

18 вариант Хутиева Эрика М30-116Б

строки

```
#include <locale.h>
```

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

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

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

```
#define WORDS 30
```

```
#define LETTERS 10
```

```
bool checkSymbol(char c);
```

```
void wordProcessing(int* i, int x);
```

```
char string[1000]; //Буферный массив
```

```
char buffer[20];
```

```
int howWords = 0;
```

```
int i = -1;
```

```
int main()
```

```
{
```

```
    setlocale(LC_ALL, "Rus"); //Включаем русский язык
```

```
    printf("Напишите строку:\n");
```

```
    fgets(string, 1000, stdin); //Считываем из stdin (standart input - поток ввода) строку в массив string с ограничением в 600 символов
```

```
    for (int x = 0; x < 1000; ++x)
```

```
    {
```

```
        i++; //Передвигаем индекс в слове на новую букву
```

```
        if (i >= LETTERS) { //Слишком много букв в слове
```

```
            printf("Может быть только 10 букв в одном слове\n");
```

```
            exit(1);
```

```
        }
```

```
        if (string[x] == ' ' | string[x] == '.') { //Нашли пробел или точку
```

```
            wordProcessing(&i, x);
```

```

    }
    else {
        if (checkSymbol(string[x]) == true)
            buffer[i] = string[x];
    }
}

return 0;
}

```

```

bool checkSymbol(char c) {
    //Печатаем только цифры и латинские буквы
    if ((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z') || (c >= '0' && c <= '9'))
        return true;
    else {
        if (c == '\n')
            printf("Напишите точку\n");
        else
            printf("Можно использовать только цифры и латинские буквы\n");
        exit(1);
    }
    return false;
}

```

```

void wordProcessing(int *i, int x) {
    buffer[*i] = '\0'; //Символ конца слова
    if (*i != 0) { //Если в слове только один символ - всё слово пробел, и мы его пропускаем
        howWords++;
        if (howWords > WORDS) { //Слишком много слов
            printf("Может быть только 30 слов\n");
            exit(1);
        }
        if ((*i) % 2 == 0) //Чётное количество букв

```

```
    {  
        printf("%d", *i);  
        printf("%s", buffer);  
    }  
  
    printf(" ");  
  
}  
if (string[x] == '.')//Конец программы  
    exit(0);  
*i = -1;//Начинаем запись слова с начала  
}
```

Динамический

```
#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

void input(int* dynamic_array, int n);
void output(int* dynamic_array, int n);
void evensDelete(int** dynamic_array, int* n);
float findMedian(int* dynamic_array, int n);

int main()
{
    //Включаем поддержку русского языка в консоли
    setlocale(LC_ALL, "Rus");

    int* dynamic_array;

    //Длина динамического массива
    int n = 0;

    printf("Введите n: ");
    scanf("%i",&n);
    if (n <= 0) {
        printf("Введите натуральную длину строки\n");
        exit(1);
    }

    //Выделяем память для массива с заданной длиной
    dynamic_array = malloc(n * sizeof(float));

    input(dynamic_array, n);
    output(dynamic_array, n);
    evensDelete(&dynamic_array, &n);
    output(dynamic_array, n);
}
```

```

        free(dynamic_array);

        return 0;
    }

float findMedian(int* dynamic_array, int n) {
    int sum = 0;

    //Сумма всех элементов
    for (int i = 0; i < n; i++)
        sum += dynamic_array[i];

    float f;

    //Медиана
    f = (float)sum / n;

    return f;
}

void evensDelete(int** dynamic_array, int* n) {
    //Находим медиану
    float median = findMedian(*dynamic_array, *n);

    printf("Среднее значение: %f\n", median);

    for (int x = 0; x < *n; x++)
        //Нашли элемент для удаления
        if ((*dynamic_array)[x] % 2 == 0 && (*dynamic_array)[x] > median) {
            //Перемещаем его в конец
            for (int y = x + 1; y < *n; y++) {
                float f = (*dynamic_array)[y];
                (*dynamic_array)[y] = (*dynamic_array)[y - 1];
                (*dynamic_array)[y - 1] = f;
            }
            (*n)--;
        }

    //Перевыделяем память под новое количество элементов, не трогая
    (*dynamic_array) = realloc((*dynamic_array), (*n * sizeof(float)));
}

