数组

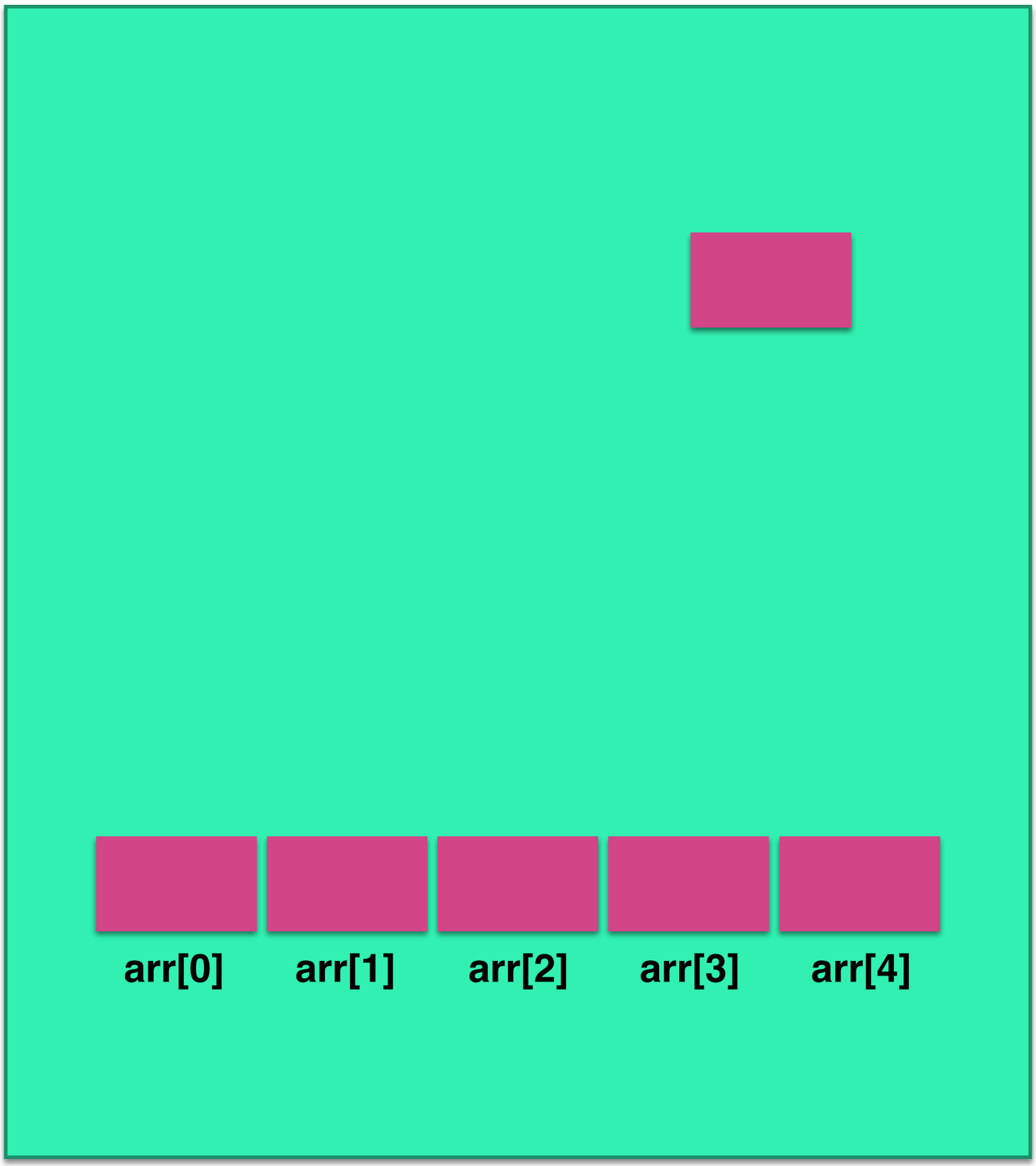
huiw@suda.edu.cn

前面的学习中，我们处理的都是**标量**。

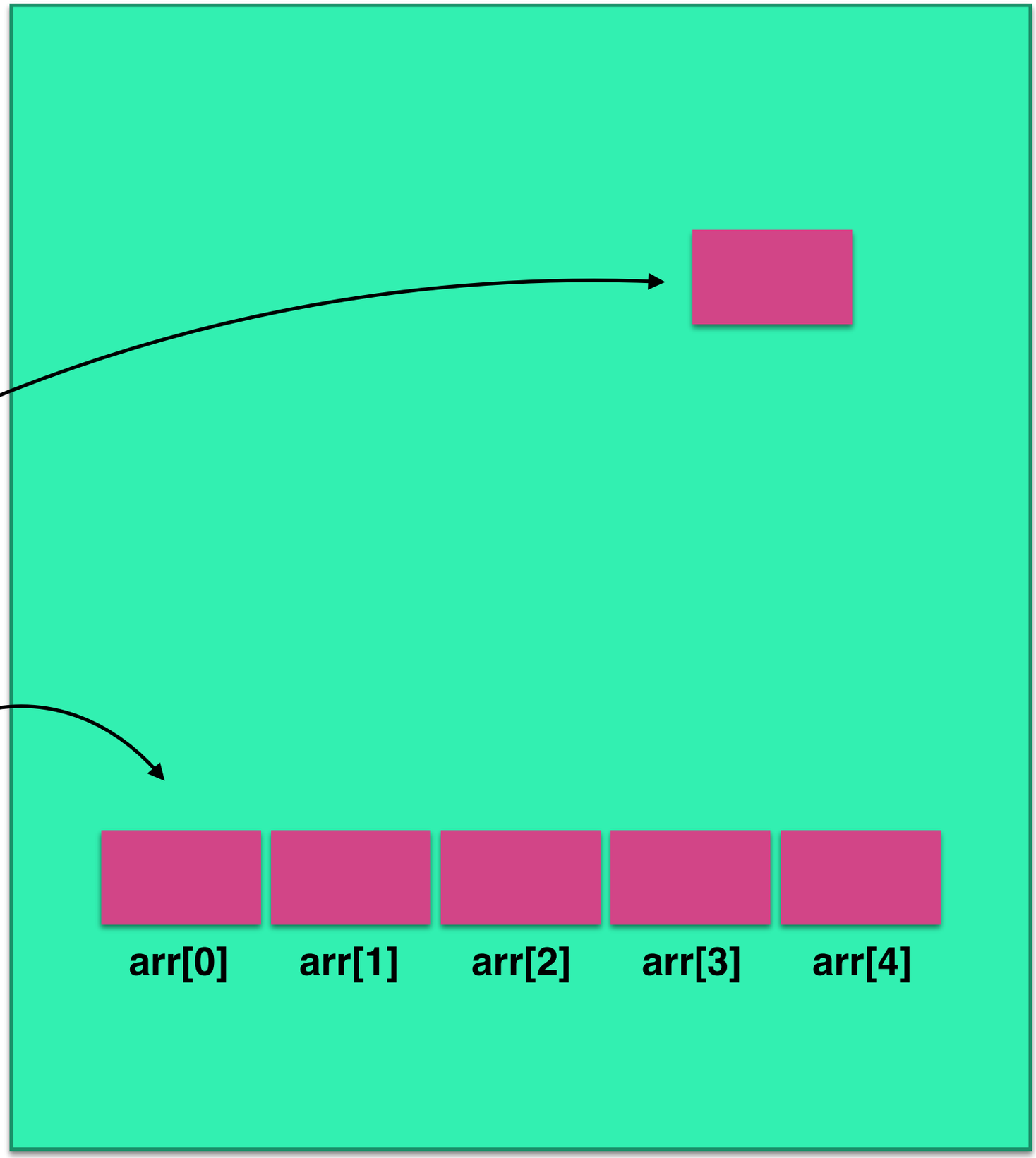
C语言为我们提供了相同类型数据的聚集能力。

```c
int a;
```

```c
int arr[5];
```



```c
int a;
```




```c
int arr[5];
```

