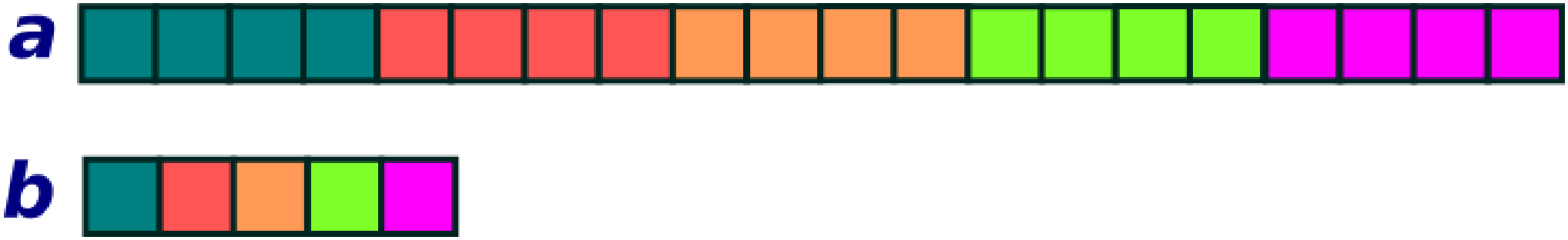
Сортировки

Основы языка C, лекция 10

Указатель на функцию

- int **add** (int x, int y) { // реализация
  return x + y;
  }
- int **add** (int, int); // прототип
- переменная func типа указатель на функцию
  int **(* func)**(int, int);
- func = add;
- add (3, 5);
  func (3, 5);
- **typedef** int **(*BinOp)**(int, int);
  BinOp f2 = add;

Сумма элементов массива



C99

- Дано число N (0 < N ≤1000)
  далее N целых чисел. Напечатать их дважды.

```
5
7 -1 3 12 8
```

-
```
int main ( ) {
    int n;
    scanf("%d", &n);
    int a[n];
    for (int i = 0; i < n; i++)
        scanf("%d", &a[i]);      // a+i

    print_arr(a, n);   print_arr(a, n);
    return 0;
}
```

N не дано

- ~~Дано число N~~ (0 < N ≤1000)
  далее N целых чисел. Напечатать их дважды.

- ```
  #define N 1000
  int main ( ) {
      int a[N], n, i;
      scanf("%d", &n);
      for (i = 0; i < ?; i++)
          scanf("%d", &a[i]);       // a+i

      print_arr(a, n);    print_arr(a, n);
      return 0;
  }
  ```

```
7 -1 3 12 8
```
```

Что возвращает scanf

- int x, y, res;
  res = scanf("%d%d", &x, &y);
- scanf возвращает количество правильно разобранных format placeholder (форматных спецификаций)

input	res	x	y
12 34	2	12	34
13abc	1	13	?
abc25	0	?	?
hello	0	?	?
конец файла	EOF	?	?

Что возвращает scanf

- int x, y;
  while (2 == scanf ("%d%d", &x, &y)) {
  ....
  }

```
12 34
-6 15
88 -21
```

input	res	x	y
12 34	2	12	34
13abc	1	13	?
abc25	0	?	?
hello	0	?	?
конец файла	EOF	?	?

N не дано, но ограничено

- ~~Дано число N~~ (0 < N ≤1000)
  далее N целых чисел. Напечатать их дважды.

- ```
  #define N 1000
  int main ( ) {
      int a[N], i = 0 ;
      while (1 == scanf("%d", &a[i] ) )
          i++;
      int n = i;
      print_arr(a, n);
      print_arr(a, n);
      return 0;
  }
  ```

```
7 -1 3 12 8
```
```

не знаем чем ограничено N

- Дано число N ~~(0 < N ≤1000)~~
  далее N целых чисел. Напечатать их дважды.

- ```
  int main ( ) {
      int n;
      scanf("%d", &n);
      int a[n];
      for (int i = 0; i < n; i++)
          scanf("%d", &a[i]);       // a+i
      print_arr(a, n);
      print_arr(a, n);
      return 0;
  }
  ```

```
5
7 -1 3 12 8
```
```

Без C99

- Дано число N ~~(0 < N ≤1000)~~
  далее N целых чисел. Напечатать их дважды.

```
5
7 -1 3 12 8
```

-
```
int main ( ) {
    int n;
    scanf("%d", &n);
    int * a = malloc( n * sizeof(int)); // взяли int a[n];
    for (int i = 0; i < n; i++)
        scanf("%d", &a[i]);      // a+i
    print_arr(a, n);
    print_arr(a, n);
    free (a);                                  // отдали
    return 0;
}
```

Функции работы с памятью

- #include <stdlib.h>
  void * **malloc** (size_t **size**);
  void * **calloc** (size_t **nmemb**, size_t **size**);
  void * **realloc** (void * **ptr**, size_t **size**);
  void **free** (void * **ptr**);
- malloc — выделить память размером size и вернуть на нее указатель
- calloc — выделить память размером nmemb*size, заполнить ее 0, вернуть указатель на память
  a = **calloc** (n, sizeof(int));
- free — вернуть выделенную память ptr

free (ptr)

- указатель на **динамически** выделенную память

  int * a = malloc(10); .... ; free(a); // ok

  char b[10]; .... free(b); // error

- указатель на **начало** выделенной памяти

  int * a = malloc(10); int * b = malloc(10);

  - free(a); // ok
  - free(b +2); // error

- free (**NULL**); - ничего не делает
- free(a); free(a); - undefined behaviour

Ничего о N неизвестно

- Дано N целых чисел. Напечатать их дважды.
- int main ( ) {

```
    int n, i = 0;                                          7 -1 3 12 8
    int * a = malloc( 1 * sizeof(int));  // взяли для 1 int
    while (1) {
        if (scanf("%d", &a[i]) != 1)          // &a[n]
            break;
        i ++;
        a = realloc(a, (i+1)*sizeof(int));    // взяли новое
    }
    n = i;  print_arr(a, n);   print_arr(a, n);
    free (a);                                  // отдали
```

newptr = realloc (ptr, size)

- захватить память размером size
- откопировать в нее память из ptr
  (если размеры разные?)
- освободить старую память, на которую указывал ptr
- вернуть указатель на новую память

Обработка ошибок

- Что будет, если не хватит памяти?
  - ничего не делает, возвращает NULL