```

```
void input(int* dynamic_array, int n) {  
    //Заполням массив из консоли  
    for (int i = 0; i < n; i++) {  
        printf("Напишите элемент массива: ");  
        scanf("%d", &dynamic_array[i]);  
    }  
}  
  
void output(int* dynamic_array, int n) {  
    //Выводим массив в консоли  
    for (int i = 0; i < n; i++)  
        printf("%d ", dynamic_array[i]);  
    printf("\n");  
}
```

Двумерный

```
#include <locale.h>
```

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

```
#define HEIGHT 3
```

```
#define WIDHT 4
```

```
void input(int matrix[HEIGHT][WIDHT]);
```

```
void output(int matrix[HEIGHT][WIDHT]);
```

```
void columnMedian(int matrix[HEIGHT][WIDHT]);
```

```
int maxNegativeElements(int matrix[HEIGHT][WIDHT]);
```

```
int main()
```

```
{
```

```
    //Включаем русский язык
```

```
    setlocale(LC_ALL, "Rus");
```

```
    int matrix[HEIGHT][WIDHT];
```

```
    input(matrix);
```

```
    output(matrix);
```

```
    columnMedian(matrix);
```

```
    return 0;
```

```
}
```

```
int maxNegativeElements(int matrix[HEIGHT][WIDHT]) {
```

```
    int maxNegative = 0;
```

```
    //Считаем количества отрицательных чисел идя по каждому столбцу сверху вниз
```

```
    for (int x = 0; x < WIDHT; ++x) {
```

```
        int howNegative = 0;
```

```
        for (int y = 0; y < HEIGHT; ++y)
```

```
            if (matrix[y][x] < 0)
```

```
                howNegative++;
```

```

        //Находим максимальное их количества
        if (howNegative > maxNegative)
            maxNegative = howNegative;
    }
    return maxNegative;
}

void columnMedian(int matrix[HEIGHT][WIDHT]) {
    //Находим максимальное количество отрицательных чисел
    int maxNegative = maxNegativeElements( matrix);
    for (int x = 0; x < WIDHT; ++x) {
        int howNegative = 0;
        int sum = 0;
        //Считаем сумму элементов столбца и количества в нём отрицательных чисел
        for (int y=0; y < HEIGHT; ++y) {
            sum += matrix[y][x];
            if (matrix[y][x] < 0)
                howNegative++;
        }
        //Если количества отрицательных чисел столбца совпадает с максимальным
        if (howNegative == maxNegative)
            printf("Среднее арифметическое элементов %d столбца = %f\n", x+1,
(float)sum / HEIGHT);
    }
}

```

```

void input(int matrix[HEIGHT][WIDHT]) {
    printf("Напишите %d элементов:\n", HEIGHT*WIDHT);
    //Записываем по элементно их консоли в матрицу
    for (int y = 0; y < HEIGHT; ++y)
        for (int x = 0; x < WIDHT; ++x)
            scanf("%d", &matrix[y][x]);
    printf("\n");
}

```



```
}  
  
void output(int matrix[HEIGHT][WIDHT]) {  
    //Выводим в консоль  
    for (int y = 0; y < HEIGHT; ++y) {  
        for (int x = 0; x < WIDHT; ++x)  
            printf("%d ", matrix[y][x]);  
        printf("\n");  
    }  
    printf("\n");  
}
```

Списки 1

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

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

```
#include <locale.h>
```

```
struct List {
```

```
    int c;
```

```
    int power;
```

```
    struct List* next;
```

```
};
```

```
struct List* find(struct List* l, struct List* end, int power);
```

```
void subtraction(struct List* l1, struct List* l2);
```

```
void input(struct List** l);
```

```
void output(struct List* l);
```

```
void freeMemory(struct List*);
```

```
void initSingleNode(struct List* singleNode);
```

```
int main()
```

```
{
```

```
    setlocale(LC_ALL, "Russian");
```

```
    struct List* l1 = NULL;
```

```
    struct List singleNode;
```

```
    input(&l1);
```

```
    initSingleNode(&singleNode);
```

```
    output(l1);
```

```
    output(&singleNode);
```

```
    subtraction(l1, &singleNode);
```

```
    output(l1);
```

```

    freeMemory(l1);

    return 0;
}

void initSingleNode(struct List* singleNode) {
    printf("Введите A и n элемента A*x^n\n");
    scanf("%d %d", &singleNode->c, &singleNode->power);
    singleNode->next = NULL;
}

void input(struct List** l) {
    int n;
    printf("Напишите n: ");
    scanf("%i", &n);
    if (n <= 0) {
        printf("Введите число большее 0 \n");
        exit(1);
    }

    //Выделяем память для первого элемента
    *l = malloc(sizeof(struct List));
    struct List* List = *l;
    printf("Напишите коэффициент и степень\n");
    //Вводим степень и коэффициент
    scanf("%i", &(List->c));
    scanf("%i", &(List->power));
    if (List->c == 0)
    {
        printf("Введите коэффициент отличный от 0\n");
        exit(1);
    }

    for (int i = 0; i < n - 1; ++i) {
        //Выделяем память для следующего элемента

