1. Указатели

1.1.Адрес переменных.

У каждой переменной есть место хранения в памяти компьютера, которое характеризуется адресом и размером. Размер памяти, выделяемый под хранение той или иной переменной, зависит от ее типа. Примеры размеров ячеек для хранения некоторых типов переменных (для систем 32-bit):

- `int – 4 байта;`
- `float- 4байта;`
- `double – 8 байт;`
- `char – 1 байт.`

Размер занимаемой памяти для любой переменной или типа переменных можно узнать с помощью функции **`sizeof()`**, передав в нее в качестве параметра объект. При этом функция возвращает значение типа `size_t`, которое необходимо преобразовать к целому типу для корректного вывода на экран или записи в переменную.

```
int size_int = (int)sizeof(int);
int size_double = (int)sizeof(double);
int size_float = (int)sizeof(float);
int h = 0;
int size_h = sizeof(h);
```

Ячейки памяти, в которых хранятся значения переменных, имеют адреса. Чтобы узнать адрес переменных необходимо использовать операцию взятия адреса «`&`». С помощью функцию **`printf()`** адрес переменной можно вывести на экран, при этом в строке форматирования адрес переменной указывается как «`%p`»:

```
int a = 7;
printf("a = %d\n", a);    //значение переменной
printf("address = %p\n", &a);    //адрес переменной
printf("size_a = %d\n", (int)sizeof(a));  // размер в байтах

double b[M] = {0};
printf("b[3] = %lf\n", b[3]);
printf("address = %p\n", &b[3]);
printf("size_b[3] = %d\n", (int)sizeof(b[3]));
```

Указатели

Указатель – это переменная, значение которой содержит адрес другой переменной. Имена указателей принято начинать с буквы «p», а для объявления указателей используют символ «*».

```
тип_указателя * имя_указателя = [значение_указателя];
```

```
int * pa;
double * pvar;
int * pfirst, * parray;
```

Для того чтобы указатель указывал на переменную, в него нужно записать адрес этой переменной. Это можно сделать как при объявлении указателя, так и отдельной операцией:

```
int a = 7;
int * pa_1;

// запись адреса переменной в указатель pa_1
pa_1 = &a;

// запись адреса переменной в указатель pa_2 при объявлении
int * pa_2 = &a;
```

На Рисунке 7.1 представлено схематическое расположение переменной и указателя в памяти компьютера для переменной `a` и указателя на эту переменную `pa`.

```
int a = 7;
int * pa;
pa = &a;
```