```
ptr = realloc(ptr, new_size);      // потеряли ptr
```

- Безопасный код:

```
new_ptr = realloc(ptr, new_size);
if (new_ptr == NULL)
    обработка ошибки (с выходом?)
ptr = new_ptr;
```

realloc(NULL, size) как malloc

- Дано N целых чисел. Напечатать их дважды.
- int main ( ) {

```
    int i = 0;
    int * a = NULL;
    while (1) {
        a = realloc(a, (i+1)*sizeof(int));    // взяли новое
        if (scanf("%d", &a[ i ]) != 1)
            break;
        i ++;
    }
    int n = i; print_arr(a, n);   print_arr(a, n);
    free (a);                                  // отдали
```

7 -1 3 12 8

nsize — выделено памяти

7 -1 3 12 8

```
int main ( ) {
    int i = 0;// чисел прочитали
    int nsize = 10;  // выделено памяти под nsize чисел
    int * a = malloc(a, (nsize)*sizeof(int));
    while (1) {
        if (scanf("%d", &a[ i ]) != 1)
            break;
        i ++;
        if (i >= nsize) {
            nsize += 10;            // увеличим
            a = realloc(a, nsize*sizeof(int));
        }
    }
```

Единообразно

7 -1 3 12 8

```
int main ( ) {
    int i = 0;// чисел прочитали
    int nsize = 0;  // выделено памяти под nsize чисел
    int * a = NULL;
    while (1) {
        if (i >= nsize) {
            nsize += 10;            // увеличим
            a = realloc(a, nsize*sizeof(int));
        }
        if (scanf("%d", &a[ i ]) != 1)
            break;
        i ++;
    }
```

функции работы с памятью

- #include <string.h>
  void * **memset** (void ***s**, int **c**, size_t **n**);
  от указателя **s** заполнить **n** байт значением **c**.
- заполнить массив a нулями:
  int a[10];
  memset(a, 0, sizeof(a));
- можно ли написать memset, который делает то же самое (записывает 10 единиц)?
  for(int i = 0; i < 10; i++)
  a[i] = 1;

функции работы с памятью

- #include <string.h>
  char * **strcpy** (char ***dst**, const char * **src**);
  void * **memcpy** (void ***dst**, const void * **src**, size_t **n**);
  void * **memmove** (void ***dst**, const void * **src**, size_t **n**);
- strcpy — из указателя src копируем в указатель dst **до \0**, возвращаем dst;
- memcpy — из указателя src копируем в указатель dst **n байт**, возвращаем dst;
- memmove — из указателя src копируем в указатель dst **n байт, можно перекрывающиеся участки памяти**, возвращаем dst; - копирование через буфер

void * в разных языках

- C: автоматическое преобразование указателей к/из void *

```
char * a = malloc(6);       // void * → char *
a = realloc(a, 12);         // char * →void *
```

- C++ типы НЕ преобразуются автоматически

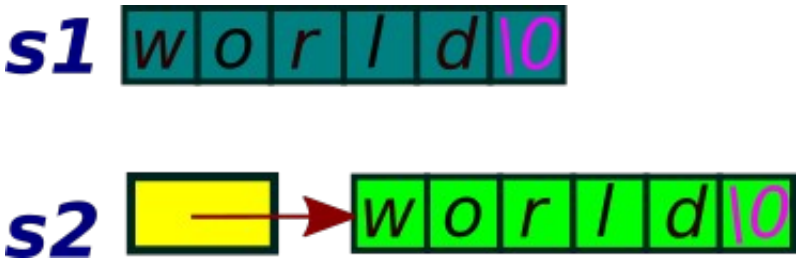
```
char * a = (char *) malloc(6);
a = (char *) realloc( (void*)a, 12);
```

- Поэтому в C++ malloc и realloc не используют. Вместо них используется new, возвращающий нужный тип:

```
int * a = new int[10];
delete [ ] a;
```

Выделение памяти под строку

- char s1[ ] = "world";
  strlen(s1); // 5
  sizeof(s1); // 6



- char * s2 = "world";
  strlen(s2); // 5
  sizeof(s2); // 8 на 64-бит
- char * str = malloc(strlen(s2) + 1);
  strcpy(str, s2);
- char * str = **strdup** (s2);

Прочитать 1 слово

- char * name;
  scanf ("**%ms**", &name);
  ...
  free (name);

Прочитать 1 строку

- #include <stdio.h>
  ssize_t **getline** (char ****lineptr**, size_t ***n**, FILE ***stream**);
- ssize_t **getdelim**(char ****lineptr**, size_t ***n**, int **delim**, FILE ***stream**);
- **getline** считывает из потока **stream** строку, сохраняя ее в памяти по адресу ***lineptr** размером **n** байтов.
- *lineptr == NULL, то getline выделит память под буфер, ее нужно потом освободить free
- Функция изначально была расширением GNU и была внесена в стандарт POSIX.1-2008.

Прочитать по 1 строке пример