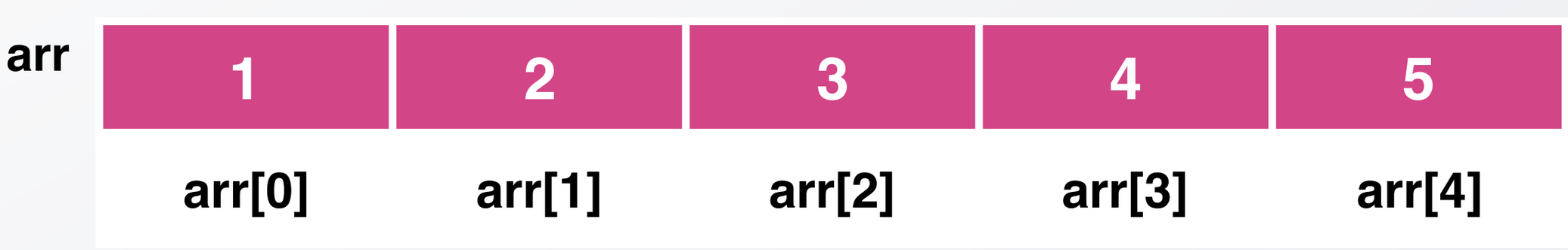
数组元素的赋值

```
arr[1] = 100;
```

数组元素在存储中是连续存放的。

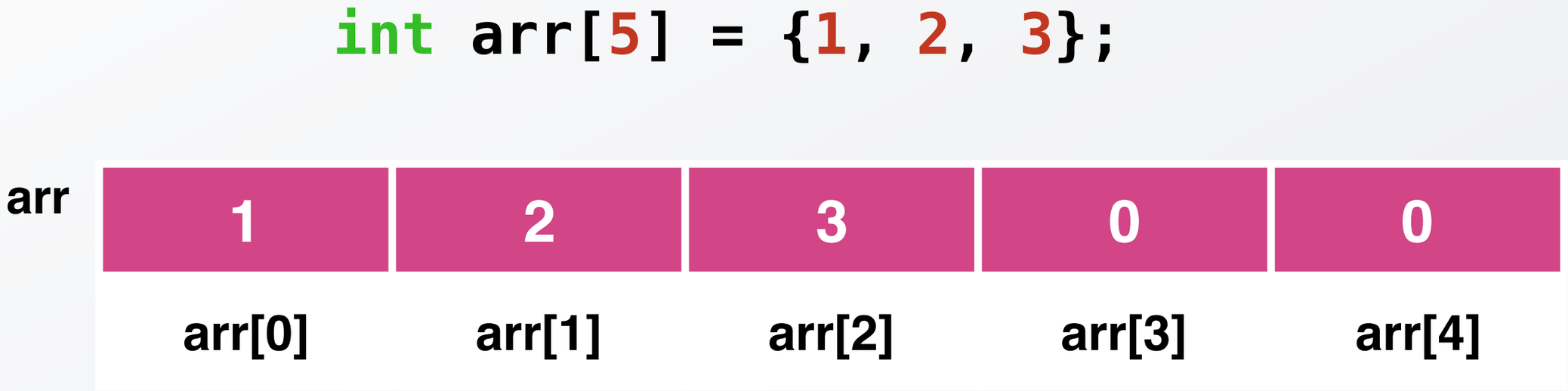
数组元素的初始化

```
int arr[5] = {1, 2, 3, 4, 5};
```



数组元素的初始化

```
int arr[5] = {1, 2, 3};
```



```c
int a[10] = {1, 2, 3, 4, 5, 6};
/* initial value of a is {1, 2, 3, 4, 5, 6, 0, 0, 0, 0} */
```

```c
int a[10] = {0};
/* initial value of a is {0, 0, 0, 0, 0, 0, 0, 0, 0, 0} */
```

```c
int a[] = {1, 2, 3, 4, 5, 6};
/* initial value of a is {1, 2, 3, 4, 5, 6} */
```

```c
int a[15] = {[14] = 48, [9] = 7, [2] = 29};

/* initial value of a is
 *
 * {0, 0, 29, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 48}.
 *
 */
```

```
int a[] = {[14] = 48, [9] = 7, [2] = 29};
```

数组a有多少个元素呢?

通过程序来初始化数组元素...

```
const unsigned int array_size = 5;

int arr[array_size];
int i;
```

```
for (i = 0; i < array_size; i++)
  arr[i] = 0;
```

```
int a = 60;
int b;
b = a;
```

a 的右值是?

```
int arr[5] = {5,6};
```

- arr 的右值是?

```int a = 60;```

a 的右值是?

变量a中存放的 整数值 60.

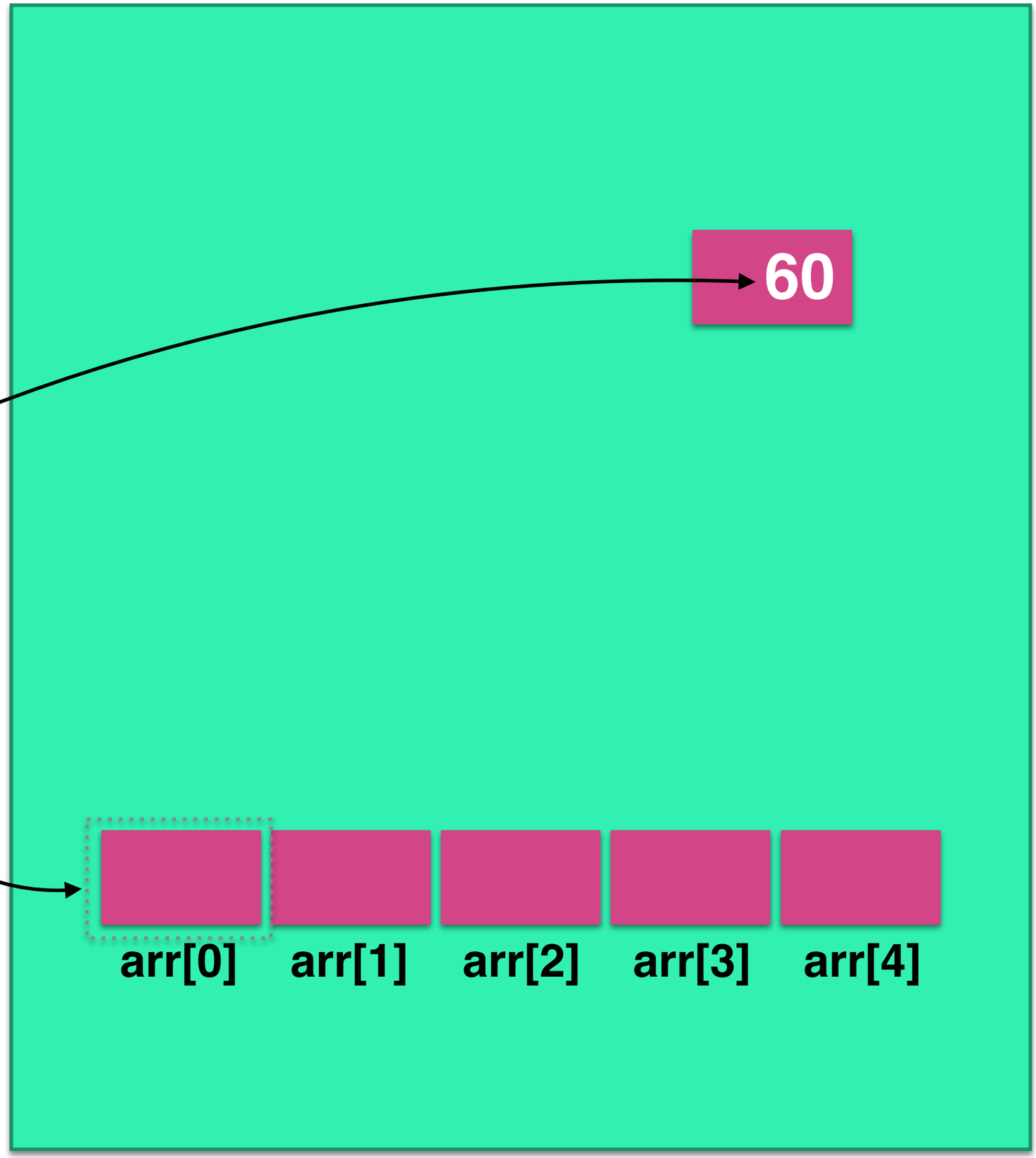
```int arr[5] = {5,6};```