```

```

List->next = malloc(sizeof(struct List));

List = List->next;

int tempPower = 0;

printf("Напишите коэффициент и степень\n");

//Вводим степень и коэффициент
scanf("%d", &(List->c));

scanf("%d", &(tempPower));

//Выводим ошибку если уже есть такая степень
if (find(*l, List, tempPower) != NULL) {

    printf("Список не должен содержать 2 элементов с одной степенью\n");

    exit(1);

}

List->power = tempPower;

//Выводим ошибку если коэффициент = 0
if (List->c == 0)

{

    printf("Ошибка: введите коэффициент отличный от 0\n");

    exit(1);

}

}

List->next = NULL;

}

struct List* find(struct List* l, struct List* end, int power)

{

    struct List* List = l;

    //Проверяем пока не дошли до заданного конца
    while (List != end) {

        if (List->power == power) //Если нашли

            return List;

        List = List->next;

    }

}

```

```

        return NULL;
    }

void subtraction(struct List* l1, struct List* l2) {
    //Пытаемся найти элемент такой же степени, как единичный
    struct List* samePower = find(l1, NULL, l2->power);

    //Не нашли
    if (samePower == NULL) {
        //Идём в конец списка
        while (l1->next != NULL)
        {
            l1 = l1->next;
        }

        //Выделяем память для нового элемента
        l1->next = malloc(sizeof(struct List));

        //Копируем данные из l2
        l1->next->c = l2->c;

        l1->next->power = l2->power;

        l1 = l1->next;

        l1->next = NULL;
    }

    //Нашли
    else
    {
        //Вычитаем
        samePower->c -= l2->c;

        if (samePower->c == 0) {
            printf("Разность коэффициентов не должна равняться 0\n");
            exit(1);
        }
    }
}

```

```

void output(struct List* l) {
    struct List* List = l;
    while (List != NULL) { //Пишем пока не найдём последний элемент (он указывает на NULL)
        if (List->c > 0) //Пишем плюс перед числом, если оно положительное
            printf("%c", '+');
        printf("%ix^%i", List->c, List->power);
        //Переходим к следующему элементу
        List = List->next;
    }
    printf("%c", '\n');
}

```

```

void freeMemory(struct List* l) {
    struct List* p = l;
    //Переходим к новому элементу и освобождаем память для предыдущего
    while (l != NULL) {
        p = l;
        l = l->next;
        free(p);
    }
}

```

Списки 2

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

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

```
#include <locale.h>
```

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

//Двусвязный список. Каждый элемент имеет длину слова, само слово и указатели на следующий и предыдущий элементы

```

struct Doubly_linked_list {
    struct Doubly_linked_list* next;
    struct Doubly_linked_list* previous;
}

```

```

        char* word;

        unsigned int length;

};

void output(struct Doubly_linked_list* l);
void input(struct Doubly_linked_list* l);

void divide(struct Doubly_linked_list** l, struct Doubly_linked_list** secondRoot);
bool checkSymbol(char c);
bool wordProcessing(int* i, int x, struct Doubly_linked_list** d_l_list);

//Освобождаем память
void clear(struct Doubly_linked_list* l);
int main()
{
    setlocale(LC_ALL, "Rus");

    struct Doubly_linked_list* d_l_list;

    d_l_list = malloc(sizeof(struct Doubly_linked_list));
    input(d_l_list);
    output(d_l_list);
    struct Doubly_linked_list* second_list;

    divide(&d_l_list,&second_list);
    output(d_l_list);
    output(second_list);
    return 0;
}

void output(struct Doubly_linked_list* l) {

    struct Doubly_linked_list* Doubly_linked_list = l;