```
char * line = NULL;
size_t len = 0;
FILE * fp = fopen("data.txt", "r");
if (fp == NULL)
    exit(EXIT_FAILURE);
ssize_t n;
while ((n = getline (&line, &len, fp)) != -1) {
    printf("Retrieved line of length %zu :\n", n);
    printf("%s", line);
}
if (line)
    free (line);
return EXIT_SUCCESS;
```

Возвращаемое значение

- getline считывает строку в *lineptr
  - буфер завершается **'\0'**;
  - если во входном потоке был **\n**, то он тоже попадает в буфер;
    если в последней строке наступил конец файла и не было \n, то в буфер \n не запишется;
- при необходимости для *lineptr делается realloc, новые значения размера буфера записываются в n;
- getdelim — позволяет использовать вместо \n указанный аргументом delim алфавит разделителей
  getdelim (**&line, &len, ".,;!\n", fp**)

Как это работает?

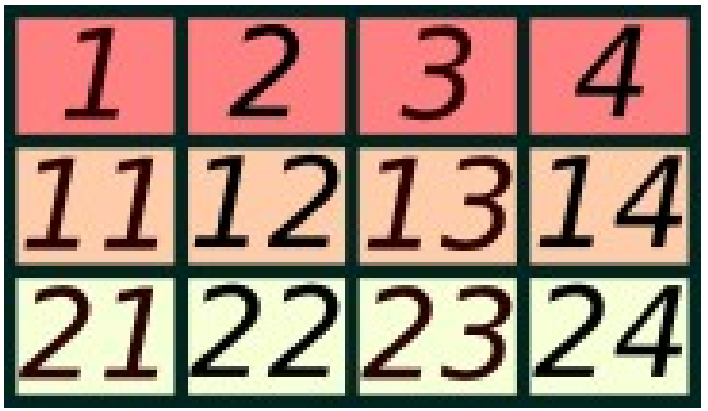
- обе функции возвращают количество считанных символов (с учетом разделителя, если он был), но без учета \0.
  - При ошибке возвращается -1.

valgrind

- Компиляция — добавим отладочную информацию
  gcc -Wall -Wextra **-g** -o hello.exe hello.c
- Запуск без valgrind
  ./hello.exe
  ./hello.exe < data.txt
- Запуск с valgrind
  valgrind ./hello.exe
  valgrind ./hello.exe < data.txt

матрица пример

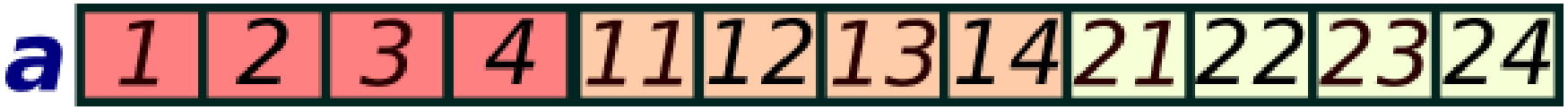
- Матрица в аналитической геометрии



- Хранение двухмерных массивов в C

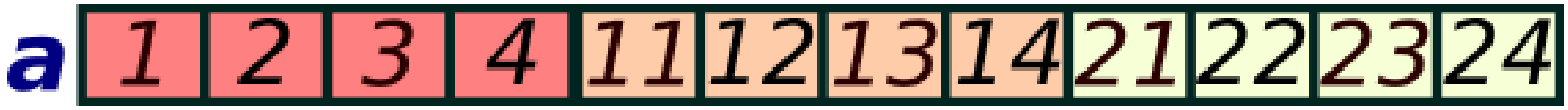
  int a[3][4] = {{1, 2, 3, 4}, {11, 12, 13, 14}, {21, 22, 23, 24}};

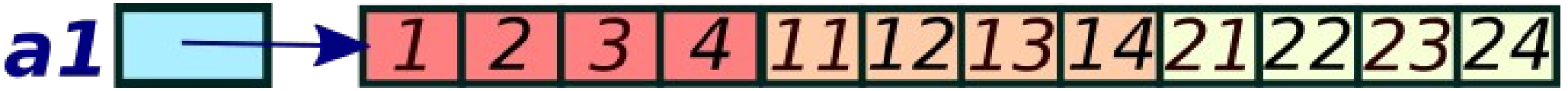
По строкам



- int a[3][4] = {
  {1, 2, 3, 4},
  {11, 12, 13, 14},
  {21, 22, 23, 24}
  };
  for (int i = 0; i < 3; i++)
  for(int j = 0; j < 4; j++)
  printf("a[%d][%d]=%d at %p\n",
  i, j, a[i][j], &a[i][j]);

Динамическая память



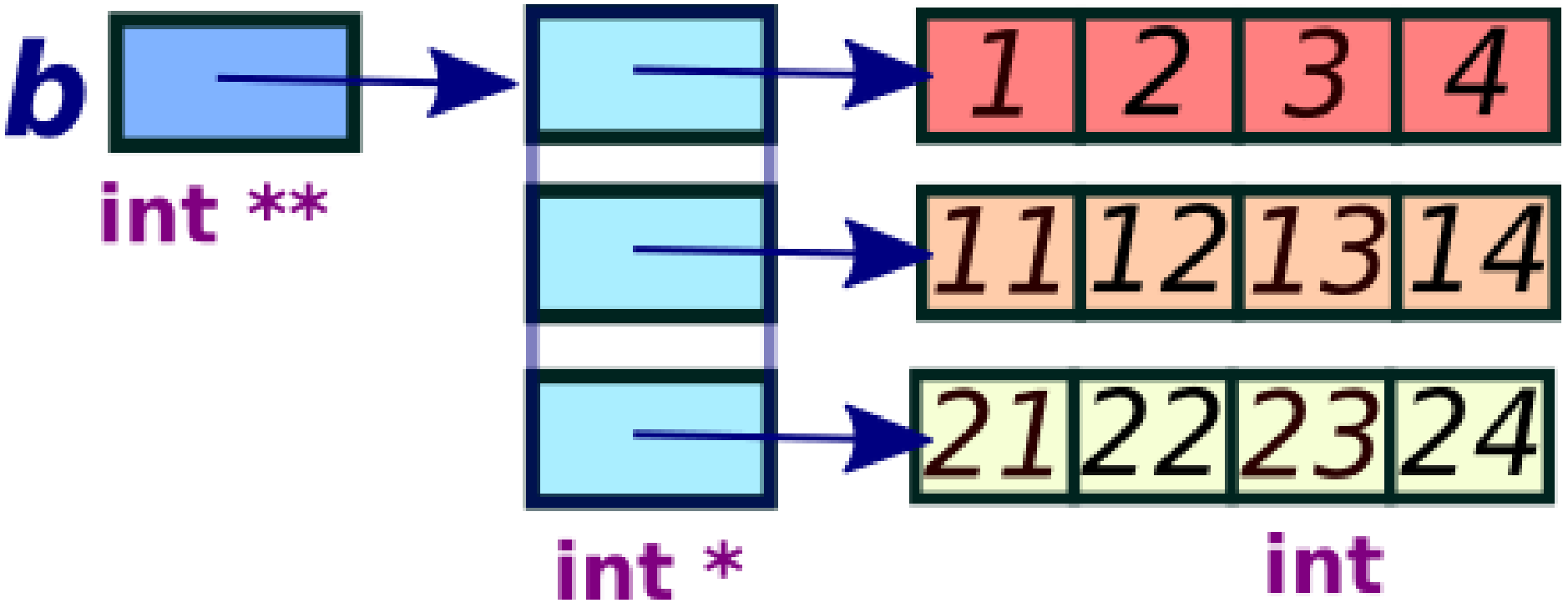


- int a[3][4];
  int * a1 = malloc (3 * 4 * sizeof(int));
  memcpy (a1, a, 3 * 4 * sizeof(int));