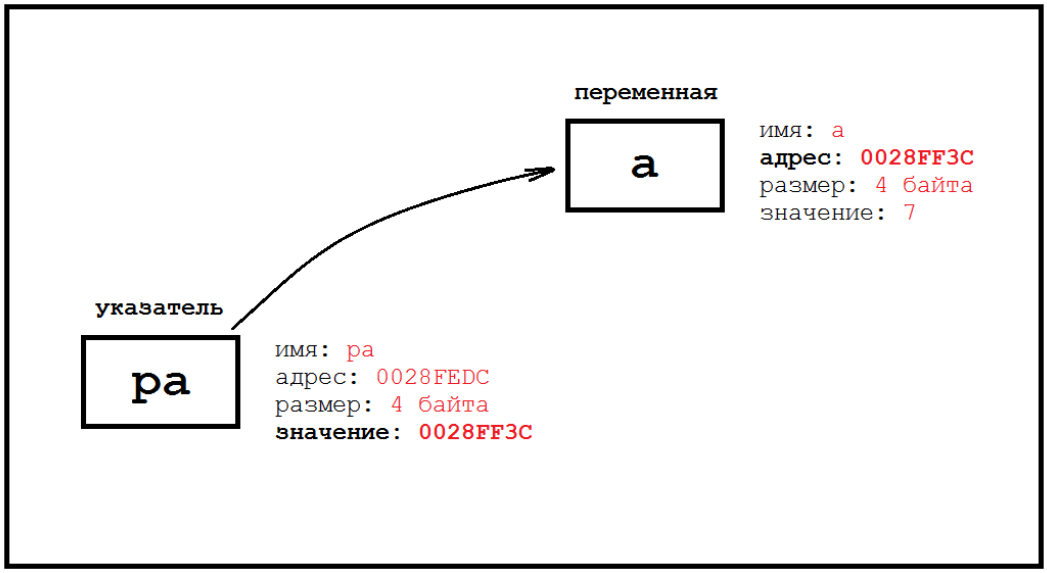

Рисунок 7.1. Схематическое изображение расположения указателя и переменной в памяти. Значение указателя равно адресу переменной.

Указатели предоставляют косвенный способ доступа к переменным через их адрес в памяти компьютера. Для получения такого доступа к переменной используется операция разыменования указателя с помощью символа «*». Например, так можно вывести на экран значение указателя и значение переменной, на которую он указывает:

```
printf("значение указателя = %p", pa);
printf("значение переменной = %d", *pa);
```

Операция разыменования может использоваться для изменения значения переменной, на которую указывает указатель.

```
int a = 7;
int * pa;
pa = &a;
printf("a = %d", a);    //  a = 7

//изменяем значение переменной
//с помощью разыменования указателя
*pa = 15;
printf("a = %d", a);    //  a = 15
```

Если перед указателем поставить знак «*», то мы будем иметь дело не с указателем, а с тем местом, куда он указывает. В нашем случае это переменная `a`.

1.2. Указатели и массивы

Указатели и массивы тесно связаны между собой. Имя любого массива является указателем на первый элемент этого массива. Указателем на каждый последующий элемент массива будет его имя, к которому прибавляется индекс элемента. В памяти компьютера элементы массива расположены следом друг за другом, поэтому, передвигая указатель на позицию вправо (прибавляя единицу) или влево (отнимая единицу), можно обратиться к нужному элементу в массиве. При этом можно заметить, что адреса элементов массива отличаются ровно на то количество байт, которое занимает одна переменная такого же типа как и массив.

Обычная запись	arr[0]	arr[1]	arr[2]	…	arr[N-1]
Запись через указатели	arr	arr + 1	arr + 2	…	arr + N-1
Адреса элементов массива	827280E0	827280E4	827280E8		82728106
Занимаемая память в байтах	4	4	4		4

Чтобы оперировать значениями элементов массива с помощью указателей, используем операцию разыменования. Вывод значений элементов массива с помощью обычной записи и с помощью указателя на элементы массива:

```
//обычный вывод массива на экран
for (i = 0; i < N; i++) {
  printf("%4d", array[i]);
}
//вывод массива с помощью указателей на элементы массива
for (i = 0; i < N; i++) {
  printf("%4d", *(array + i));
}
//вывод адресов элементов массива с помощью указателей
for (i = 0; i < N; i++) {
  printf("%10p", array + i);
}
```

Указатели и аргументы функций

Передача значений аргументов в функцию может происходить двумя способами: с помощью копирования и через указатели.

Передача значений в функцию через копирование данных в значения аргументов функции рекомендуется использовать для сравнительно небольших объектов, таких как целочисленные и дробные переменные, символы.

Вызов функции	**Описание функции**

`int a = 5, b = 7;`
`int summa = sum(a, b);`
	`int sum(int x, int y) {`
	`  return x + y;`
	`}`

Для массивов, переменных файлового типа, структур и подобных переменных рекомендуется использование указателей при передаче их в функцию. При этом копирование переменной не происходит, а происходит лишь передача информации о местонахождении интересующего объекта в памяти компьютера.

