

Раздел «Язык Си» . CoffeArPointFunc :

- **Массивы**
 - Как зависит указатель от типа.
 - Объявление массива. Элементы массива.
 - Некоторые примеры вычисления номеров элементов.
 - Копирование массивов.
 - Массивы как аргументы функции
 - Указатели в помощь
 - Массивы структур.
 - Массивы в структурах.
 - Задачи на одномерные массивы
 - Простые задачи
 - 🍌 Точки на прямой
 - 🍌 Шары
 - 🍌 Бегущая строка
 - 🍌 Ближайший город
 - 🍌 Навести порядок
 - Задачи «на подумать»
 - 🍌 В десятичную систему
 - 🍌 Сравнить два числа
 - 🍌 Десятичное представление
 - 🍌 Затруднительное положение
 - Многомерные массивы (на примере двумерных)
 - Задачи на двумерные массивы
 - Простые задачи
 - 🍌 Крест
 - 🍌 Косой крест
 - 🍌 Прямоугольники
 - Задача "на подумать" (Узор)

Поиск

Поиск

Раздел «Язык Си»

Главная
Зачем учить C?
Определения

Инструменты:

Поиск
Изменения
Index
Статистика

Разделы

Информация
Алгоритмы
Язык Си
Язык Ruby
Язык
Ассемблера
E! Judge
Парадигмы
Образование
Сети
Objective C

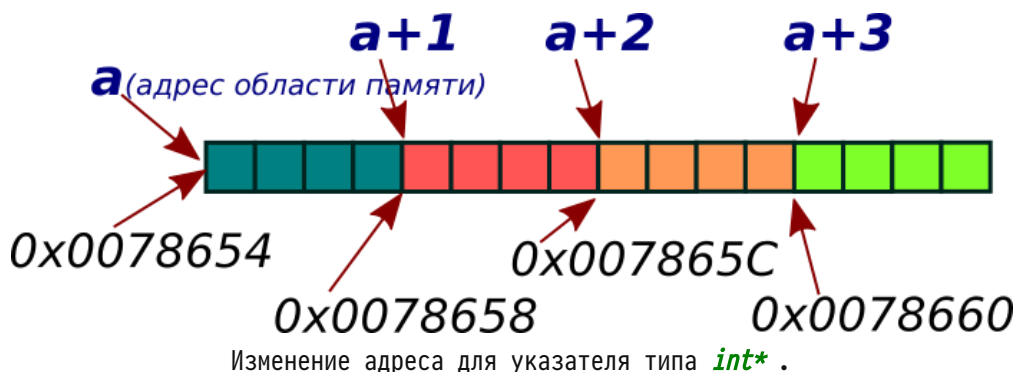
Login>>

Массивы

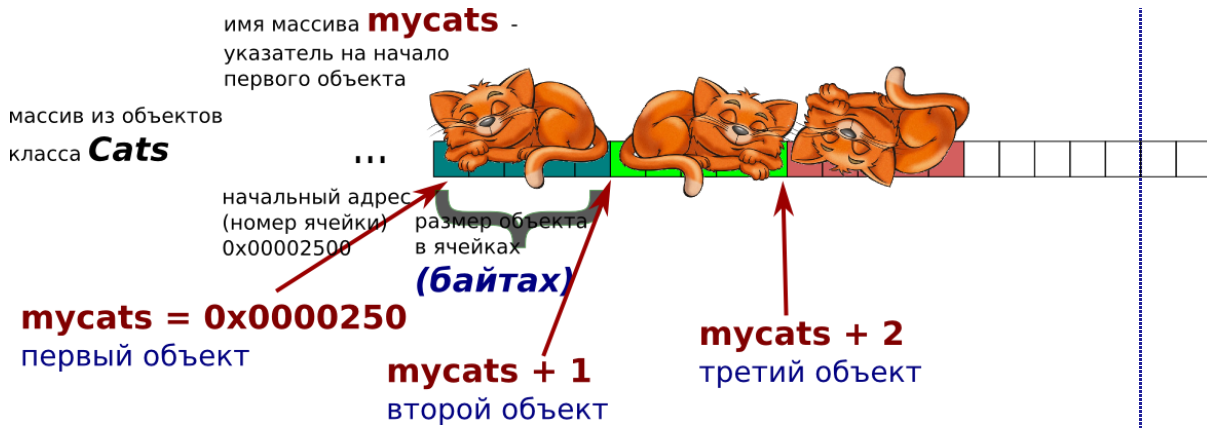
Как зависит указатель от типа.

