczba_utworów;
};

struct album * initAlbum(char asd[100], int bbs){
   struct album * temp = malloc(sizeof(struct album));
   if(strlen(asd)<3 || bbs < 3){
      return NULL;
   }
   strcpy(temp->nazwa, asd);
   temp->liczba_utworów = bbs;
   return temp;
}

void dodaj_utwory(struct album * arg){
   arg->liczba_utworów += 5;
}

int main(void) {
   struct album * al1 = initAlbum("ABC", 10);
   if(al1 != NULL){
      printf("%s %d\n", al1->nazwa, al1->liczba_utworów);
      dodaj_utwory(al1);
      printf("%s %d\n", al1->nazwa, al1->liczba_utworów);
   }
   return 0;
}
```

Пример работы с union

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

union XYZ{
   int a;
   char b;
};

int main(void) {

   union XYZ xyz[6];
   xyz[0].a = 77;
   xyz[1].b = 'a';
   xyz[2].a = 11;
   xyz[3].b = 'c';
   xyz[4].a = 67;
   xyz[5].b = 'a';

   for(int i = 0; i < 6; i++){
      printf("zyz[%d].a = %d\n", i, xyz[i].a);
      printf("zyz[%d].b = %d\n", i, xyz[i].b);
   }

   return 0;
}
```

Замена элементов списка без головы местами

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

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

void foo(struct element * lista)
{
   struct element * wsk = lista->next;
   struct element * wsk2 = lista->next->next;
   struct element * wsk3 = lista->next->next->next;
   lista->next = wsk2;
   wsk2->next = wsk;
   wsk->next = wsk3;
}
```

// ВЫВОД СПИСКА В КОНСОЛЬ

```
void wyswieltListeBezGlowy(struct element * lista)
{
   struct element * wsk=lista;
   while(wsk!=NULL)
   {
      printf("%d\n", wsk->i);
      wsk=wsk->next;
   }
   printf("---\n");
}

int main()
{
   struct element * lista = malloc(sizeof(struct element));
   lista->i = 7;
   lista->next = malloc(sizeof(struct element));
   lista->next->i = 8;
   lista->next->next = malloc(sizeof(struct element));
   lista->next->next->i = -3;
   lista->next->next->next = malloc(sizeof(struct element));
   lista->next->next->next->i = 20;
   lista->next->next->next->next = NULL;
   wyswieltListeBezGlowy(lista);
   foo(lista);
   wyswieltListeBezGlowy(lista);
   return 0;
}
```

Сдвиг элементов списка с головой и установка послденего элемента списка в начало

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

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

void foo(struct element * lista)
{
   if (lista->next == NULL || lista->next->next == NULL)
      return;
   struct element * wsk = lista->next;
   struct element * wsk2 = lista->next;
   while(wsk2->next->next != NULL)
   {
      wsk2=wsk2->next;
   }
   struct element * wsk3 = wsk2->next;
   lista->next= wsk3;
   wsk3->next = wsk;
   wsk2->next = NULL;
}

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

int main()
{
   struct element * lista = malloc(sizeof(struct element));
   lista->next = malloc(sizeof(struct element));
   lista->next->i = 2;
   lista->next->next = malloc(sizeof(struct element));
   lista->next->next->i = -4;
   lista->next->next->next = NULL;
   wyswieltListeZG(lista);
   foo(lista);
   wyswieltListeZG(lista);
   return 0;
}
```

Рекурсия в enum (последовательный вывод)

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

enum DAY  { PON, WT, SR, CZW, PT, SO, ND};

void printdays(enum DAY arg1, int n)
{
   if(n>0)
   {
      if (arg1 == PON)
      {
         printf("PON\n");
      }
      else if (arg1 == WT)
      {
         printf("WT\n");
      }
      else if (arg1 == SR)
      {
         printf("SR\n");
      }
      else if (arg1 == CZW)
      {
         printf("CZW\n");
      }
      else if (arg1 == PT)
      {
         printf("PT\n");
      }
      else if (arg1 == SO)
      {
         printf("SO\n");
      }
      else if (arg1 == ND)
      {
         printf("ND\n");
      }

      n--;
      arg1++;
      if(arg1 >6)
      {
         arg1 -=7;
      }
      printdays(arg1, n);
   }
}

int main()
{
   printdays(PON, 10);
   return 0;
}
```

Работа с whar_t и L, проверка массива символов на содержание int либо string