Вызов функции	**Описание функции**

`int array[M] = {0};`
`int summa= sum(array, M);`
	`int sum (int * arr, int size_arr) {`
	`  int summa = 0, i = 0;`
	`  for(i = 0; i < size_arr; i++){`
	`  summa += arr[i];`
	`  }`
	`  return summa;`
	`}`

При передаче массива в функцию используется имя массива и указание размерности массива. Одномерный массив может передаваться двумя способами:

- `int sum (int * array, int n)`
- `int sum (int array[N])`

Оба эти способа обеспечивают передачу одномерного массива в функцию по указателю.

Двумерные и многомерные массивы передаются в функцию единственным способом:

- `double sum (double arr[N][M])`

Пример. С помощью функции вычислить сумму целочисленного одномерного массива из 45ти случайных чисел в диапазоне [16; 67].

```
#define N 45
int summa(int *, int);
int main(void) {
  int i = 0, arr[N] = {0};
  for (i = 0; i < N; i++) {
    arr[i] = 16 + rand() % (67 - 16 + 1);
  }
  for (i = 0; i < N; i++) {
    printf("%4d", arr[i]);
  }
  printf("\n");
  int sum = summa(arr, N);
  printf("Сумма массива равна %d", sum);
  return EXIT_SUCCESS;
}

int summa(int * array, int size_arr) {
  int i = 0, sum = 0;
  for (i = 0; i < size_arr; i++) {
    sum += array[i];
  }
  return sum;
}
```

Пример. Для двумерного массива, состоящего из 5ти строк и 9ти столбцов и заполненного случайными целыми числами от -55 до 78, с помощью функций найти и вывести на экран расположения всех элементов равных максимальному.

```
#define N 5
#define M 9
int maximum(int [N][M]);
void location(int [N][M], int);
int main(void) {
  int mas[N][M];
  int i = 0, j = 0;
  for (i = 0; i < N; i++) {
    for (j = 0; j < M; j++) {
      mas[i][j] = -55 + rand() % (78 - (-55) + 1);
    }
```

```
  }
  for (i = 0; i < N; i++) {
    for (j = 0; j < M; j++) {
      printf("%4d", mas[i][j]);
    }
    printf("\n");
  }
  printf("\n");
  int max = maximum(mas);
  printf("Максимальный элемент массива равен %d\n\n", max);
  location(mas, max);
  return EXIT_SUCCESS;
}

int maximum(int mas[N][M]) {
  int i = 0, j = 0, max = mas[0][0];
  for (i = 0; i < N; i++) {
    for (j = 0; j < M; j++) {
      if (mas[i][j] > max) {
        max = mas[i][j];
      }
    }
  }
  return max;
}

void location(int mas[N][M], int max) {
  int i = 0, j = 0;
  printf("Расположение:\n");
  for (i = 0; i < N; i++) {
    for (j = 0; j < M; j++) {
      if (mas[i][j] == max) {
        printf("%d строка, %d столбец\n", i+1, j+1);
      }
    }
  }
}
```

Пример. Для двумерного квадратного массива из 81 элемента, каждый элемент которого равен сумме номера строки и номера столбца, осуществить несколько изменений значений элементов массива и вывести массив на экран после каждого из изменений. Все действия осуществить с помощью функций.