Пусть в некоторой области памяти компьютера последовательно расположены несколько переменных одного типа, например, *int* или переменные более сложные, например, структура типа *Cat* .

Мы можем передать адрес этой области указателю соответствующего типа. При этом операции вычисления адреса, например, *p++* или *p--* будут зависеть от размеров переменных данного типа.



Для указателя типа *Cat** размер, конечно, будет другой.



Изменение адреса для указателя типа *int** .

Объявление массива. Элементы массива.

Непрерывная область памяти с однородными элементами – это **массив**.

Объявить его в языке C можно указав из каких элементов будет состоять массив.

```
// массив из 5 целых чисел.
// а - имя массива и указатель на эту область памяти
int a[5];

// массив из 10 дробных чисел.
// real - имя массива и указатель на эту область памяти
float real[10];

// массив из 100 символов.
// str - имя массива и указатель на эту область памяти
char str[10];
```

Мы можем описать некоторую структуру и объявить массив таких структур.

```
typedef struct A{
    int x,y;
    char name;
}Star;

//.....
int main(){
    // массив из 20 элементов типа Star
    // z - имя массива и указатель на него.
    Star z[20];
}
```

Для обращения к значению каждого элемента массива служит оператор **[номер]**.

Номер элемента – это смещение относительно начала массива. Значит первый элемент будет иметь номер **0**. А последний, если всего **n** элементов, **n - 1**.

Адрес каждого элемента получается через указатель на этот элемент.

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

int main(){
    int i;
    //массив из 5 целых чисел
    // а - имя массива и указатель
    // на эту область памяти
    int a[5];
    // после выделения памяти для 5 элементов,
    // в каждом из них находится "мусор"
    // Инициализация элементов массива
    a[0] = 0; // в первый элемент - 0
    a[4] = 2; // в последний (5-1=4) - 2
```

```
// Печать элементов массива (первый и последний)
printf("%d %d\n",a[0], a[4]);

// Получить ВСЕ элементы массива можно циклом:
for(i = 0; i < 5; i++){
// scanf требует указатель на элемент
// указатель на первый элемент - a
// указатель на следующий: a + 1,
// на второй: a + 2 и т.д
// Для вычисления смещения используем переменную i
scanf("%d",a + i);
}

// Напечатаем все полученные элементы
// номер для оператора [] тоже можно вычислять
for(i = 0; i < 5; i++){
printf("%d ",a[i]);
printf("\n");
}

return 0;
}
```

Некоторые примеры вычисления номеров элементов.

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

int main(){
int i;
//массив из 7 дробных чисел
// line - имя массива и указатель
// на эту область памяти
// Можно инициализировать сразу все элементы массива
float line[7] = {1.2, -4.89, 7.0, 0.0, 3.5, 9.12, -0.5};

// Напечатаем их в обратном порядке:
for( i = 7 - 1; i > -1; i--){
printf("%.2f ",line[i]);
}
printf("\n");
// Напечатаем их через один
for( i = 0; i < 7 -1; i += 2){
printf("%.2f ",line[i]);
}
printf("\n");
return 0;
}
```

Копирование массивов.

Иногда необходимо скопировать элементы одного массива в другой. Казалось бы можно просто присвоить один массив другому.

НО, имя массива – это указатель, и простое присваивание приведет к тому, что два указателя будут «смотреть» на один и тот же массив. А доступ ко второму массиву будет утерян. К тому же современные компиляторы не позволяют записать такую операцию.

Поэтому копирование двух массивов возможно только копированием всех каждого одного массива в соответствующий элемент второго. Можно пользоваться специальными функциями копирования, которые мы рассмотрим позже.

Массивы как аргументы функции

Действия с массивами всегда очень громоздки в написании. поэтому удобно, по-возможности, оформлять их в виде функций. Массивы можно передавать в функции как аргументы.

Напишем две функции: получение элементов массива с консоли и печать элементов массива.

```
#include <stdio.h>
#include <stdlib.h>
// Первым аргументом getArr получает
// указатель на массив целых чисел.
// При вызове функции в него будет скопирован
// адрес начала массива целых чисел
// Второй аргумент - количество элементов
// С ничего "не знает" о размере массива
void getArr(int *m, int n){
    int i;
    for(i = 0; i < n; i++){
        scanf("%d", m + i);
    }
};

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

int main(){
    int i;
    int a[5];
    // Вызов функции getArr
    // Первый аргумент - имя массива a
    // второй - количество элементов
    getArr(a,5);
    printArr(a,5);
    // аналогично
    return 0;
}
```

Указатели в помощь

Иногда нужно совершить действия только с частью массива. Для этого в функцию можно было бы передать: имя массива, номер начального элемента, номер конечного элемента. То есть для того, чтобы напечатать элементы с четвертого по десятый нужно писать другую функцию печати (с тремя аргументами).