- arr 的右值是?

arr第一个元素在内存中的 位置 .

数组的右值，为什么如此诡异?

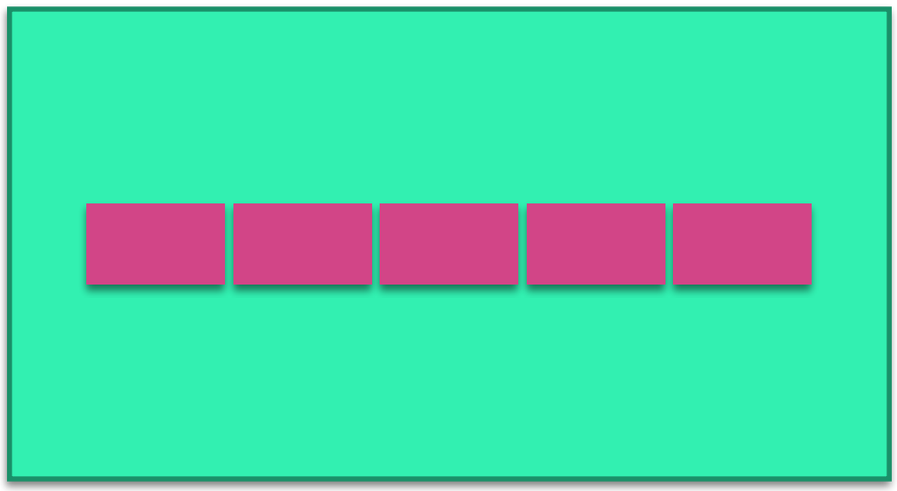
```
int a;
```



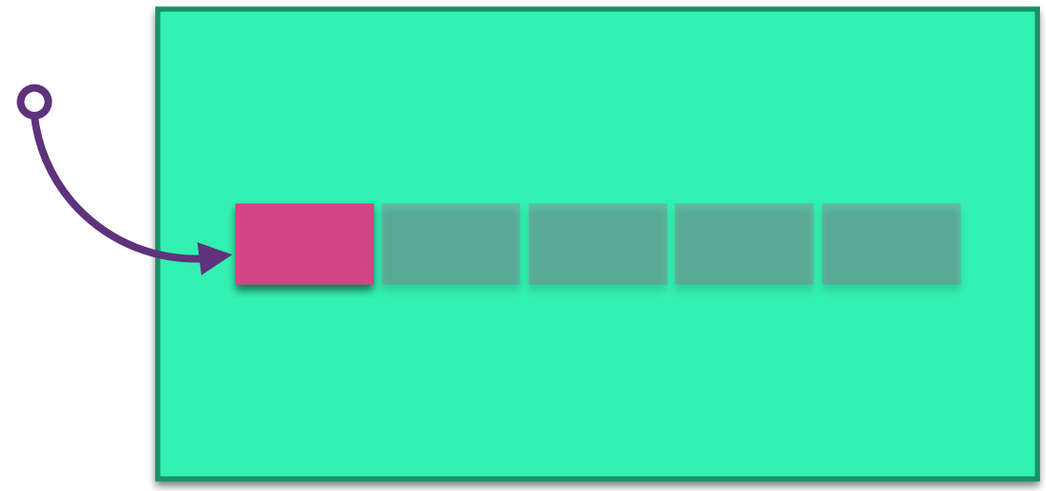

```
int arr[5];
```

数组的右值

数组的左值



数组的右值



这说明数组是不能直接赋值的。

```c
int array_1[5] = {0};

int array_2[5] = {1,2,3,4,5};

array_1  =  array_2;
```