- Умножить каждый четный элемент массива на случайное число в диапазоне от нуля до величины этого элемента.
- Прибавить к соответствующим элементам случайной строки соответствующие элементы случайного столбца.

```
#define N 9
void init_massiv(int [N][N]);
void print_massiv(int [N][N]);
void multiply(int [N][N]);
void add(int [N][N]);
int main(void) {
  srand(time(0));
  int mas[N][N];
  printf("Изначальный массив\n\n");
  init_massiv(mas);
  print_massiv(mas);
  printf("Умножаем четные элементы\n\n");
  multiply(mas);
  print_massiv(mas);
  printf("Прибавляем столбец к строке\n\n");
  add(mas);
  print_massiv(mas);
  return EXIT_SUCCESS;
}

void init_massiv(int mas[N][N]) {
  int i, j;
  for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
      mas[i][j] = (i + 1) + (j + 1);
    }
  }
}
void print_massiv(int mas[N][N]) {
  int i, j;
  for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
      printf("%4d", mas[i][j]);
```

```
        }
        printf("\n");
      }
      printf("\n");
    }
    void multiply(int mas[N][N]) {
      int i, j;
      for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
          if (mas[i][j] % 2 == 0) {
            mas[i][j] *= rand() % (mas[i][j] + 1);
          }
        }
      }
    }
    void add(int mas[N][N]) {
      int j;
      int i_rand = rand() % N;
      int j_rand = rand() % N;
      printf("Прибавляем элементы столбца %d к элементам строки
%d\n\n", j_rand + 1, i_rand + 1);

      for (j = 0; j < N; j++) {
        mas[i_rand][j] += mas[j][j_rand];
      }
    }
```

1.3.Динамическая память

Переменные, функции, константы записываются и хранятся в памяти компьютера во время выполнения программы. Все эти элементы записываются в стек по мере их появления и удаляются оттуда после окончания своей жизни. Например, во время объявления функции происходит выделение памяти для самой функции и ее параметров. Во время вызова функции все ее переменные по очереди записываются в стек (в том порядке, в котором они объявлены). После окончания работы функции стек очищается, и все переменные удаляются из него, но уже в обратном порядке. Таким образом, чем позже была объявлена переменная, тем меньше времени она будет существовать в стеке.

Также существует область динамической памяти, которую называют «Куча» (Heap). Если в стеке все объекты располагаются в определенном порядке, то Куча позволяет более свободно работать с памятью. Она позволяет создавать и удалять

объекты, освобождать память тогда, когда это требуется. Куча предоставляет возможность удалять переменные и объекты не в строгой последовательности, а по мере необходимости, в нужном программисту порядке.
За выделение и освобождение динамической памяти отвечает стандартная библиотека stdlib.h.

Функция выделения динамической памяти malloc()
Функция **malloc**() выделяет необходимое количество памяти из Кучи и возвращает указатель на первый байт области памяти, которая была выделена.

```
указатель = (void*)malloc(размер_памяти_в_байтах);
```

- (**void***) – это операция приведения типа. При выделении памяти мы заменяем **void** на необходимый нам тип.
- размер_памяти_в_байтах находим с помощью функции **sizeof**()

```
// выделяем память для одного целого числа
int * p_int = (int*)malloc(sizeof(int));

// выделяем память для пяти целых чисел
int * p_int = (int*)malloc(5 * sizeof(int));

// выделяем память для N целых чисел
#define N 25
int * p_int = (int*)malloc(N * sizeof(int));

//выделяем память для десяти дробных чисел
double * p_double = (double*)malloc(sizeof(double) * 10);
```

Функция освобождения динамической памяти free()
Функция **free**() освобождает память, на которую указывает указатель, который мы передаем этой функции.

```
free(указатель);
```

```
free(p_int);
free(p_double);
```

Память, которую мы выделяем из Кучи, необходимо обязательно освобождать, так как эта память не очищается после окончания программы. Область динамической

памяти очищается только после перезагрузки компьютера, и если ее вовремя не очистить, то это приведет к утечке памяти, замедлению работы компьютера за счет уменьшения свободной оперативной памяти при каждом запуске программы.