Указатели в C позволяют избежать этого.

Пример печать части массива с помощью указателей.

```
#include <stdio.h>
#include <stdlib.h>
// Первым аргументом getArr получает
// указатель на массив целых чисел.
// При вызове функции в него будет скопирован
// адрес начала массива целых чисел
// Второй аргумент - количество элементов
// С ничего "не знает" о размере массива
void getArr(int *m, int n){
    int i;
    for(i = 0; i < n; i++){
        scanf("%d", m + i);
    }
};

void printArr(int* m, int n){
    int i;
    for(i = 0; i < n; i++){
        printf("%d ", m[i]);
    }
}
```

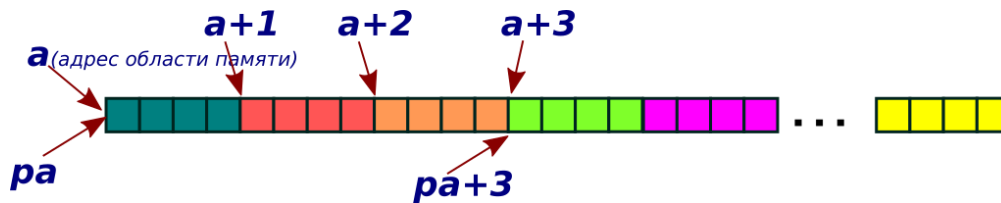
```

    printf("\n");
};

int main(){
    int i;
    int a[10];
    int *pa; // указатель на целое число
    // Вызов функции getArr
    // Первый аргумент - имя массива a
    // второй - количество элементов
    getArr(a,10);

    pa = a; // передача адреса массива a указателю pa
    pa += 3; // получили адрес четвертого элемента

```



```

// передача адреса третьего элемента
// функция будет "думать", что он первый
// всего нужно напечатать 10 - 3 = 7 элементов
printArr(pa,10-3);
// Или можно так вызвать функцию. Тогда мы напечатаем 5
// последних элементов.
printArr(a + 5,10-5);
return 0;
}

```

Массивы структур.

Структуры тоже могут образовывать массивы. И эти массивы ведут себя так же как и с «обычными» переменными.

Рассмотрим массив точек на плоскости. Каждая точка задана двумя координатами **x** и **y**.

```

#include <stdio.h>
#include <stdlib.h>
// описание структуры Point
typedef struct P{
    int x,y;
}Point;

int main(){
    //описание массива из переменных тип Point
    // и инициализация элементов:
    // каждому элементу по два значения: x и y
    Point figure[5] = {1,1, 2,2, 3,4, 10,0, 5,5};
    int i;
    // печать всех элементов.
    // сначала обращаемся к номеру элемента в массиве figure,
    // а затем к полям этого элемента
    for(i = 0; i < 5; i++){
        printf("(%d,%d) ",figure[i].x, figure[i].y);
    }

    return 0;
}

```

Пример функций для работы с массивами структур.

```

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

```

```
// описание структуры Point
typedef struct P{
    int x,y;
}Point;

void getPoints(Point* a, int n){
    int i;
    for(i = 0; i < n; i++){
        // Здесь нужен адрес не элемента структуры,
        // а поля в структуре
        scanf("%d%d", &(a[i].x), &(a[i].y));
    }
};

void printPoint(Point a){
    printf("(%d,%d) ",a.x, a.y);
};

void listPoints(Point* a, int n ){
    int i;
    for(i = 0; i < n; i++){
        // Воспользуемся уже готовой функции печати
        // одной точки
        printPoint(a[i]);
    }
};

int main(){
    //описание массива из переменных тип Point

    Point figure[5];
    int i;
    // вызов функции заполнения массива figure
    getPoints(figure,5);
    // вызов функции печати массива figure
    listPoints(figure,5);

    return 0;
}
```

Массивы в структурах.