```
int zad3(wchar_t *tab) {
   int isAllDigits = 1;
   for (int i = 0; i < 3; i++) {
      if (tab[i] < L'0' || tab[i] > L'9') {
         isAllDigits = 0;
         break;
      }
   }
   return isAllDigits;
}

int main(void) {
   wchar_t x[] = L"7214";
   printf("%d\n", zad3(x));
   return 0;
}
```

Освобождение памяти

(универсальный)

```
struct node *current2 = lista;
while (current2 != NULL) {
   struct node *temp = current2;
   current2 = current2->next;
   free(temp);
}
```

Вывод списка без головы в консоль

(универсальный)

```
struct node * current = lista;
while (current != NULL) {
   printf("%d ", current->a);
   current = current->next;
}
```

Замена нечетных чисел на нули в списке без головы

```
void foo(struct node * lista) {
   struct node * wsk = lista;
   while (wsk != NULL) {
      if (wsk->a % 2 != 0) {
         wsk->a = 0;
      }
      wsk = wsk -> next;
   }
}
```

Замена местами первого и последнего элемента в списке с головой

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

void foo(struct element * lista)
{
if (lista->next == NULL || lista->next->next == NULL) {
return;
}
struct element * first = lista->next;
struct element * last = lista->next;
struct element * secondtolast = lista->next;
while(last->next != NULL)
{
secondtolast = last;
last = last->next;
}
lista->next= last;
last->next = first->next;
secondtolast->next = first;
first->next = NULL;
}

void wyswieltListeZGlowa(struct element * lista)
{
struct element * wsk=lista->next;

while(wsk!=NULL)
{
printf("%d\n", wsk->i);
wsk=wsk->next;
}
printf("---\n");
}

int main(void) {
// stwórz listę z głową o elementach 5,7,-4
struct element * lista = malloc(sizeof(struct element));
lista->next = malloc(sizeof(struct element));
lista->next->i = 1;
lista->next->next = malloc(sizeof(struct element));
lista->next->next->i = 2;
lista->next->next->next = malloc(sizeof(struct element));
lista->next->next->next->i = 3;
lista->next->next->next->next = malloc(sizeof(struct element));
lista->next->next->next->next->i = 4;
lista->next->next->next->next->next = malloc(sizeof(struct element));
lista->next->next->next->next->next->i = 5;
lista->next->next->next->next->next->next = NULL;

wyswieltListeZGlowa(lista);
foo(lista);
wyswieltListeZGlowa(lista);
return 0;
}
```

Переписывание одного значения в другое
(const int * ^2 -> int *)

```
void zad2(const int *a, int * b){
   *b = (*a)*(*a);
}

int main(int argc, const char * argv[]) {
   const int a = 2;
   int b = 0;
   printf("%d %d\n", a, b);
   zad2(&a, &b);
   printf("%d %d\n", a, b);
   return 0;
}
```

Нахождение среднего значения из структуры

```
   struct Temperature{
   char* city;
   float temperature;
};

float average_temperature(struct Temperature tab[], int n) {
   float wynik = 0;
   for (int i = 0; i < n; i++) {
      wynik += tab[i].temperature;
   }
   return wynik / n;
}

int main(int argc, const char * argv[]) {
   struct Temperature tab[] = {{"Kraków",9},{"Warszawa",10},{"Olsztyn",9}};
   int n = 3;
   printf("%f\n", average_temperature(tab, n));
   return 0;
}
```

Выводим в консоль адреса элементов которые больше среднего значения всех элементов списка без головы

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

void wyswieltListeBezGlowy(struct node * lista) {
   struct node *wsk = lista;
   int sum = 0;
   int count = 0;

   while (wsk != NULL) {
      sum += wsk->a;
      count++;
      wsk = wsk->next;
   }

   printf("Average: %d\n", sum/count);


   wsk = lista;
   while (wsk != NULL) {
      if (wsk->a > sum/count) {
         printf("Adres: %p Wartosc: %d\n", (void *)wsk, wsk->a);
      }
      wsk = wsk->next;
   }
}
```