Пример. Создать динамический одномерный массив из `50ти` элементов. Заполнить этот массив случайными числами и вывести на экран в строку. Графическая иллюстрация представлена на Рисунке 3.

```
//создание указателя
int * arr;
//выделение динамической памяти
arr = (int*)malloc(N * sizeof(int));

//работа с динамическим массивом
for(i = 0; i < N; i++) {
  arr[i] = -15 + rand() % 31;
  printf("%5d", arr[i]);
}

//освобождение динамической памяти
free(arr);
```

Рисунок 7.1. Жизненный цикл динамического одномерного массива

Пример. Создать динамический двумерный массив из 56ти элементов. Заполнить этот массив случайными числами и вывести на экран в прямоугольном виде. Графическая иллюстрация представлена на Рисунке 4.

```
    //создание указателя на указатель
    int ** arr;
    //выделение динамической памяти для массива указателей
    arr = (int**)malloc(N * sizeof(int*));
    //выделение памяти для каждого указателя массива указателей
    for(i = 0; i < N; i++) {
        arr[i] = (int*)malloc(M * sizeof(int));
    }

    //работа с динамическим массивом
    for(i = 0; i < N; i++) {
      for(j = 0; j < M; j++) {
        arr[i][j] = -15 + rand() % 31;
        printf("%5d", arr[i][j]);
      }
      printf("\n");
    }

    //освобождение памяти для каждого указателя в массиве
указателей
    for(i = 0; i < N; i++) {
        free(arr[i]);
    }
    //освобождение памяти, содержащей массив указателей
    free(arr);
```