操作数组仅有右值显然是不够的。

1. 数组的位置
2. 数组第一个元素的存储空间（size）
3. 数组总共有多少个元素

1 数组的位置

数组的右值 arr

2 数组第一个元素的大小（size）

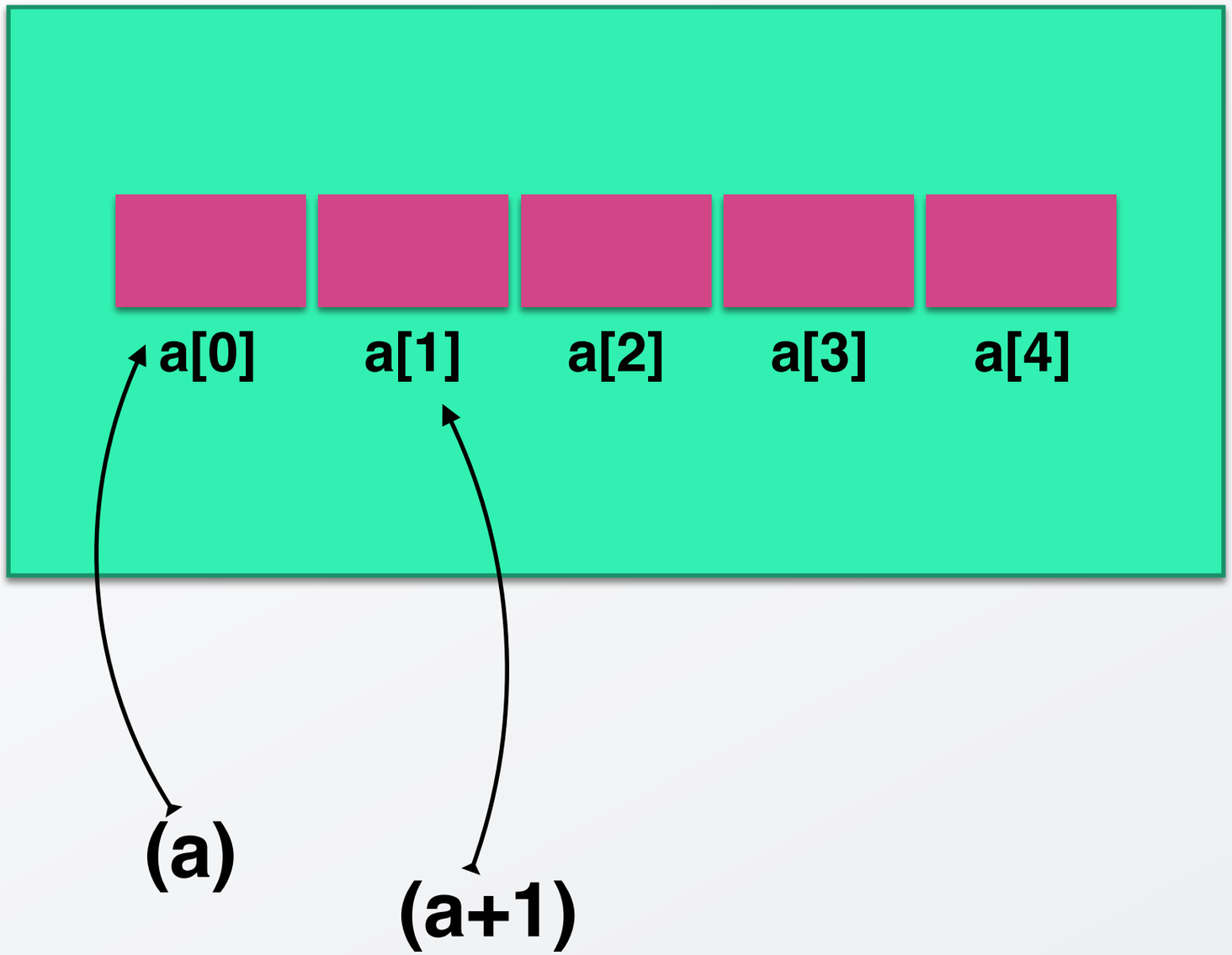
sizeof (arr[0])

3 数组总共有多少个元素

sizeof(arr) / sizeof (arr[0])

换句话说...对于数组这种数据类型

其右值仅仅记录了数组第一个元素的位置（指针）。



(a)是我们将来要学习的指针类型。

加括号是想特别强调一下， a的右值是指针。

判断一个正整数是否有相同数字

```
bool
has_digits_repeated(unsigned int n) {

  int digits_bucket[10] = {0};

  for (int i = n; i > 0; i = i / 10) {
    int last_digit = i % 10;

    digits_bucket[last_digit]++;
  }

  // check digit buckets, finding a non-zero element.
  for (int i = 0; i < 10; i++) {
    if (digits_bucket[i] > 1 ).
      return true;
  }

  return false;
}
```

```
bool
has_digits_repeated(unsigned int n) {

  int digits_bucket[10] = {0};

  for (; n > 0; n /= 10) {
    int last_digit = n % 10;

    digits_bucket[last_digit]++;
  }

  // check digit buckets, finding a non-zero element.
  for (int i = 0; i < 10; i++) {
    if (digits_bucket[i] > 1 ).
      return true;
  }

  return false;
}
```

编程计算e的值

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots + \frac{1}{n!}$$

注意： 这次用循环语句实现哦。

怎么做?

数组元素整体逆转

```
void
swap(int arr[], int i, int j) {
  int temp = arr[i];
  arr[i] = arr[j];
  arr[j] = temp;
}
```

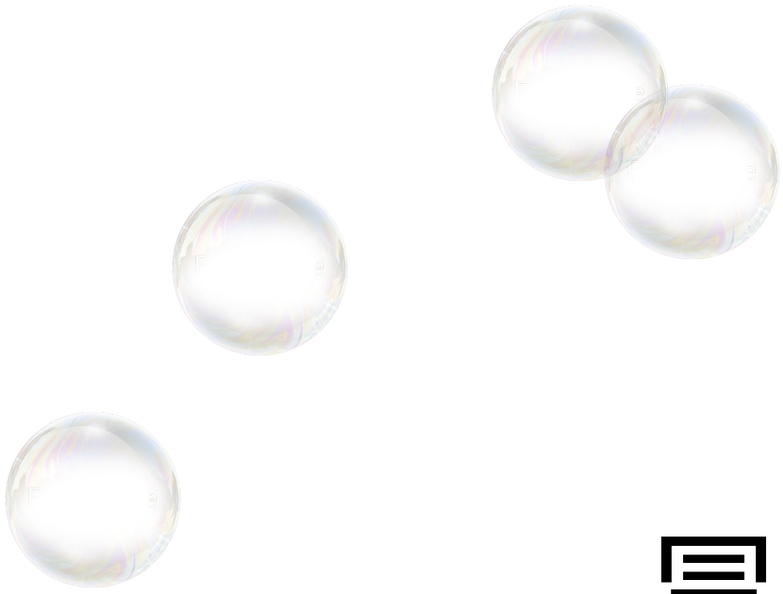
```
void
reverse(int a[], unsigned int size) {

  unsigned int i, j;

  for (i=0, j=size-1; i<j; i++, j--)
    swap(a, i, j);

}
```