```
int main() {
   struct node * lista = malloc(sizeof(struct node));
   lista->a = 3;
   lista->next = malloc(sizeof(struct node));
   lista->next->a = 4;
   lista->next->next = malloc(sizeof(struct node));
   lista->next->next->a = 5;
   lista->next->next->next = malloc(sizeof(struct node));
   lista->next->next->next->a = 6;
   lista->next->next->next->next = NULL;

   wyswieltListeBezGlowy(lista);
   return 0;
}
```

Функция возвращающая struct, находит самое большое значение

```
struct Kierowca{
   char imie[10];
   int liczba_przejechanych_kilometrow;
};

struct Kierowca  zad3(struct Kierowca tab[],int n){
   struct Kierowca tmp = tab[0];
   struct Kierowca chuj;
   int minn=tmp.liczba_przejechanych_kilometrow;
   for(int i = 0;i<n;i++){
      tmp=tab[i];
      if(minn<=tmp.liczba_przejechanych_kilometrow){
         minn=tmp.liczba_przejechanych_kilometrow;
         chuj=tmp;
      }
   }
   return chuj;
}

int main(){
struct Kierowca tab[] = {{"Andrii",5,6},{"KillReal",2,3},{"Denys",1,29}};
struct Kierowca wynik = zad3(tab, 3);
printf("%s, %d", wynik.imie, wynik.liczba_przejechanych_kilometrow);
return 0;
}
```

Удаление элементов делящихся на 3 из списка с головой

```
struct element {
   int x;
   struct element * next;
};
void remove_minus(struct element **lista) {
   struct element *current = *lista;
   struct element *previous = NULL;
   while (current != NULL) {
      if (current->x %3==0) {
         if (previous == NULL) {
            *lista = current->next;
            free(current);
            current = *lista;
         } else {
            previous->next = current->next;
            free(current);
            current = previous->next;
         }
      } else {
         previous = current;
         current = current->next;
      }
   }
}
void viewlist(struct element *lista){
   struct element * wsk = lista->next;
   while(wsk != NULL)
   {
      printf("%d\n", wsk->x);
      wsk = wsk->next;
   }
}

int main(){
struct element * lista = malloc(sizeof(struct element));
lista->next = malloc(sizeof(struct element));
lista->next->x = 5;
lista->next->next = malloc(sizeof(struct element));
lista->next->next->x = 9;
lista->next->next->next = malloc(sizeof(struct element));
lista->next->next->next->x = -4;
lista->next->next->next->next = NULL;

//wyswietl listę
viewlist(lista);

remove_minus(&lista);
viewlist(lista);
}
```

Поиск элемента в массиве tab размером n, используя заданную функцию сравнения isEqual для определения равенства элементов. Первый поиск использует стандартную функцию isEqual, а второй поиск - менее стандартную функцию isEqual2, которая проверяет, делится ли один элемент на другой без остатка. Код выводит индексы найденных элементов или -1, если элемент не найден.

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

int findElement(int *tab, int n, int val, int (*isEqual)(int, int)) {
   for (int i = 0; i < n; i++) {
      if (isEqual(tab[i], val)) {
         return i;
      }
   }
   return -1;
}

//klasyczna funkcja
int isEqual(int a, int b) {
   return a == b;
}

//mniej standardowa funkcja
int isEqual2(int a, int b) {
   return a % b == 0;
}

int main()
{
   int tab[] = {1, 2, -3, 41, 5, 6, 7, 8};
   int n = 8;
   int val = 4;
   int index = findElement(tab, n, val, isEqual);
   printf("Index: %d\n", index);
   index = findElement(tab, n, val, isEqual2);
   printf("Index: %d\n", index);
   return 0;
}
```

возвращает указатель на последний элемент списка с головой, который имеет положительное значение;