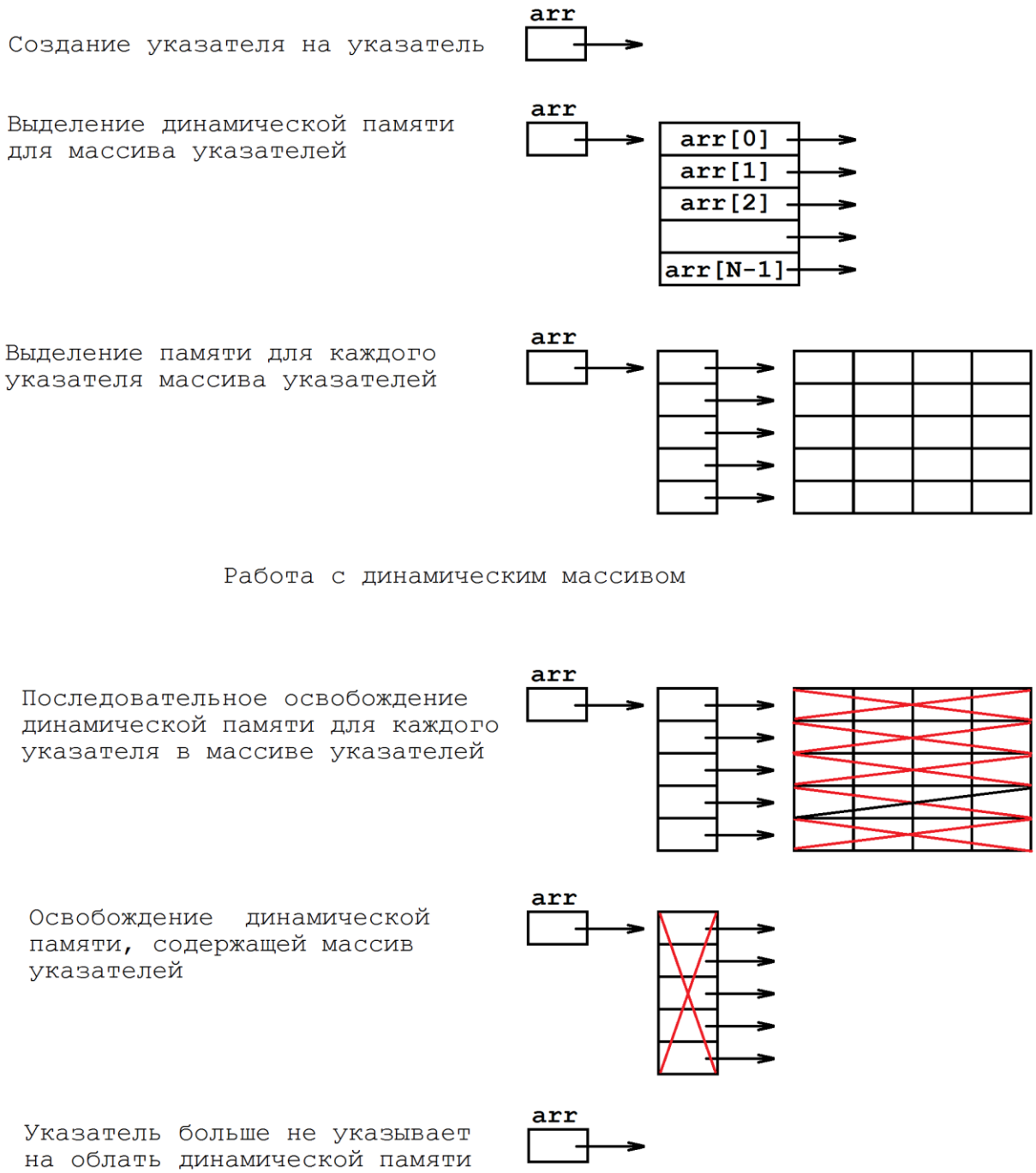

Рисунок 7.2. Жизненный цикл динамического двумерного массива

Передача динамического массива в функцию

Одномерные и двумерные динамические массивы передаются в функции только с помощью знака указателя «*».

Для одномерного динамического массива:

```
void init_array(int*, int);
void print_array(int*, int);
```

```c
int main(void) {
  int i = 0, size_arr = 50;
  int *  arr;
  arr = (int*)malloc(size_arr * sizeof(int));

  init_array(arr, size_arr);
  print_array(arr, size_arr);

  free(arr);
  return EXIT_SUCCESS;
}

void init_array(int * arr, int n) {
  int i = 0;
  for(i = 0; i < n; i++) {
    arr[i] = -15 + rand() % 31;
  }
}

void print_array(int * arr, int n) {
  int i = 0;
  for(i = 0; i < n; i++) {
    printf("%5d", arr[i]);
  }
}
```

Для двумерного динамического массива:

```c
void init_array(int **, int, int);
void print_array(int **, int, int);

int main(void) {
  int i = 0, size_n = 4, size_m = 8;
  int ** arr;
  arr = (int**)malloc(size_n * sizeof(int*));
  for(i = 0; i < size_n; i++) {
    arr[i] = (int*)malloc(size_m * sizeof(int));
  }

  init_array(arr, size_n, size_m);
  print_array(arr, size_n, size_m);

  for(i = 0; i < N; i++) {
```

```
    free(arr[i]);
  }

  free(arr);

  return EXIT_SUCCESS;
}

void init_array(int ** arr, int n, int m) {
  int i = 0, j = 0;
  for(i = 0; i < n; i++) {
    for(j = 0; j < m; j++) {
      arr[i][j] = -15 + rand() % 31;
    }
  }
}

void print_array(int ** arr, int n, int m) {
  int i = 0, j = 0;
  for(i = 0; i < n; i++) {
    for(j = 0; j < m; j++) {
      printf("%5d", arr[i][j]);
    }
    printf("\n");
  }
}
```

Пример. В файле `input.txt` на первой строке записано три целых числа: размерность одномерного массива, начало диапазона, конец диапазона значений элементов массива. Прочитать этот файл, создать соответствующий массив и заполнить его случайными значениями из заданного диапазона. Вывести массив на экран.

```
FILE * file;
if((file = fopen("input.txt", "r")) == 0) {
  printf("Файл не найден");
  exit(1);
}
int size = 0, a = 0, b = 0;
fscanf(file, "%d%d%d", &size, &a, &b);
fclose(file);
```

```
int * arr;
arr = (int*)malloc(size * sizeof(int));

int i = 0;
for(i = 0; i < size; i++) {
  arr[i] = a + rand() % (b - a + 1);
}
for(i = 0; i < size; i++) {
  printf("%5d", arr[i]);
}
free(arr);
```