冒泡排序

冒泡排序

冒泡排序

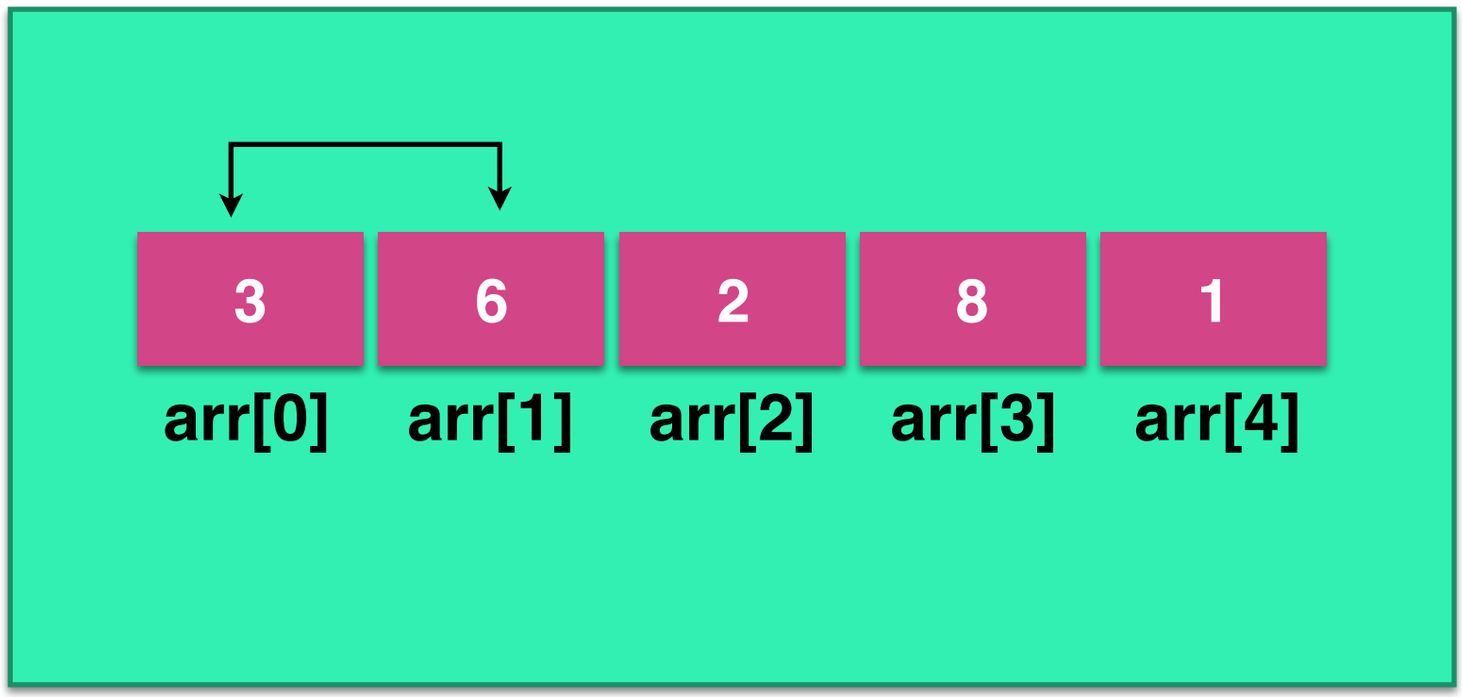
冒泡排序

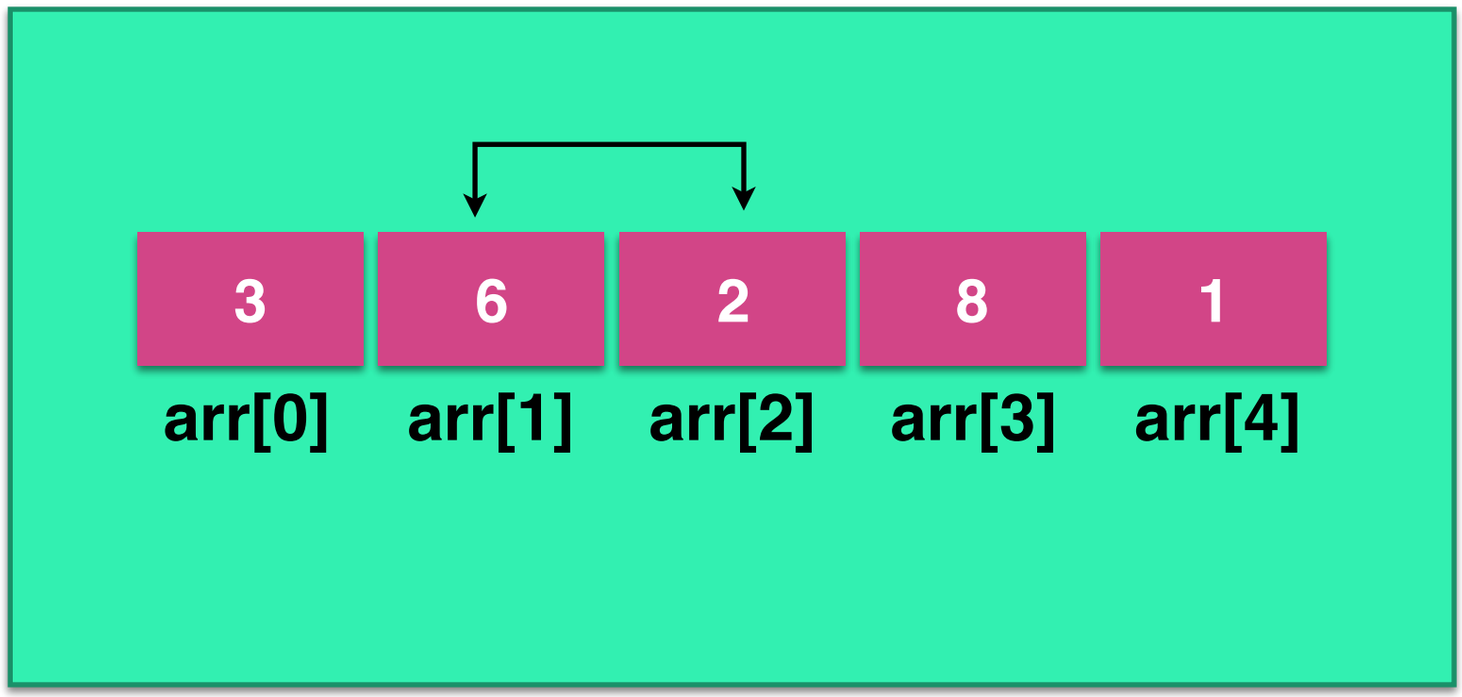


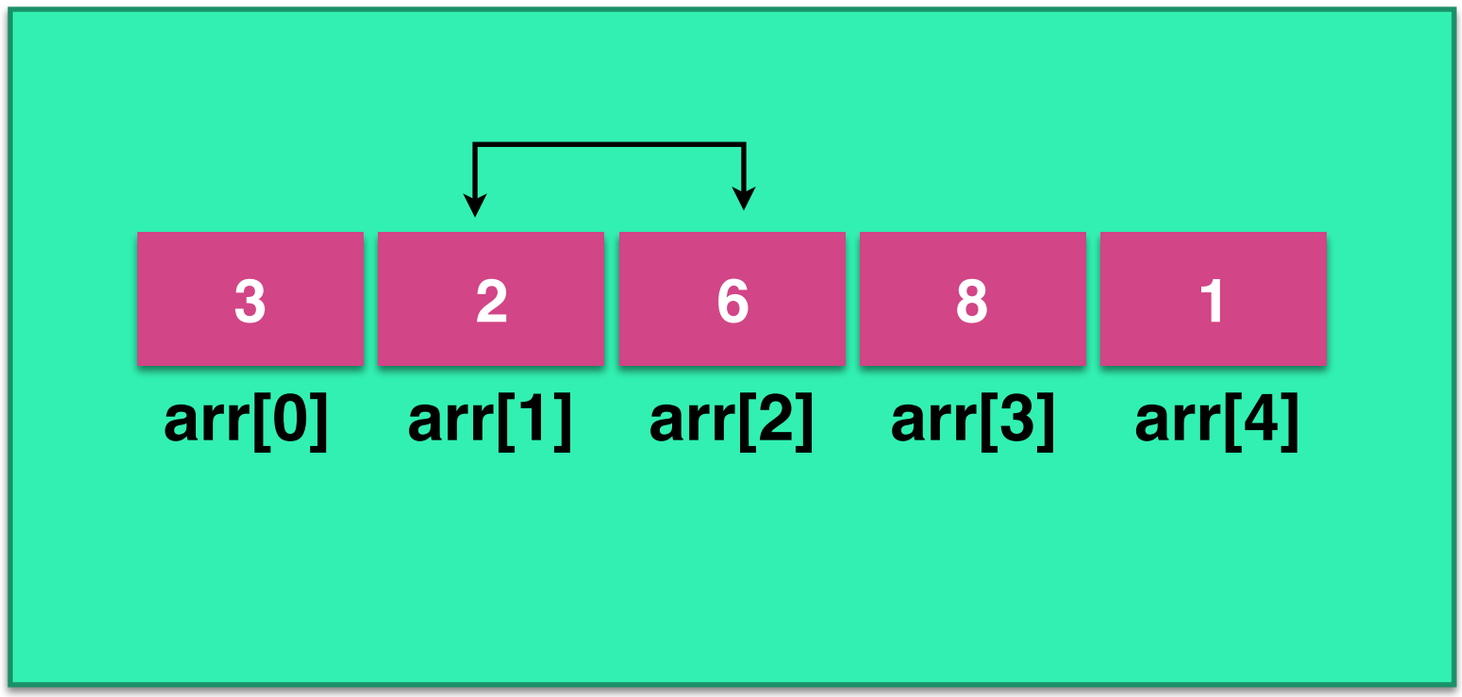
冒泡排序

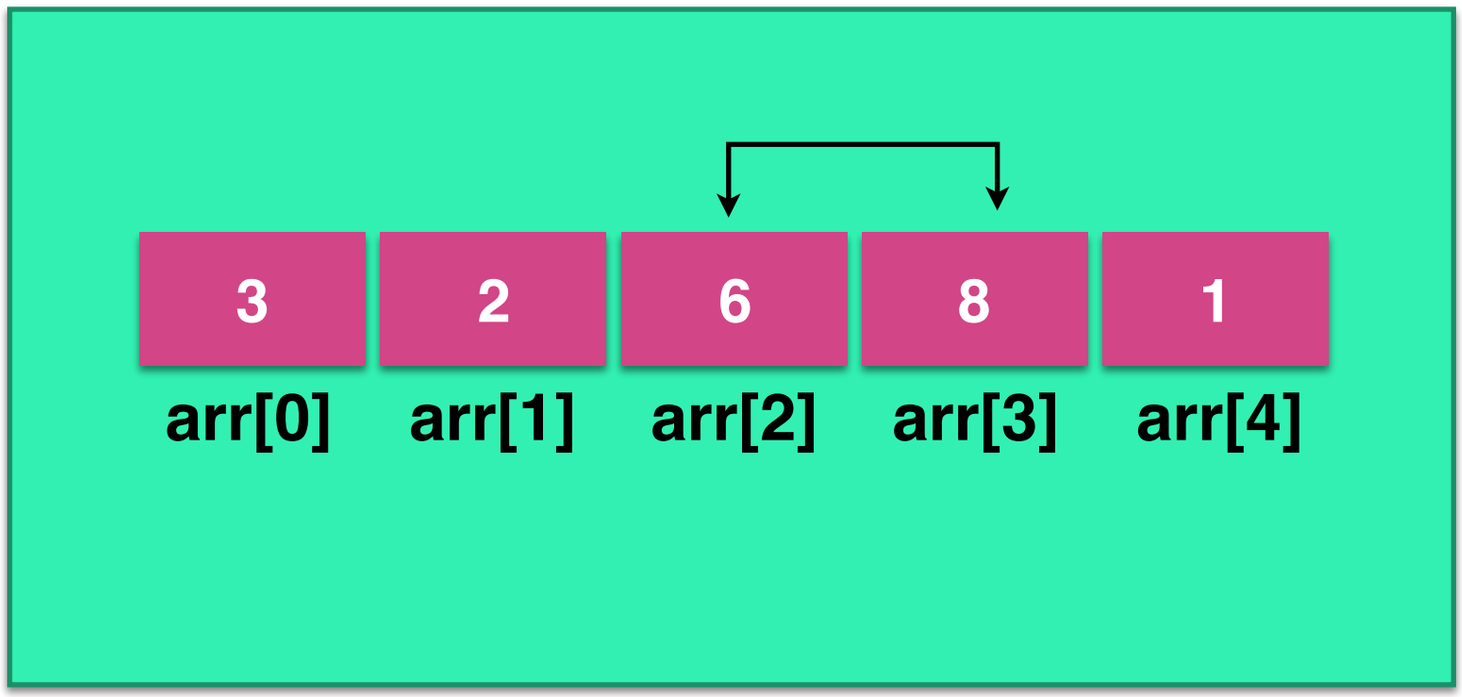
```
void
swap(int arr[], int i, int j) {
  int temp = arr[i];
  arr[i] = arr[j];
  arr[j] = temp;
