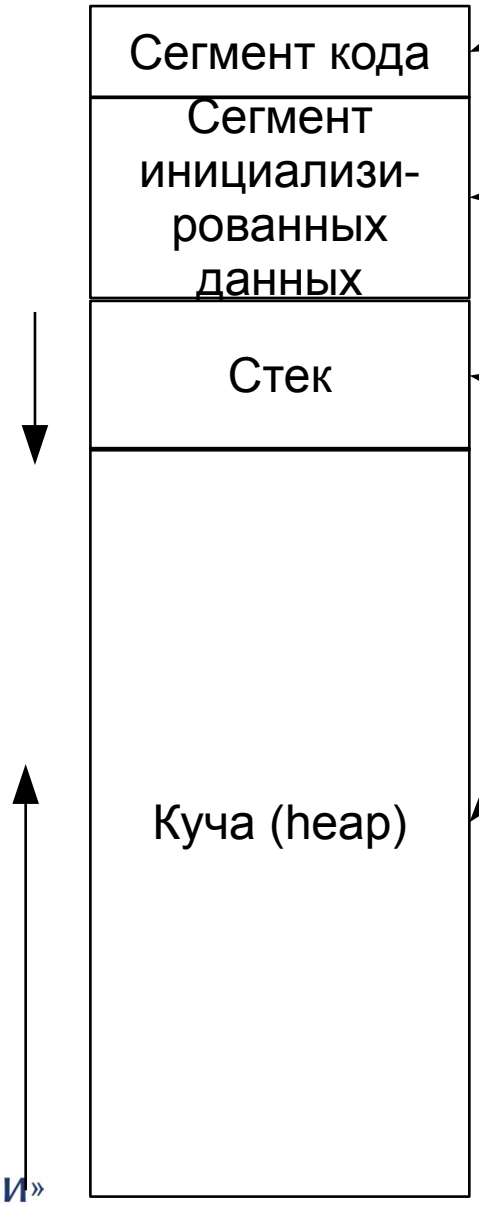
Динамическая память, динамические числовые массивы.

Условная «карта памяти»



Код программы

Переменные с установленными начальными значениями (включая литералы)

Статические локальные переменные, указатели

Динамические переменные

Размер стека ограничен.

Размер кучи — практически вся оперативная память.

Динамическая память

Динамическая память – это часть оперативной памяти, предоставляемая программе для работы.

Размер динамической памяти можно варьировать в широких пределах, всю ее можно использовать для размещения данных.

Динамическое размещение данных означает выделение и освобождение динамической памяти непосредственно во время работы программы. Оперативная память при этом используется наиболее эффективным образом.

Для динамического размещения данных используются указатели.

```
int *p1;
p1 = (int*)malloc(sizeof(int));
```

Функции для работы с динамической памятью — **`malloc()`**, **`calloc()`**, **`realloc()`**, **`free()`** – определены в **`stdlib.h`** (пример lect-04-01.c — см. разницу в адресах).

Динамическая память

Утечка памяти – процесс неконтролируемого уменьшения объёма свободной памяти компьютера, связанный с ошибками в работающих программах, ***вовремя не освобождающих уже ненужные участки памяти***.

Утечки памяти приводят к тому, что ***потребление памяти программой неконтролируемо возрастает***, в результате рано или поздно вступают в действие архитектурные ограничения среды исполнения, и тогда ***новое выделение памяти становится невозможным***. В этой ситуации в программе, которая запрашивает память, обычно ***происходит аварийная остановка***. ***Это может по стечению обстоятельств произойти и совсем с другой программой*** после того, как программа, подверженная утечкам, исчерпает всю память.

Динамическая память

Ошибка сегментирования (***Segmentation fault***) возникает при попытке обращения в область памяти, в которую не следует обращаться (попытка изменения значения константы, выход за диапазон индексов массива, обращение к одному из байтов числа и т.п.).