```
for (int i = 0; i < 3; i++)
    for(int j = 0; j < 4; j++)
        printf("a[%d][%d]=%d at %p\n",
                    i, j, a[i][j], &a[i][j]);   // a[i][j] error
```

a[1][2]
***(a1 + 4*1 + 2)**
a1[4*1 + 2]

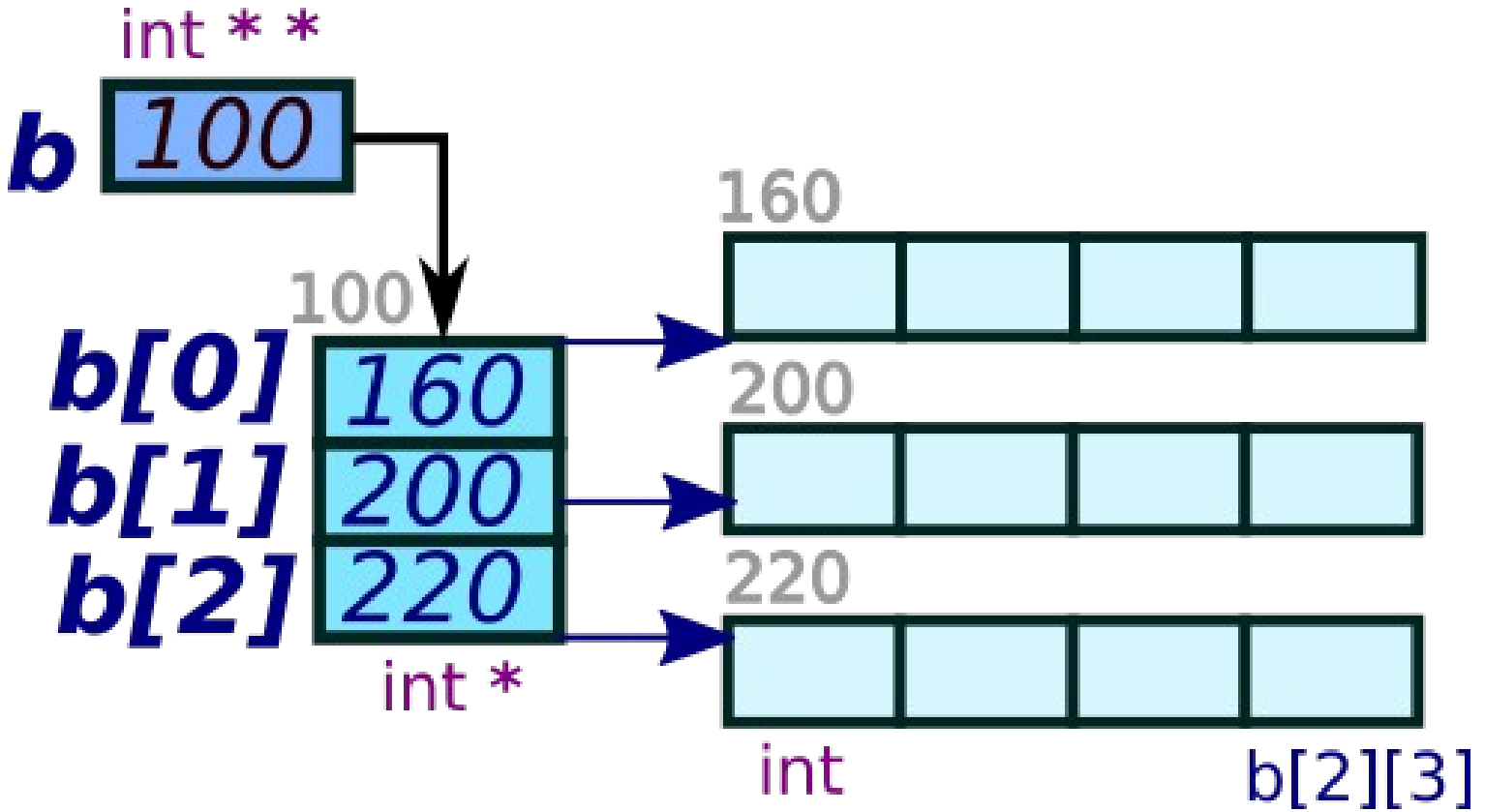
Классический способ



Реализация

- **int **** b = **malloc** (3 * sizeof(**int***));