}
```

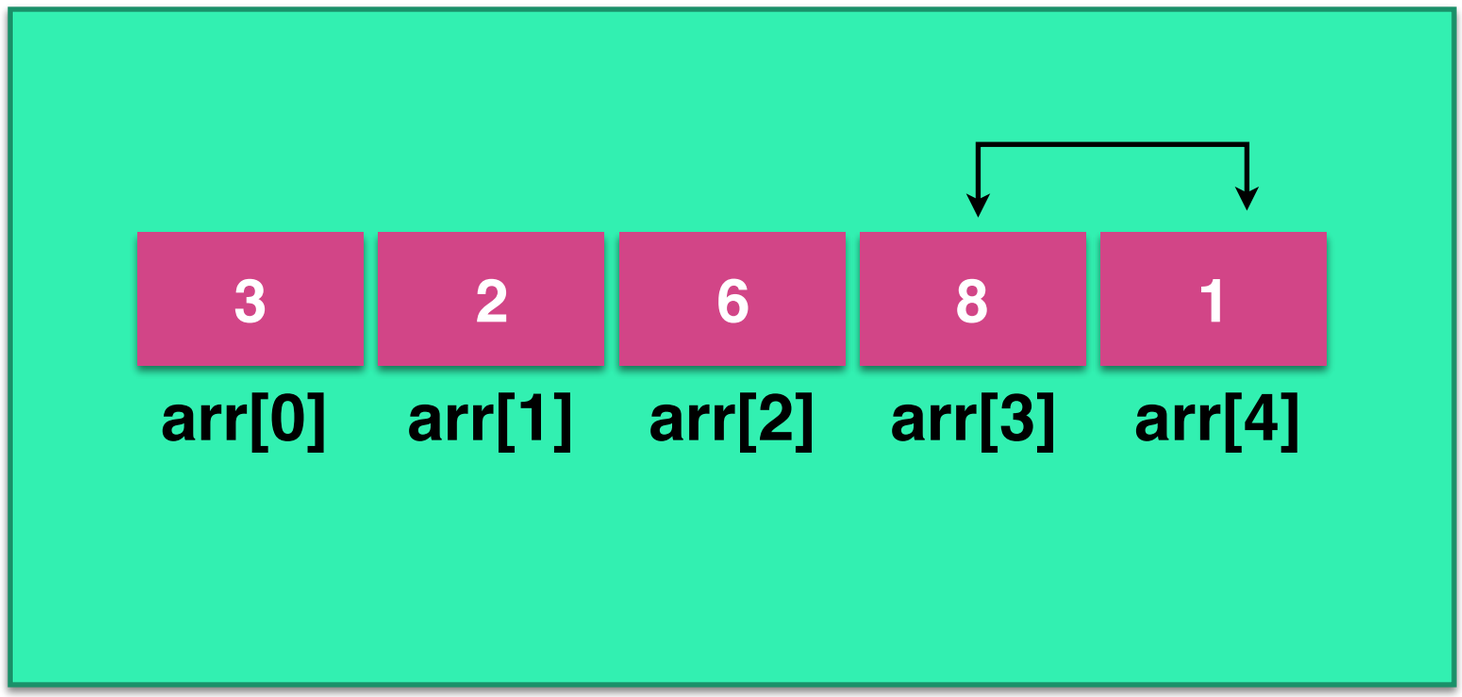
```
bool
move_max_rightmost(int arr[], int n) {

  bool arr_changed = false;
  int j;


  for (j = 0; j < n - 1; j++)
    if (arr[j] > arr[j + 1]) {


      swap(arr, j, j+1);
      arr_changed = true;
    }

  return arr_changed;
}
```