Ошибка ***double free or corruption*** возникает при попытке вызвать `free()` для уже освобожденного участка динамической памяти.

Динамическая память

malloc(), **calloc()** – выделение памяти

realloc() – изменение размера выделенной памяти.

Для этих функций тип возвращаемого значения не определен (void*), поэтому требуется явное приведение типа.

Всегда требуется проверка успешности выделения памяти

```
if((p1=(int*)malloc(sizeof(int)))!=NULL)
{
    <работаем>
}
else puts("Error at memory allocation");
```

free() — очистка динамически выделенной памяти – **обязательно, автоматически не делается!**

Можно занести значение в известный адрес (пример lect-04-02.c).

Одномерные массивы в динамической памяти

Имя массива является указателем-константой, равной адресу начала массива.
Добавление числа к этому адресу равносильно изменению индекса.

int x[4]

x[0]	x[1]	x[2]	x[3]

```
int *y;
```

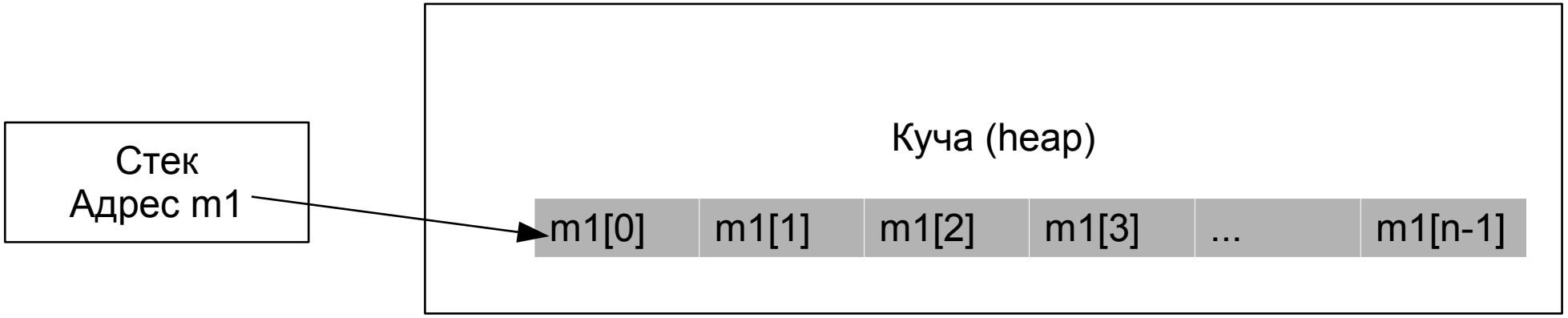
`y=x` равносильно `y=&x[0]`
`y+1` равносильно `&x[1]` и т. п.

`x[2]` равносильно `*(y+2)`

Пример lect-04-03.c

Одномерные массивы в динамической памяти

Если выделять память для массива динамически, то указатель размещается в стеке, сам массив — в куче. Экономится ограниченное пространство в стеке.



Пример lect-04-04.c

Как проверить отсутствие утечек памяти?

Одномерные массивы в динамической памяти

1) Запрашиваем количество элементов
2) Выделяем память динамически (**`calloc()`** или **`malloc()`**) в соответствии с типом элементов
3) Если память успешно выделена (функция вернула не **`NULL`**), то работаем, по окончании работы очищаем память и сбрасываем указатель в **`NULL`**.
4) Если память не выделилась, выдаем сообщение.

Если в процессе работы требуется изменить размер массива, используется **`realloc()`**. При увеличении размера — проверяем на успешность выделения памяти (см. п. 3). При уменьшении размера — не проверяем (нет необходимости).

На схемах алгоритмов действия с динамической памятью не изображаются (специфика языка).