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

struct node * ostatnia_dodania(struct node * list) {
   struct node * last_positive = NULL;
   struct node * current = list->next;
   while (current != NULL) {
      if (current->value > 0) {
         last_positive = current;
      }
      current = current->next;
   }
   return last_positive;
}

void print_list(struct node * list) {
   struct node * current = list->next;
   while (current != NULL) {
      printf("%f %p\n", current->value, current);
      current = current->next;
   }
   printf("---\n");
}

int main()
{
   //przyklad pozytywny
   struct node * lista = malloc(sizeof(struct node));
   lista->next = malloc(sizeof(struct node));
   lista->next->value = 12;
   lista->next->next = malloc(sizeof(struct node));
   lista->next->next->value = 13;
   lista->next->next->next = malloc(sizeof(struct node));
   lista->next->next->next->value = -14;
   lista->next->next->next->next = NULL;
   print_list(lista);
   printf("%p\n", ostatnia_dodania(lista));
   //przyklad bez dodatnich liczb
   struct node * lista2 = malloc(sizeof(struct node));
   lista2->next = malloc(sizeof(struct node));
   lista2->next->value = -12;
   lista2->next->next = malloc(sizeof(struct node));
   lista2->next->next->value = -13;
   lista2->next->next->next = NULL;
   print_list(lista2);
   printf("%p\n", ostatnia_dodania(lista2));
}
```

- **создает двумерный динамический массив tab размером 4x3 и заполняет его значениями.**
- **Выводит содержимое массива tab на экран.**
- **Вызывает функцию swap_rows, которая меняет местами строки массива tab, но только если n (количество строк) больше или равно 4.**
- **Затем выводит измененное содержимое массива tab после вызова swap_rows.**

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

void swap_rows(int **tab, int n, int m) {
   if (n < 4) return;
   int *temp = tab[1];
   tab[1] = tab[n - 2];
   tab[n - 2] = temp;
}

void print_tab(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");
   }
}

int main()
{
   int n = 4;
   int m = 3;
   int **tab = malloc(n * sizeof(int *));
   for (int i = 0; i < n; ++i) {
      tab[i] = malloc(m * sizeof(int));
   }
   tab[0][0] = 2;
   tab[0][1] = 3;
   tab[0][2] = -3;
   tab[1][0] = 1;
   tab[1][1] = 4;
   tab[1][2] = 7;
   tab[2][0] = -3;
   tab[2][1] = -6;
   tab[2][2] = 11;
   tab[3][0] = -2;
   tab[3][1] = 8;
   tab[3][2] = 23;
   print_tab(tab, n, m);
   printf("\n");
   swap_rows(tab, n, m);
   print_tab(tab, n, m);
   return 0;
}
```

(Dwie listy z glową) zwraca sumę kwadratów elementów z obu list.

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

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

float sum(struct element *list1, struct element *list2) {
   float sum = 0;
   while (list1->next != NULL) {
      list1 = list1->next;
      sum += list1->x * list1->x;
   }
   while (list2->next != NULL) {
      list2 = list2->next;
      sum += list2->x * list2->x;
   }
   return sum;
}

int main()
{
   struct element *list1 = malloc(sizeof(struct element));
   list1->next = malloc(sizeof(struct element));
   list1->next->x = 1;
   list1->next->next = malloc(sizeof(struct element));
   list1->next->next->x = 2;
   list1->next->next->next = NULL;
   struct element *list2 = malloc(sizeof(struct element));
   list2->next = malloc(sizeof(struct element));
   list2->next->x = -3;
   list2->next->next = malloc(sizeof(struct element));
   list2->next->next->x = 4;
   list2->next->next->next = malloc(sizeof(struct element));
   list2->next->next->next->x = 0;
   list2->next->next->next->next = NULL;
   printf("%f", sum(list1, list2));
   return 0;
}
```

Zad.3. Napisz strukturę Samochod z polami marka (tablica znaków długości 50) 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 na nowo utworzoną strukturę, ustawiając składowe z przekazanych argumentów. Do- datkowo funkcja powinna sprawdzić, aby marka była napisem długości co najmniej 2 i przebieg był większy niż 0. W przypadku nie spełnienia jednego z warunków, funkcja powinna zwracać NULL.
b) zwiekszPrzebieg - funkcja, której argumentem jest wskaźnik do struktury typu Samochod. Funkcja ma dodać 1000 do przebiegu w przekazanym argumencie.

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

struct Samochod {
   char marka[50];
   int przebieg;
};

struct Samochod * initSamochod(char marka[], int przebieg) {
   if (przebieg > 0 && strlen(marka) >= 2) {
      struct Samochod *samochod = malloc(sizeof(struct Samochod));
      strcpy(samochod->marka, marka);
      samochod->przebieg = przebieg;
      return samochod;
   }
   return NULL;
}

void zwiekszPrzebieg(struct Samochod *samochod) {
   samochod->przebieg += 1000;
}

int main() {
   struct Samochod *samochod = initSamochod("Fiat", 1000);
   if (samochod != NULL) {
      zwiekszPrzebieg(samochod);
      printf("%d", samochod->przebieg);
   }
   return 0;
}
```

Zad.4. Napisz funkcję, która otrzymuje jako argument listę bez głowy o elementach typu:
```
struct node {
    int a;
    struct node * next;
};
```
Funkcja powinna wyzerować wartości elementów listy, które są podzielne przez 3. Stwórz przypadek testowy.