Можно организовать такие структуры данных, что исчезнет необходимость предавать в функцию количество элементов массива. Это возможно при описании структуры, которая содержит массив и поле для количества элементов.

```
#include <stdio.h>
#include <stdlib.h>
// описание структуры Point
typedef struct P{
    int x,y;
}Point;
//структура для массива точек.
// При решении задач не всегда обязательно
// использовать все элементы массива
// Иногда память выделяется " с запасом"
// n - указывает сколько элементов используется
typedef struct MP{
    Point mp[5]; // массив точек
    int n;       // размер используемого массива
}MassPoints;

// функция получения элементов массива
// Так как мы будем изменять элементы
// нашей структуры, нужно передать указатель.
// Если передавать саму структуру, то передается
// копия структуры (и копия массива в структуре)
void getPoints(MassPoints * a){
```

```
    int i;
    // номер можно получить и другим способом.
    a->n = 5;
    for(i = 0; i < a->n; i++){
    // По указателю обращаемся к полю mp (это массив),
    // далее обращаемся к элементам этого массива и
    // к полям каждого элемента
        scanf("%d%d", &(a->mp[i].x), &(a->mp[i].y));
    }
};

void printPoint(Point a){
    printf("(%d,%d) ",a.x, a.y);
};
// Функция печати точек.
// Передаем указатель, но чтобы не изменить элементы
// массива ставим const.
// При попытке изменить значения MassPoints
// сообщение выдаст компилятор
void listPoints(const MassPoints* a){
    int i;
    for(i = 0; i < a->n; i++){
    // Обращаемся к i-тому элементу поля mp
        printPoint(a->mp[i]);
    }
};

int main(){
    //описание массива из переменных тип Point

    MassPoints figure;
    int i;
    // вызов функции заполнения массива figure
    getPoints(&figure);
    // вызов функции печати массива figure
    listPoints(&figure);

    return 0;
}
```

Задачи на одномерные массивы

Простые задачи

Точки на прямой

Точки на прямой заданы целыми числами. Точки записаны случайным образом.

Написать программу, которая печатает только те, точки, которые лежат слева от 0.

Шары

10 ячеек заполнены красными и зелеными шарами. Вес красных шаров обозначается положительными целым числом, вес зеленых отрицательным.

Написать программу, которая определяет какие шары тяжелее: все красные или все зеленые.

В ответе писать: RED, GREEN или ==

Бегущая строка

Дан массив из 10 символов.

Написать функцию **void toRight(char *a, int n)** , которая перемещает символы налево по принципу «бегущей строки»: первый символ становится вторым, второй – третьим, а последний – первым. Первый аргумент функции – имя массива символов, второй – количество прочитанных символов

Ближайший город

Города на карте представлены именем (один символ) и двумя целыми координатами. Имена всех городов уникальны.

Описать структуру для представления каждого города.

```
typedef struct S{
    char name;
    int x,y;
}City;
```

Описать структуру для массива из 10 городов.

```
typedef struct CN{
    City goroda[10];
    int n;
}Country;
```

Написать функцию **char getNearest(const Country map, City a)** , которая возвращает имя города, ближайшего к городу **a** на карте.

Навести порядок

10 ячеек заполнены красными и зелеными шарами. Вес красных шаров обозначается положительными целым числом, вес зеленых отрицательным.

Написать функцию **void regularize(int* balls, int n)** , которая перемещает все зеленые шары налево, а красные направо..

Задачи «на подумать»

В десятичную систему

Двоичное число записано в строку в виде символов 0 и 1. Строка не более 100 символов. При вводе число заканчивается либо концом строки, либо символом *.

Написать программу, которая печатает это число в десятичном виде, умноженное на 2.

Сравнить два числа

Две строки: число в восьмеричной системе счисления $0 \leq k \leq 10^{100}$ и число в шестнадцатеричной системе счисления $0 \leq n \leq 10^{100}$.

Написать программу, которая печатает большее из этих чисел.

Десятичное представление

Любое рациональное число представляется обыкновенной дробью. Числитель и знаменатель дроби – целые числа. Рациональное число также может быть представлено **периодической** десятичной дробью.

Дано рациональное число n/k , где $1 \leq k \leq 1000$ и $0 \leq n \leq 1000$.

Написать программу, которая представляет это число в виде периодической десятичной дроби:

<целая часть>.<непериодическая часть>(<период>).