```

```

while (Doubly_linked_list != NULL) {
    printf("%s ", Doubly_linked_list->word);
    Doubly_linked_list = Doubly_linked_list->next;
}
printf("\n");
}

```

```

char str[700];

```

```

char w[50];

```

```

void input(struct Doubly_linked_list* l) {
    printf("Напишите строку\n");
    fgets(str, 700, stdin); //Считываем 700 символов из stdin в str
    struct Doubly_linked_list* d_l_list = l;
    l->previous = NULL;

```

```

    int i = -1;
    for (int x = 0; x < 700; ++x){
        i++;
        //Нашли новое слово
        if (str[x] == ' ' || str[x] == '.') {
            if (wordProcessing(&i, x, &d_l_list) == false)
                break;
        }
        else {
            //Записываем, если это буква
            if(checkSymbol(str[x]) == true)
                w[i] = str[x];
        }
    }
}

```

```

}

bool wordProcessing(int* i, int x, struct Doubly_linked_list** d_l_list) {

```



```

//Если i=0, то всё слово это пробел
if (*i != 0) {
    w[*i] = '\0';

    //Выделяем память для слово
    (*d_l_list)->word = malloc(sizeof(char) * (*i + 1));

    //Переносим слово из буфера в word
    for (int y = 0; y <= *i; ++y)
        (*d_l_list)->word[y] = w[y];

    (*d_l_list)->length = *i;

    //Выделяем память для следующего узла
    (*d_l_list)->next = malloc(sizeof(struct Doubly_linked_list));

    if (str[x] == '.') {
        (*d_l_list)->next = NULL;

        return false;
    }

    //Ставим, чтобы указатель "назад" следующего указывал на текущий
    (*d_l_list)->next->previous = (*d_l_list);

    //Переходим к следующему
    (*d_l_list) = (*d_l_list)->next;
}

if (str[x] == '.') {
    (*d_l_list)->previous->next = NULL;

    return false;
}

*i = -1;

return true;
}

bool checkSymbol(char c) {
    //Печатаем только латинские буквы
    if (c >= 'a' && c <= 'z')
        return true;

    else {

```

```

        if (c == '\n')
            printf("Напишите точку\n");
        else
            printf("Можно использовать латинские буквы\n");
        exit(1);
    }
    return false;
}

void clear(struct Doubly_linked_list* l) {
    struct Doubly_linked_list* p = l;
    //Переходим к новому элементу и освобождаем память предыдущего
    while (l != NULL) {
        p = l;
        l = l->next;
        free(p->word);
        free(p);
    }
}

void divide(struct Doubly_linked_list** l, struct Doubly_linked_list** secondRoot) {
    int N;
    printf("Введите N\n");
    scanf("%d", &N);
    if (N <= 0) {
        printf("Введите натуральное число\n");
        exit(1);
    }

    int i = 1;
    struct Doubly_linked_list *temp_Doubly_linked_list = *l;
    //Находим разделяющий узел

```

```

while ( i!=N&& temp_Doubly_linked_list != NULL) {
    temp_Doubly_linked_list = temp_Doubly_linked_list->next;
    i++;
}
//Если N больше количества узлов
if (temp_Doubly_linked_list == NULL) {
    printf("Сликом большое N\n");
    exit(1);
}
//Нет указателя на предыдущий элемент только у первого
//Делаем первый список пустым
if (temp_Doubly_linked_list->previous != NULL)
    //Обозначаем, что элемент перед разделяющим последний
    temp_Doubly_linked_list->previous->next = NULL;
else (*l) = NULL;

//Нет указателя на следующий элемент только у последнего
//Делаем второй список пустым
if (temp_Doubly_linked_list->next != NULL) {
    //Обозначаем, что элемент после разделяющего первый
    *secondRoot = temp_Doubly_linked_list->next;
    (*secondRoot)->previous = NULL;
}
else *secondRoot = NULL;

//Удаляем разделяющий узел
free(temp_Doubly_linked_list->word);
free(temp_Doubly_linked_list);
}

```