```
for (int i = 0; i < 3; i++) {
    b[i] = malloc ( 4 * sizeof(int) );
    memcpy(b[i], a[i],  4 * sizeof(int));
}
```

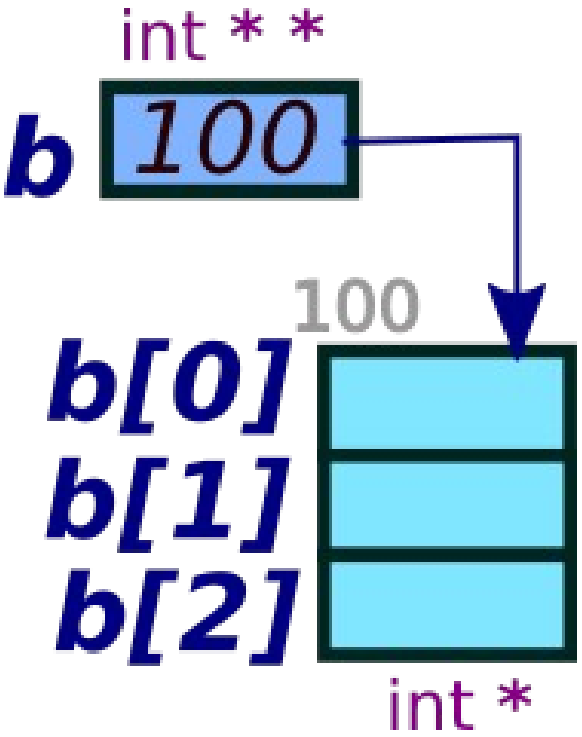


Объявление переменной

- **int **** b;

Массив указателей

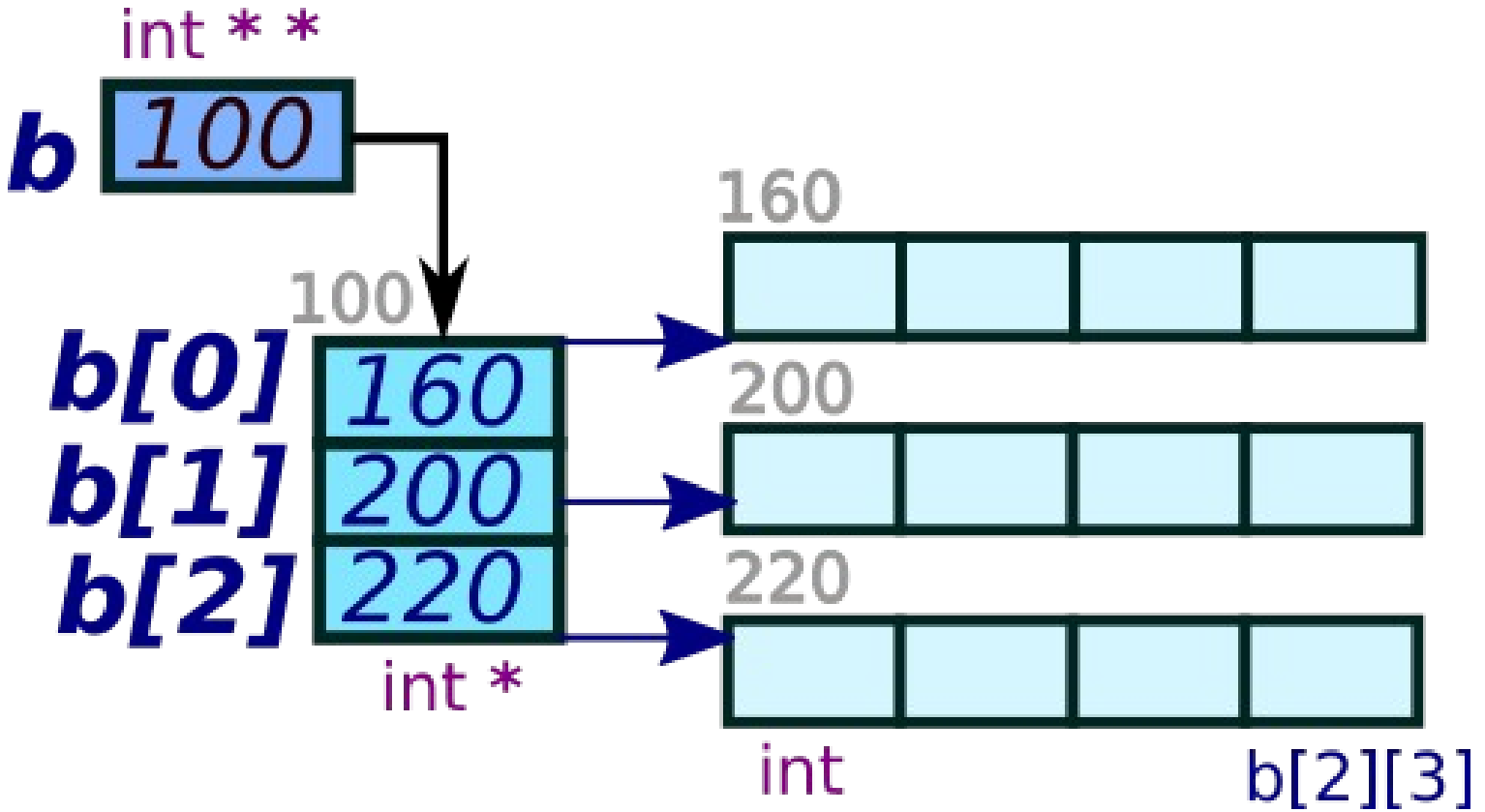
- **int **** b = **malloc** (3 * sizeof(**int***));



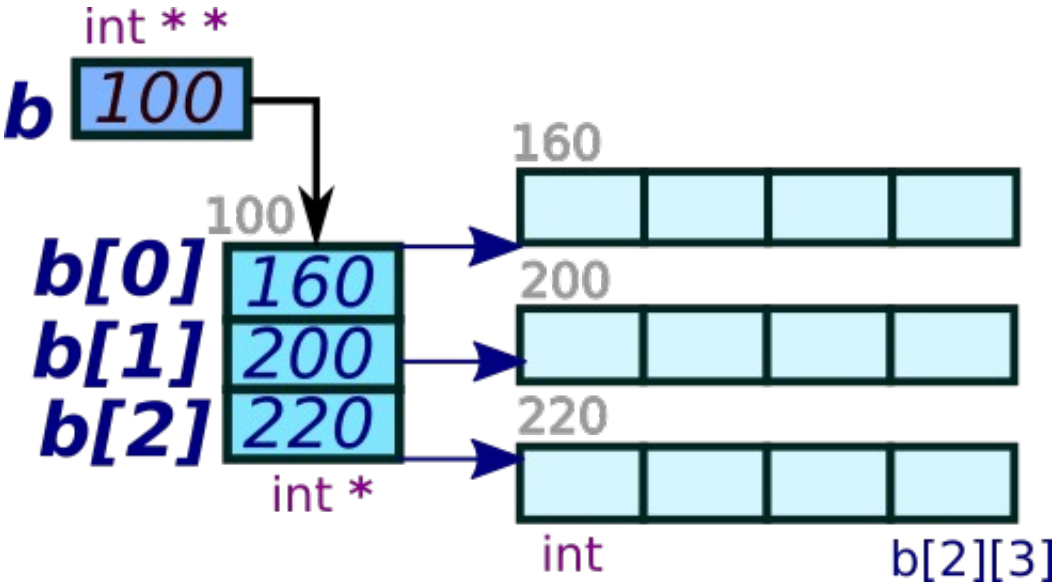
Выделяем память для данных

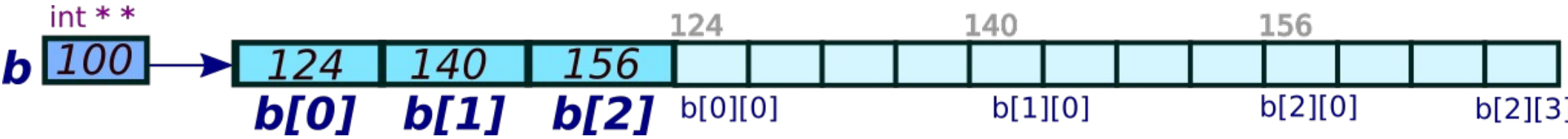
- **int **** b = **malloc** (3 * sizeof(**int***));