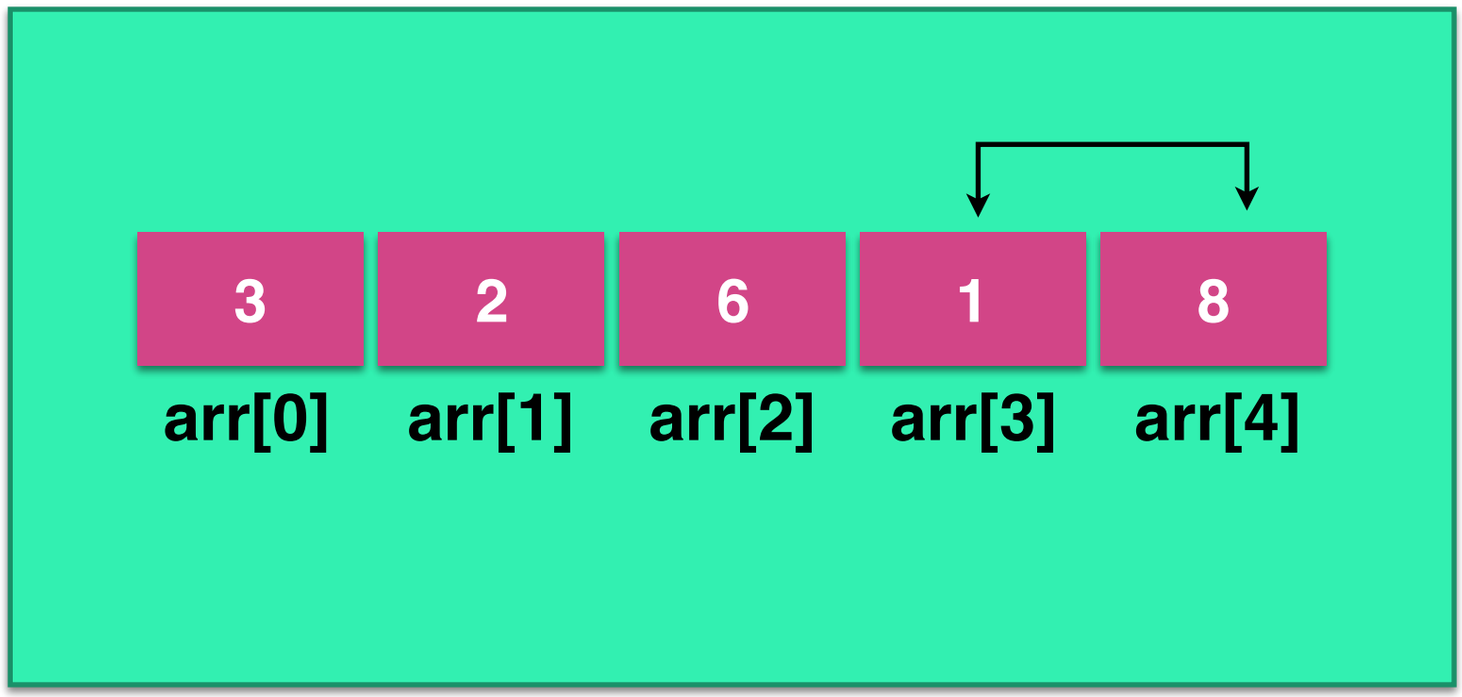

3
6
2
8
1
arr[0]
arr[1]
arr[2]
arr[3]
arr[4]


3
6
2
8
1
arr[0]
arr[1]
arr[2]
arr[3]
arr[4]


3
2
6
8
1
arr[0]
arr[1]
arr[2]
arr[3]
arr[4]


3
2
6
8
1
arr[0]
arr[1]
arr[2]
arr[3]
arr[4]


3
2
6
8
1
arr[0]
arr[1]
arr[2]
arr[3]
arr[4]


3
2
6
1
8
arr[0]
arr[1]
arr[2]
arr[3]
arr[4]

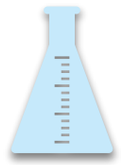
```
void bubble_sort(int arr[], int n) {
   int i;

   for (i = 0; i < n; i++) {
      bool array_remain_unchanged =
                        !move_max_rightmost(arr, n-i);
      if (array_remain_unchanged) return;
   }
}
```

```c
int main(int argc, char *argv[]) {

  const unsigned int array_size = 10;
  int arr[array_size];
  int i;

  for (i = 0; i < array_size; i++)
    arr[i] = array_size - i - 1;

  bubble_sort(arr, array_size);

  for (i = 0; i < array_size; i++)
    putchar('0'+arr[i]);

  return 0;
}
```

向量的点乘计算

```
int
dot_mul(int arr_a[], int arr_b[], unsigned int size ) {

   int sum = 0;
   for (unsigned int i = 0; i<size; i++) {
      sum += (arr_a[i] * arr_b[i]);
   }

   return sum;
}
```

```c
int main(int argc, char *argv[]) {

  const unsigned int array_size = 10;
  int arr[array_size];
  int i;

  for (i = 0; i < array_size; i++)
    arr[i] = i;

  assert(dot_mul(arr, arr, array_size) == 9*(9+1)*(2*9+1)/6);
  return 0;
}
```