Пример. Одномерный целочисленный массив заполнен случайными элементами из диапазона [-34; 87]. Образуйте новый одномерный массив из положительных элементов исходного массива.

```
#define N 70
int i = 0, arr[N] = {0};
for(i = 0; i < N; i++) {
    arr[i] = -34 + rand() % (87 - (-34) + 1);
}

//подсчет количества положительных элементов
int count = 0;
for(i = 0; i < N; i++) {
  if(arr[i] > 0) {
    count++;
  }
}

// выделяем память для нового массива
int * new_arr;
new_arr = (int*)malloc(count * sizeof(int));

//заполняем новый массив
int h = 0;
for(i = 0, h = 0; i < N; i++) {
  if(arr[i] > 0) {
    new_arr[h] = arr[i];
    h++;
  }
}
```

```
//освобождаем память
free(new_arr);
```

Пример. Для одномерного целочисленного массива найти минимальный элемент и на основе его расположения в массиве образовать два новых массива. В первый новый массив войдут элементы слева от минимального элемента исходного массива, а во второй новый массив – элементы справа от минимального элемента исходного массива.

```
#define N 7
int arr[N] = {0}, i = 0;
//находим минимальный элемент и его индекс
int min = arr[0], ind_min = 0;
for(i = 0; i < N; i++) {
  if(arr[i] < min) {
    min = arr[i];
    ind_min = i;
  }
}

//создаем новый динамический массив left_arr
int size_left = ind_min;
int * left_arr;
left_arr = (int*)malloc(size_left * sizeof(int));

//создаем новый динамический массив right_arr
int size_right = N - 1 - ind_min;
int * right_arr;
right_arr = (int*)malloc(size_right * sizeof(int));

//заполняем новый массив left_arr
for(i = 0; i < size_left; i++) {
  left_arr[i] = arr[i];
}

//заполняем новый массив right_arr
for(i = 0; i < size_right; i++) {
  right_arr[i] = arr[ind_min + 1+ i];
```

```
}

//освобождаем память
free(left_arr);
free(right_arr);
```

Пример. Образовать одномерный массив из двумерного массива целых чисел, записав туда все элементы двумерного массива, которые больше среднего значения положительных элементов.

```
int arr[N][M] = {{0}}, sum = 0, count = 0;
for (i = 0; i < N; i++) {
  for (j = 0; j < M; j++) {
    if (arr[i][j] > 0) {
      sum += arr[i][j];
      count++;
    }
  }
}
double average = (double)sum / count;

int *  new_arr;
new_arr = (int*)malloc(count * sizeof(int));

h = 0;
for (i = 0; i < N; i++) {
  for (j = 0; j < M; j++) {
    if (arr[i][j] > average) {
      new_arr[h] = arr[i][j];
      h++;
    }
  }
}

free(new_arr);
```

Пример. Для двумерного квадратного массива из 36ти элементов образовать новый одномерный массив из элементов выше главной диагонали. Элементы нового массива должны быть кратны трем.

```
#define N 6
int arr[N][N] = {{0}}, i = 0, j = 0;

// считаем сколько элементов выше главной диагонали кратны трем
int count = 0;
for(i = 0; i < N; i++) {
  for(j = 0; j < N; j++) {
    if((j > i) && (arr[i][j] % 3 == 0)) {
      count++;
    }
  }
}

// выделяем память для нового массива
int * new_arr;
new_arr = (int*)malloc(count * sizeof(int));

// заполняем новый массив
int h = 0;
for(i = 0; i < N; i++) {
  for(j = 0; j < N; j++) {
    if((j > i) && (arr[i][j] % 3 == 0)) {
      new_arr[h] = arr[i][j];
      h++;
    }
  }
}

//выводим на экран новый массив
for(i = 0; i < count; i++) {
  printf("%5d", new_arr[i]);
}

//освобождаем память
free(new_arr);
```