```
for (int i = 0; i < 3; i++) {
    b[i] = malloc ( 4 * sizeof(int) );
    memcpy(b[i], a[i],  4 * sizeof(int));
}
```

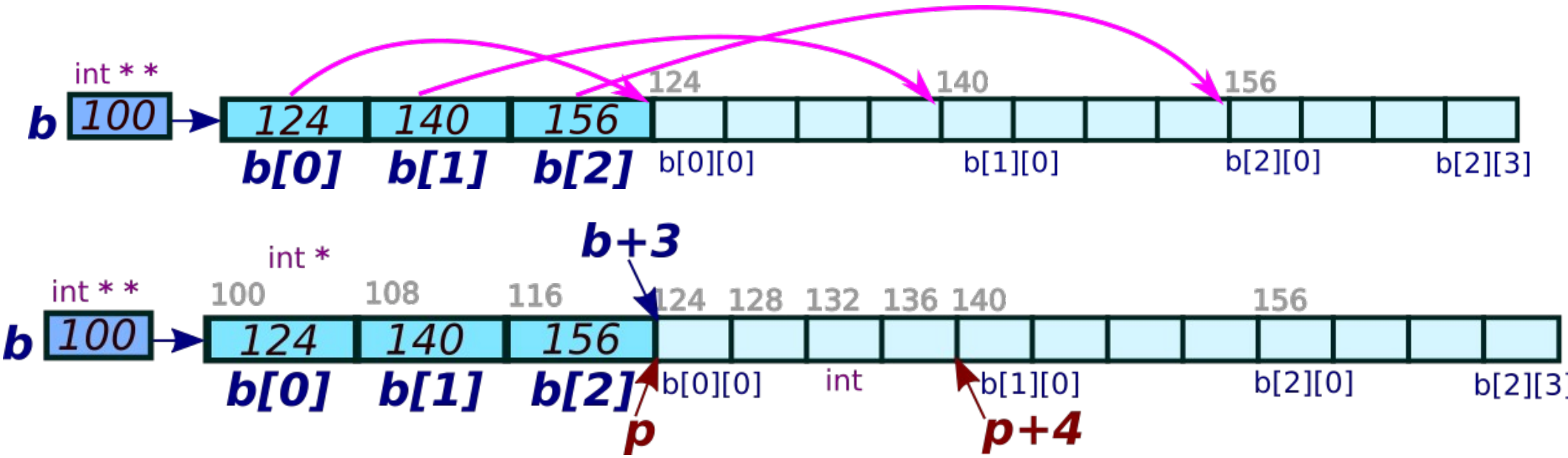


В 1 malloc с работой b[i][j]





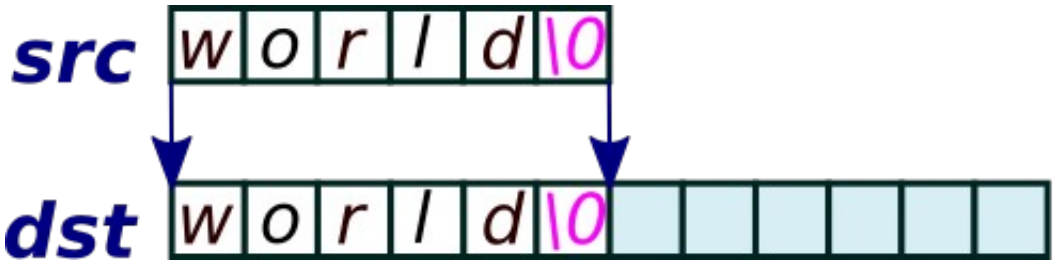
Куда указывают b[i]?

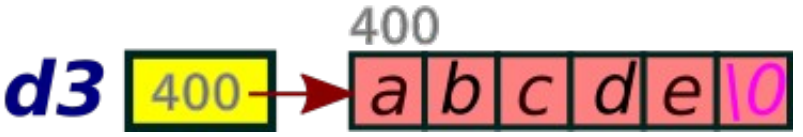


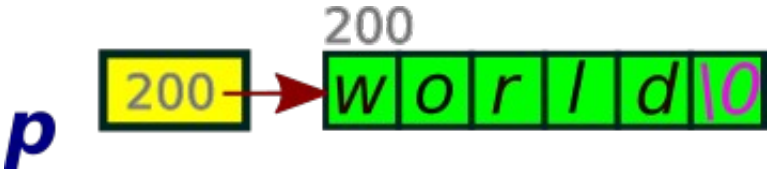
- **int **** b = malloc (3*sizeof(**int***) + 3*4*sizeof(**int**));
  int * p = (int *) (b + 3); // (int*) &b[3] p = 124
  memcpy(p, a, 3*4*sizeof(int));
  b[0] = p; p = p + 4; // p = 140
  b[1] = p; p = p + 4; // p = 156
  b[2] = p;

Выделение памяти + strcpy

- char src[ ] = "world";
- char dst[100];
  strcpy(dst, src);
- char * d1; // куда?
  strcpy(d, "world");
- char * d2 = NULL;
- char * d3 = "Good bye!";
- char * p = malloc(strlen(s) + 1);
  strcpy(p, s);
- char * p = strdup(s);

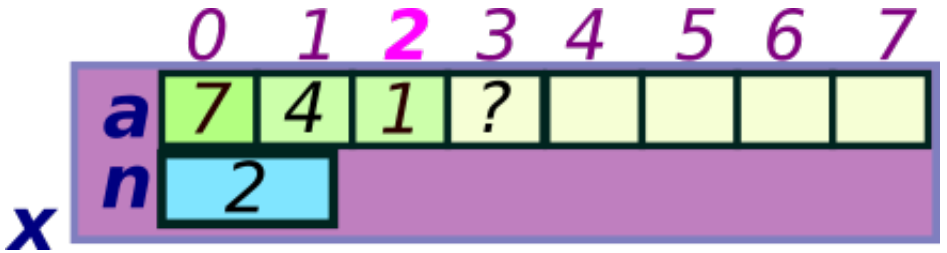






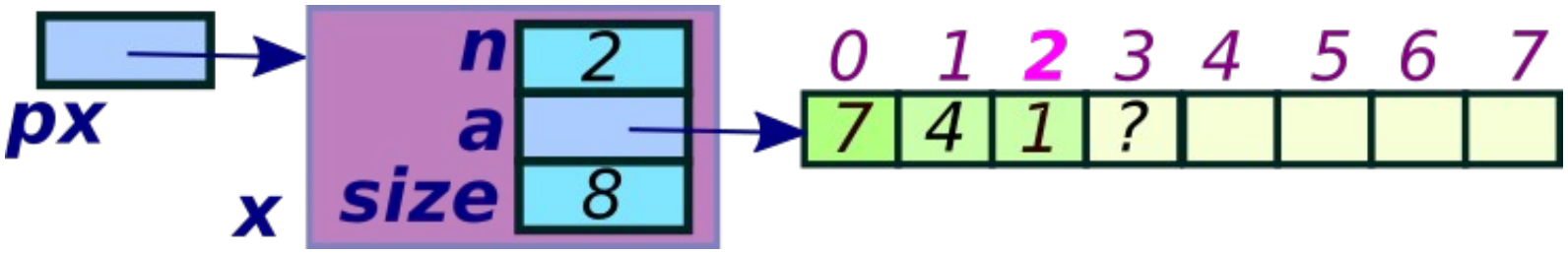