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

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

void printList(struct node * list) {
   struct node * tmp = list;
   while (tmp != NULL) {
      printf("%d ", tmp->a);
      tmp = tmp->next;
   }
   printf("\n");
}

void foo(struct node * list) {
   struct node * tmp = list;
   while (tmp != NULL) {
      if (tmp->a % 3 == 0) {
         tmp->a = 0;
      }
      tmp = tmp->next;
   }
}

int main() {
   struct node * list = malloc(sizeof(struct node));
   list->a = 1;
   list->next = malloc(sizeof(struct node));
   list->next->a = 2;
   list->next->next = malloc(sizeof(struct node));
   list->next->next->a = 3;
   list->next->next->next = malloc(sizeof(struct node));
   list->next->next->next->a = -6;
   list->next->next->next->next = malloc(sizeof(struct node));
   list->next->next->next->next->a = 5;
   list->next->next->next->next->next = NULL;
   printList(list);
   foo(list);
   printList(list);
   return 0;
}
```

Определяет функцию sum_of_min_indexes, которая принимает двумерный массив tab размером n x n и находит сумму индексов строк и столбцов элемента с минимальным значением в массиве. Если есть несколько элементов с одинаковым минимальным значением, то выбирается элемент с наименьшей суммой индексов строк и столбцов.

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

int sum_of_min_indexes(int **tab, int n) {
   int min = tab[0][0];
   int sum = 0;
   for (int i = 0; i < n; i++) {
      for (int j = 0; j<n; j++) {
         if (tab[i][j] < min) {
            min = tab[i][j];
            sum = i + j;
         }
      }
   }
   for (int i = 0; i < n; i++) {
      for (int j = 0; j<n; j++) {
         if (tab[i][j] == min && sum > i + j){
            sum = i + j;
         }
      }
   }
   return sum;
}

int main()
{
   int ** tab = malloc(3 * sizeof(int *));
   for (int i = 0; i < 3; i++) {
      tab[i] = malloc(3 * sizeof(int));
   }
   tab[0][0] = 1;
   tab[0][1] = -5;
   tab[0][2] = 3;
   tab[1][0] = 4;
   tab[1][1] = -5;
   tab[1][2] = 6;
   tab[2][0] = 7;
   tab[2][1] = 8;
   tab[2][2] = -5;
   printf("%d", sum_of_min_indexes(tab, 3));
}
```

lista bez glowy

- **Определяет функцию isSquare, которая проверяет, является ли целое число квадратом другого целого числа, путем итерации по числам и сравнения квадрата числа с данным значением.**
- **Определяет функцию printSquares, которая принимает связанный список list и выводит на экран все элементы списка, которые являются квадратами целых чисел, используя функцию isSquare.**

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

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

//funkcja pomocnicza sprawdzająca czy liczba całkowita moze byc kwadratem innej liczby
całkowitej
int isSquare(int n) {
   int i =0;
   while (i*i <= n) {
      if ((i*i) == n) {
         return 1;
      }
      i++;
   }
   return 0;
}

void printSquares(struct node * list) {
   struct node * tmp = list;
   while (tmp != NULL) {
      if (isSquare(tmp->i)) {
         printf("%d\n", tmp->i);
      }
      tmp = tmp->next;
   }
}

int main() {
   struct node *list = malloc(sizeof(struct node));
   list->i = 4;
   list->next = malloc(sizeof(struct node));
   list->next->i = -5;
   list->next->next = malloc(sizeof(struct node));
   list->next->next->i = 6;
   list->next->next->next = malloc(sizeof(struct node));
   list->next->next->next->i = 25;
   list->next->next->next->next = NULL;
   printSquares(list);
}
```