Одномерные массивы в динамической памяти

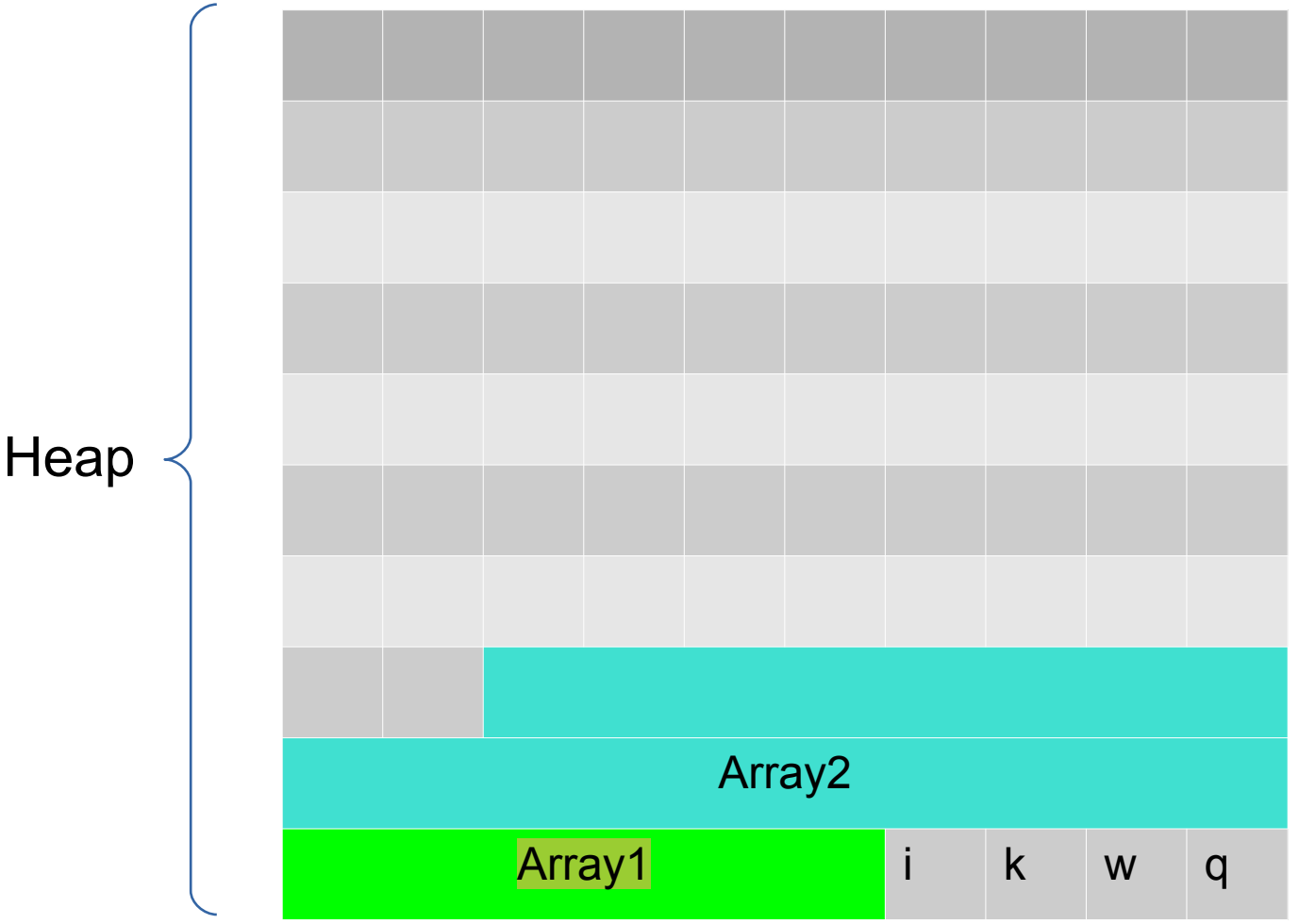
Применение функций выделения памяти

```
float *m1; /* динамический массив вещественных чисел */
int n; /* размер массива */
int k; /* количество добавляемых элементов */
…
scanf("%d",&n); /* получаем размер */
…
m1=(float*)calloc(n,sizeof(float)); /* или */
m1=(float*)malloc(n*sizeof(float));
…
m1=(float*)realloc(m1,(n+k)*sizeof(float)); /*увеличение */
/* исходного массива на k элементов */
```

Функция `calloc()` полезна для числовых массивов, т. к. заполняет массив нулями (инициализирует элементы массива).