elong 147 = $7*10^0 + 4*10^1 + 1*10^2$

- typedef struct {
  unsigned int n;
  unsigned char a[100];
  } Decimal;
  Decimal x = {2, {7, 4, 1}}; // 147
  Decimal zero = { 0, {0} }; // 0



- void print(Decimal d);
- Decimal **set_int** (unsigned int x);
  Decimal y = **set_int**(147);
- void **set_int** (Decimal * res, unsigned int x);
  Decimal b; set_int(&b, 147);

Массив переменной длины



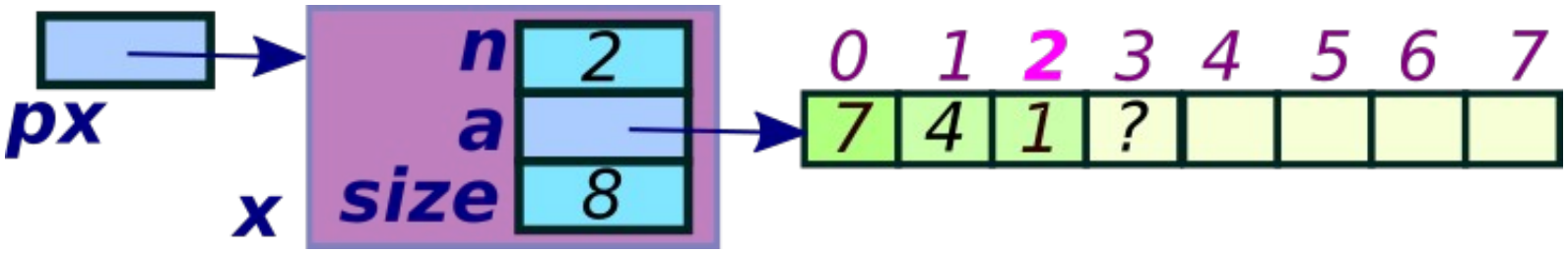
- typedef struct {
  unsigned int n;
  unsigned char * a;
  unsigned int size;
  } Decimal;

  Decimal x;
  set_number(&x, 147);

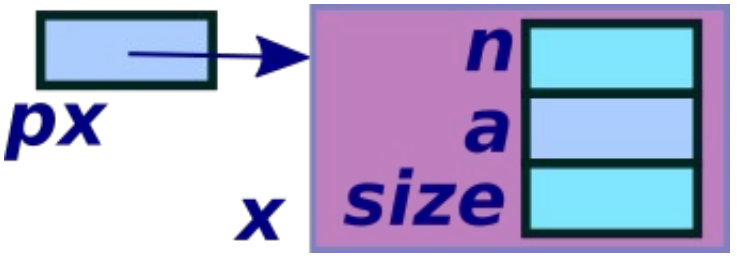
- Decimal x;
  x.n = 2;
  x.size = 3;
  x.a = malloc(x.size);
  x.a[0] = 7;
  x.a[1] = 4;
  x.a[2] = 1;

  Decimal * px = &x;

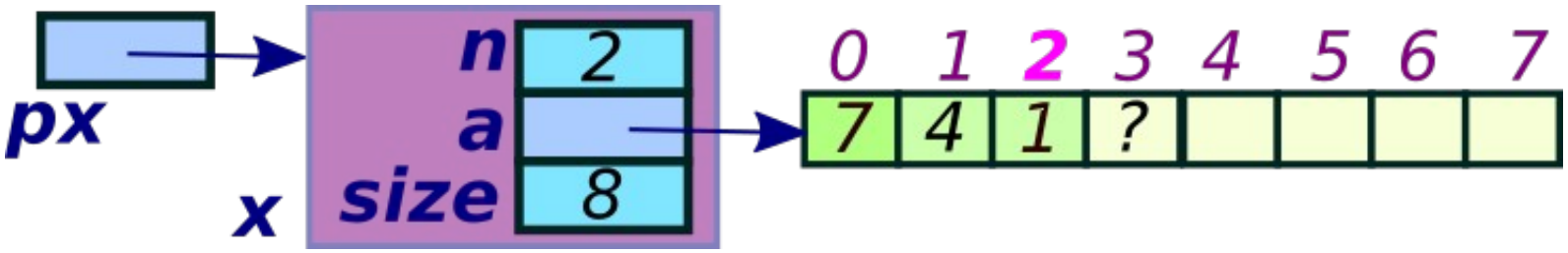
Память только под Decimal



- void set_int (Decimal * px, int number) {
  px→size = 100;
  px→a = malloc(px→size);
  ....
  }