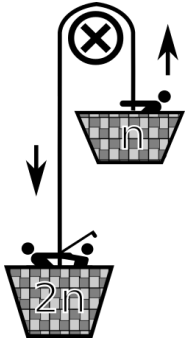


Затруднительное положение

Однажды Л.Ю. Бознательный был на экскурсии в прекрасном замке в горах. В этот момент случился обвал, и все выходы из замка оказались завалены. Однако Л.Ю. Бознательный обнаружил строительный блок, через который была перекинута веревка с двумя корзинами. Более того, в одной корзине оказался мешок с песком, весом 10 кг.

Блок устроен так, что корзины будут плавно подниматься и опускаться если вес одной корзины будет больше другой, но не более чем в **два** раза. Если вес одной корзины равен весу другой корзины, они окажутся на одной высоте (половина расстояния над пропастью), и никто уже не сможет спуститься или подняться. Если вес одной корзины превышает удвоенный вес второй, первая корзина разбивается.

Распросив всех людей о том какой у них вес, Л.Ю. Бознательный установил, что самый легкий человек обладает весом 20, а самый тяжелый – весом $20 \leq w_2 \leq 10000$. При на экскурсии оказались люди со всеми весами от 20 до максимального с шагом 10: (например: 20, 30, 40...).



Корзины были такие прочные, что могли выдержать всех людей одновременно.

Вначале пустая корзина поднята вверх, (к месту где оказались все экскурсанты), а корзина с мешком внизу.

Необходимо спустить всех вниз за минимальное количество спусков корзин.

Требуется написать программу, которая определяет минимальное число спусков, необходимое для безопасного спуска всех экскурсантов (последнее падение мешка безопасно и не учитывается в подсчете).

Пример правильного решения. Для максимального веса – 60 кг. минимальное количество спусков – 7.

Многомерные массивы (на примере двумерных)

Для работы с плоскими таблицами, изображениями и т.д. можно использовать двумерные массивы.

Например, в таблицу 2x3 (2 строки по 3 колонки) нужно записать все числа от 1 до 6.

Рассмотрим решение проблемы.

	0	1	2
0	1	2	3
1	4	5	6

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

int main(){
    // описание двумерного массива
    // целых чисел
    // 2 строки по 3 колонки
    int a[2][3];
    // номера строк и столбцов начинаются с 0
    // Первую колонку заполним числами с 1 до 3,
    // а вторую с 4 до 6
```

```
// номера колонок и строк
int x; // номер колонки
int y; // номер строки
int i = 1;
// сначала выбираем номер строки
for( y = 0; y < 2; y++){
// выбираем в этой строке номер колонки
    for( x = 0; x < 3; x++){
// i увеличится СРАЗУ после выполнения операции
// присваивания
        a[y][x] = i++;
    }
}

// Печать двумерного массива:

// сначала выбираем номер строки
for( y = 0; y < 2; y++){
// выбираем в этой строке номер колонки
    for( x = 0; x < 3; x++){
        printf("%d ",a[y][x]);
    }
// новая строка с "новой строки"
    printf("\n");
}
return 0;
}
```

Попробуем написать функции для чтения и печати двумерного массива.

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

void get2Arr(int n, int m, int a[n][m]){
    int x,y;
    // сначала выбираем номер строки
    for( y = 0; y < n; y++){
// выбираем в этой строке номер колонки
        for( x = 0; x < m; x++){
            scanf("%d",&(a[y][x]));
        }
    }
};

void print2Arr(int n, int m, int a[n][m]){
    int x,y;
    // сначала выбираем номер строки
    for( y = 0; y < n; y++){
// выбираем в этой строке номер колонки
        for( x = 0; x < m; x++){
// "Красивый" вывод:
// %2.0d : 2 - ширина поля под все число,
// если цифр меньше, заполняется пробелами
            printf("%2.d ",a[y][x]);
        }
// новая строка с "новой строки"
        printf("\n");
    }
};

int main(){
// описание двумерного массива
// целых чисел
// 2 строки по 3 колонки
    int a[5][4];
// номера строк и столбцов начинаются с 0
// Первую колонку заполним числами с 1 до 3,
// а вторую с 4 до 6
}
```

```
// номера колонок и строк
int x; // номер колонки
int y; // номер строки
int i = 1;

// заполнить двумерный массив числами
get2Arr(5,4,a);

// Печать двумерного массива:
print2Arr(5,4,a);
return 0;
}
```

Вывод:

```
> ./m2df<a2.dat
1  2  3  4
5  6  7  8
9 10 11 12
13 14 15 16
17 18 19 20
>
```

Задачи на двумерные массивы

Простые задачи

Крест

Двумерный массив 5x5 заполнен нулями. Написать программу, которая заполняет центральную колонку всеми единицами (1), а центральную строку – двойками (2). На пересечении строки и колонки стоит число 3.