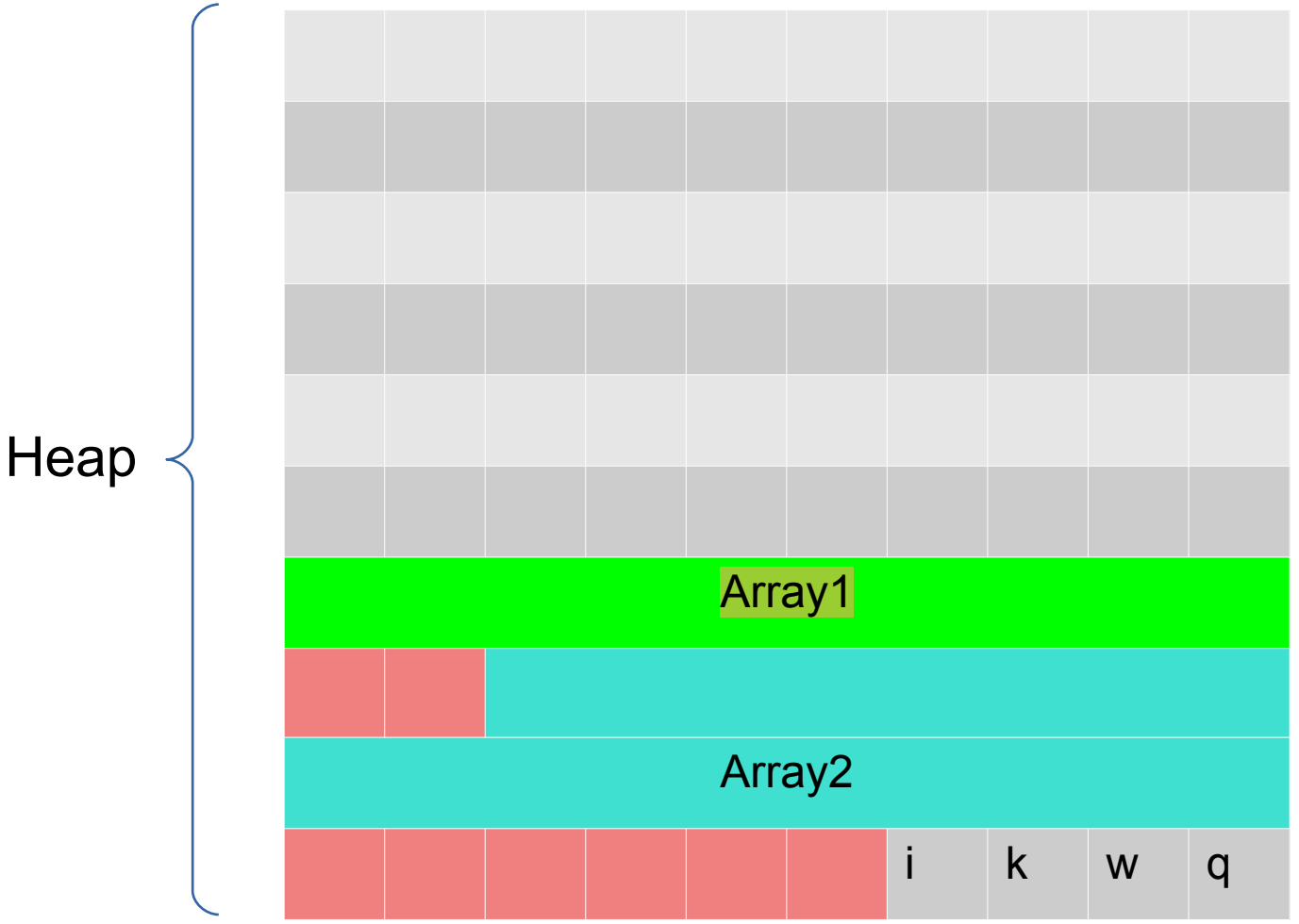
См. примеры lect-04-05.c — lect-04-08.c

Фрагментация памяти



Фрагментация памяти

realloc() на увеличение Array1



Фрагментация памяти

При частом использовании **`realloc()`** может возникнуть ситуация, когда есть много маленьких пустых областей памяти, в которые нельзя поместить новые объекты (массивы и т. п.).
Память есть, но ее нельзя использовать.

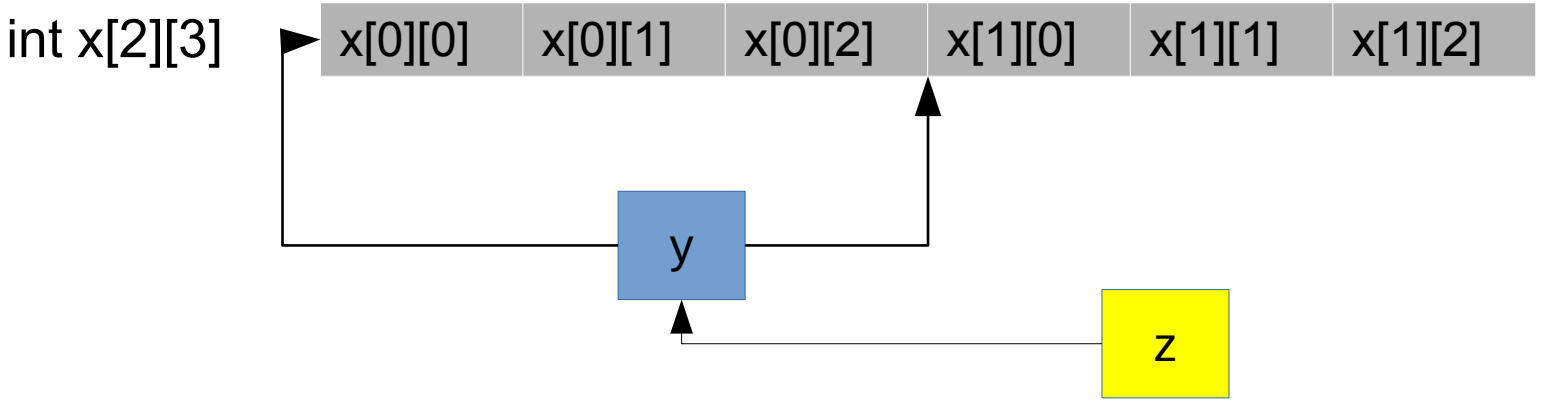
Это вредный эффект.
К применению **`realloc()`** нужно относиться с осторожностью.

Двумерные массивы в динамической памяти

- Имя двумерного массива является указателем-константой на массив указателей-констант.
- Элементами массива указателей являются указатели-константы на начало каждой из строк массива.
- С элементами двумерного массива можно работать с помощью индексов, а можно использовать механизм указателей.
- Для указателей точкой отсчета может служить как первый элемент строки, в которой находится искомый элемент, так и первый элемент двумерного массива

Пример lect-04-09.c.