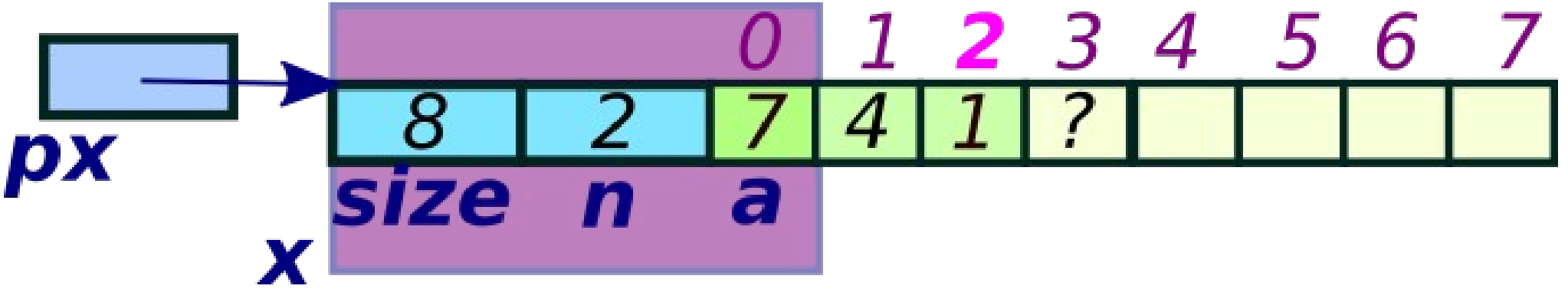


Все надо выделять самим



- Decimal * set_int (int number) {
  Decimal * px = malloc(sizeof(Decimal));
  px→size = 100;
  px→a = malloc(px→size);
  ....
  return px;
  }
  // Сколько нужно free ?

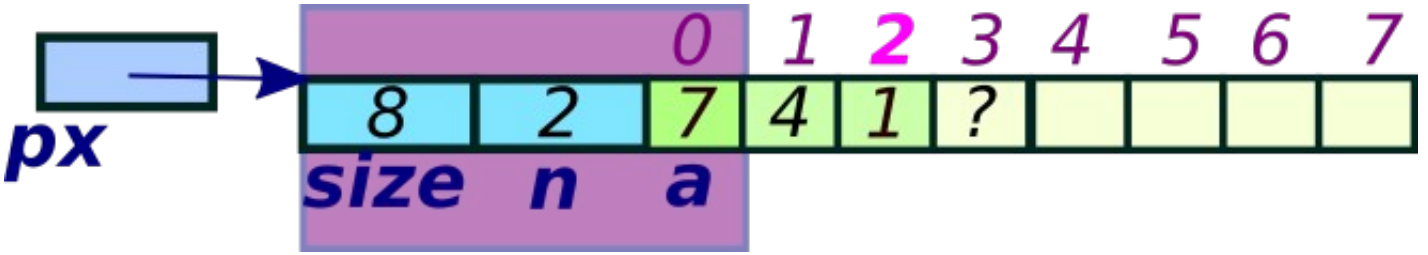
Единым malloc



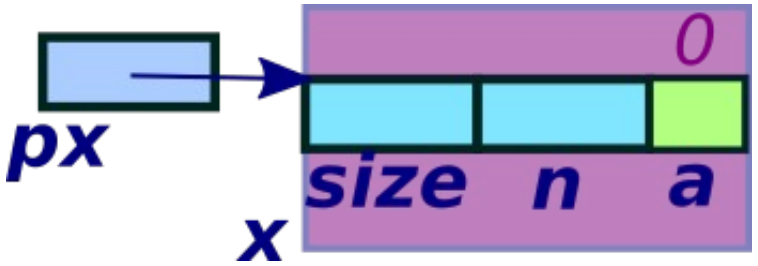
- typedef struct {
  unsigned int size;
  unsigned int n;
  unsigned char a[1]; // a[0]
  } Decimal;
  Decimal * px;
  px = malloc(sizeof (Decimal) + (8-1));

Единым malloc

- Decimal * px;
  px = malloc(sizeof (Decimal) + (8-1));
  px→size = 8;
  px→n = 2;
  px→a[0] = 7;
  px→a[1] = 4;
  ...
  free(px);



- // бессмысленно
  Decimal x;
  Decimal * px = &x;



replace

- char * **replace**(char * dst, const char * src) {
  char * s = src; // откуда ищем бомбу
  char * p; // указатель на найденную бомбу
  // или NULL
  char * d = dst; // куда копировать
  // тут заменяем
  return dst;
  }