		1		
		1		
2	2	3	2	2
		1		
		1		

Косой крест

Двумерный массив 5x5 заполнен нулями. Написать программу, которая заполняет левую диагональ всеми единицами (1), а правую – двойками (2). На пересечении диагоналей стоит число 3.

1				2
	1		2	
		3		
	2		2	
2				2

Прямоугольники

Двумерный массив символов 10x10 точками (".") и звездочками ("*"). Точка – белая клетка, "*" – черная. Черным нарисованы прямоугольники, которые НЕ ПЕРЕСЕКАЮТСЯ и НЕ СОПРИКАСАЮТСЯ. Одна черная клетка – тоже прямоугольник.

Написать программу, которая заполняет вычисляет количество прямоугольников.

--	--	--	--	--	--	--	--	--	--

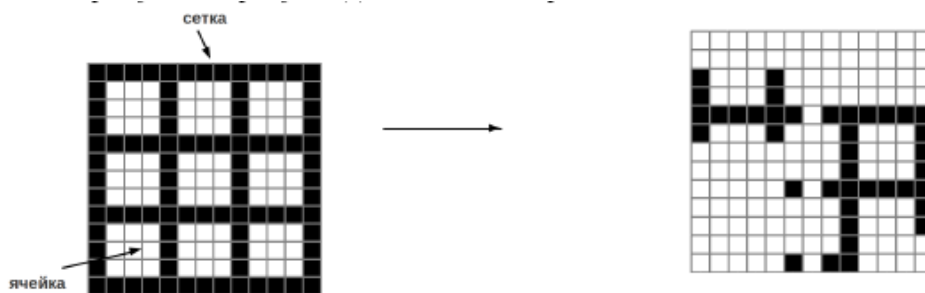
.	.	.	.	*	*	*	*	.	.
.	*	.	.	*	*	*	*	.	.
.
.
.	.	.	*
.	.	.	*
.	.	.	*
.	*	*	.	.
.	*	*
.	*	*

Здесь 5 прямоугольников.

Задача "на подумать" (Узор)

Знаменитый путешественник и почетный археолог Л.Ю. Бознательный обнаружил, что пол в холе библиотеки очень древний. Он был выложен белыми и черными плитками. Причем черные плитки были очень ценными. Иногда узор разрушался. Тогда поврежденные плитки заменялись другими, но всегда белыми.

Л.Ю. Бознательный выяснил, что первоначальный узор был прямоугольная квадратная "сетка". То есть, сетка была выложена черными плитками шириной в одну плитку по вертикали и горизонтали как на левом рисунке. При укладке плитки не резали!!



В современном узоре сохранилась как минимум одна ячейка сетки.

Написать программу, которая по заданному рисунку пола (черные плитки обозначаются "*", а белые – ".") вычисляет какое минимальное количество черных плиток нужно, чтобы восстановить первоначальный рисунок.

Входные данные

Ввод из файла **pict.dat**.

Простой ввод из файла:

```
./myprog <myfile
```

Первая строка файла два целых числа ($N \leq 400$ и $M \leq 400$), разделенных пробелами – размер рисунка по горизонтали и вертикали

Далее N строк по M символов, описывающих рисунок.






Выходные данные

Вывод на консоль – одно целое число (количество необходимых черных плиток).

Примеры

Вход	Выход
4 7	6
****...	
.....	
.....	
*****.	

-- [TatyanaOvsyannikova2011](#) - 25 Oct 2016

Attachment 	Action	Size	Date	Who	Comment
 mass_int1.png	manage	29.1 K	25 Oct 2016 - 14:52	TatyanaOvsyannikova2011	
 points_mass.png	manage	160.5 K	25 Oct 2016 - 14:53	TatyanaOvsyannikova2011	
 point_funcnt_m.png	manage	19.5 K	25 Oct 2016 - 16:45	TatyanaOvsyannikova2011	
 updown.png	manage	21.1 K	25 Oct 2016 - 19:59	TatyanaOvsyannikova2011	
 pattern.png	manage	9.4 K	31 Oct 2016 - 10:27	TatyanaOvsyannikova2011	

(с) Материалы раздела "Язык Си" публикуются под лицензией [GNU Free Documentation License](#).