Двумерные массивы в динамической памяти



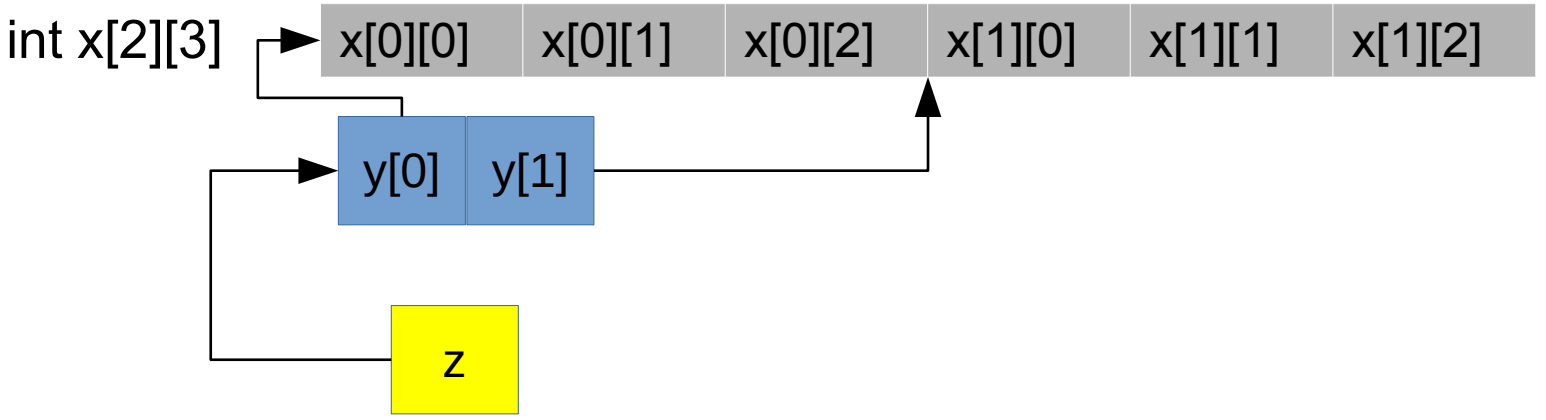

```
int *y;
```

Тогда возможны присваивания `y=x[0]` или `y=x[1]` (адрес первого элемента первой или второй строки, куда-то записывается)

```
int **z;
```

z — адрес какого-то из вариантов y.

Массивы указателей




```
int *y[2];
```

y — массив из двух указателей (пример lect-04-10.c).
Первый элемент — указатель на начало первой строки массива x,
второй элемент — указатель на начало второй строки массива x.

```
int **z;
```

z — адрес массива y.

Динамические двумерные массивы

1) Выделяем память для массива указателей, если успешно — то
2) Выделяем память для указателей на строки, если успешно, то работаем. Иначе очищаем память, выделенную ранее для строк, очищаем память массива указателей, завершаем работу
3) При удачном выделении — работаем, потом очищаем память, выделенную для строк, очищаем память массива указателей, завершаем работу.

Двумерный массив «разворачивается» в куче в одну строку → строки двумерного массива не обязаны быть одинаковой длины!

Примеры lect-04-11.c, lect-04-12.c

Классы памяти

В Си определено 4 (четыре) различных варианта (класса) памяти:

- auto
- register
- static
- external

Обычное определение переменных (`int a;`) соответствует автоматической памяти (класс `auto`), переменные размещаются в стеке (`int auto a;`).

Глобальные переменные (объявленные вне модулей) не могут размещаться в автоматической памяти. Они размещаются в сегменте инициализированных данных.

Если переменная объявленная как регистровая (`register int a;`), то к ней нельзя обращаться по адресу. Компилятор пытается разместить ее в регистрах процессора.

Классы памяти

Класс **`static`** (**`static int a=2;`**) имеет смысл при использовании функций.

Такая переменная не уничтожается при выходе из функции и сохраняет последнее вычисленное значение. При повторном вызове функции такая переменная заново не инициализируется, используется сохраненное ранее значение.

Переменная класса **`static`** не может быть инициализирована вызовом функции.

Переменная класса **`static`** доступна только в том модуле, в котором объявлена. Такие переменные хранятся в сегменте инициализированных данных.

Глобальная переменная класса **`static`** доступна только в пределах «своей» единицы трансляции (файла .c).

Переменная, объявленная как **`extern`**, может быть использована в других единицах трансляции (файлах) при условии, что она была определена.