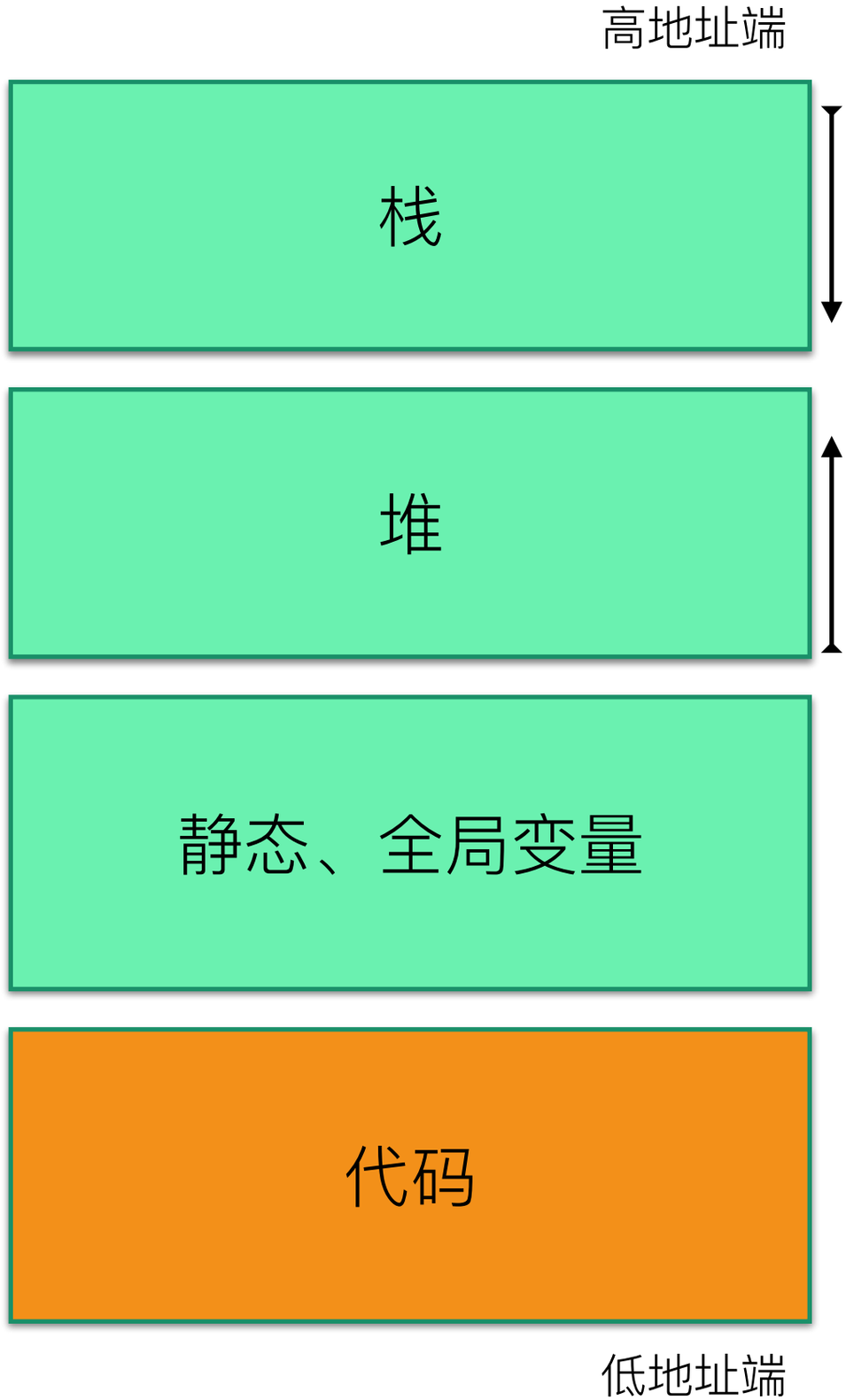


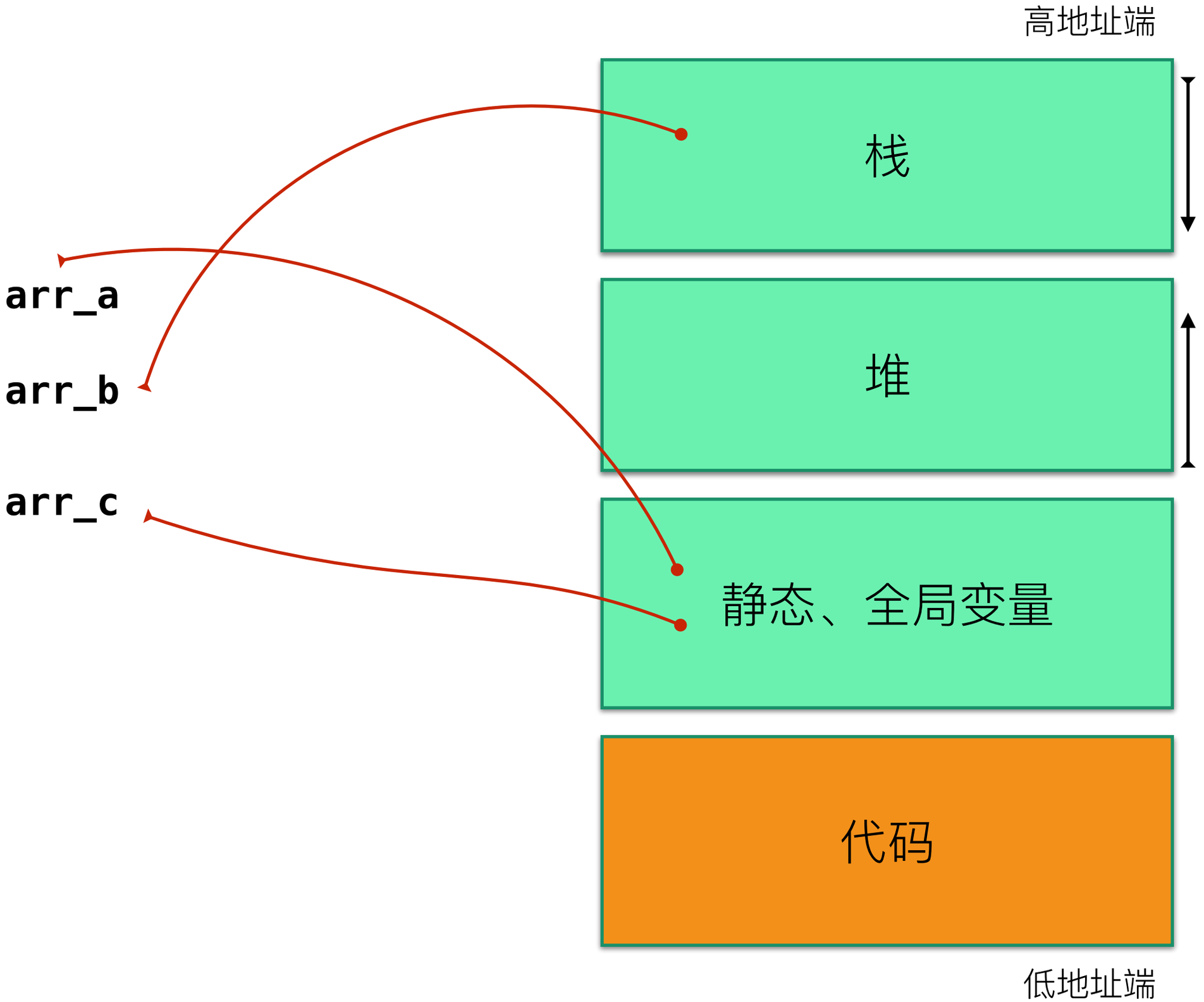
C - L A B
实验室

```c
int arr_a[10] = {0, 1, 12, 3, [8]=5, [7]=1};

int main(int argc, char* argv[]) {
    int arr_b[10] = {1, 2, 3, [8]=5, [4]=1};

    static int arr_c[10] = {2, 3, [8]=5, [5]=1};

    return 0;
}
```

```
arr_a

arr_b

arr_c
```

这些数组对象存放在哪里呢?




高地址端
栈
堆
静态、全局变量
代码
低地址端
arr_a
arr_b
arr_c

思考一下

数组参数可以作为C函数的输入，
试问C函数能不能使用**局部数组类型变量**作为输出?

答案是不能。

原因是就算函数能返回一个完整的数组，
程序也无法使用这个数组。

因为数组不能被赋值。

再思考一下

如果脑洞大开，允许一个整数数组作为函数的返回类型，函数声明该怎么写?

```
int arr[5];
```

提示数组的声明

```
int func_proto_impossible(int arr[], int i, int j) [5];
```

于是...故事的发生只能是这样的...

1. 先准备好数组对象
2. 传入函数，处理数组元素
3. 结果存放在原来准备好的数组中

二分查找（递归实现）

```
bool binary_search(int a[], int key, uint32_t lower, uint32_t upper) {
  uint32_t mid;
  if (lower > upper) return false;.
  mid = (lower + upper) / 2;

  if (key < a[mid]).
    return binary_search(a, key, lower, mid - 1);

  if (key > a[mid]).
    return binary_search(a, key, mid + 1, upper);

  return true;
}
```

stdbool.h

stdint.h

```
bool binary_search(int a[], int key, uint32_t lower, uint32_t upper) {
  uint32_t mid;
  if (lower > upper) return false;.
  mid = (lower + upper) / 2;

  if (key < a[mid]).
    return binary_search(a, key, lower, mid - 1);

  if (key > a[mid]).
    return binary_search(a, key, mid + 1, upper);

  return true;
}
```

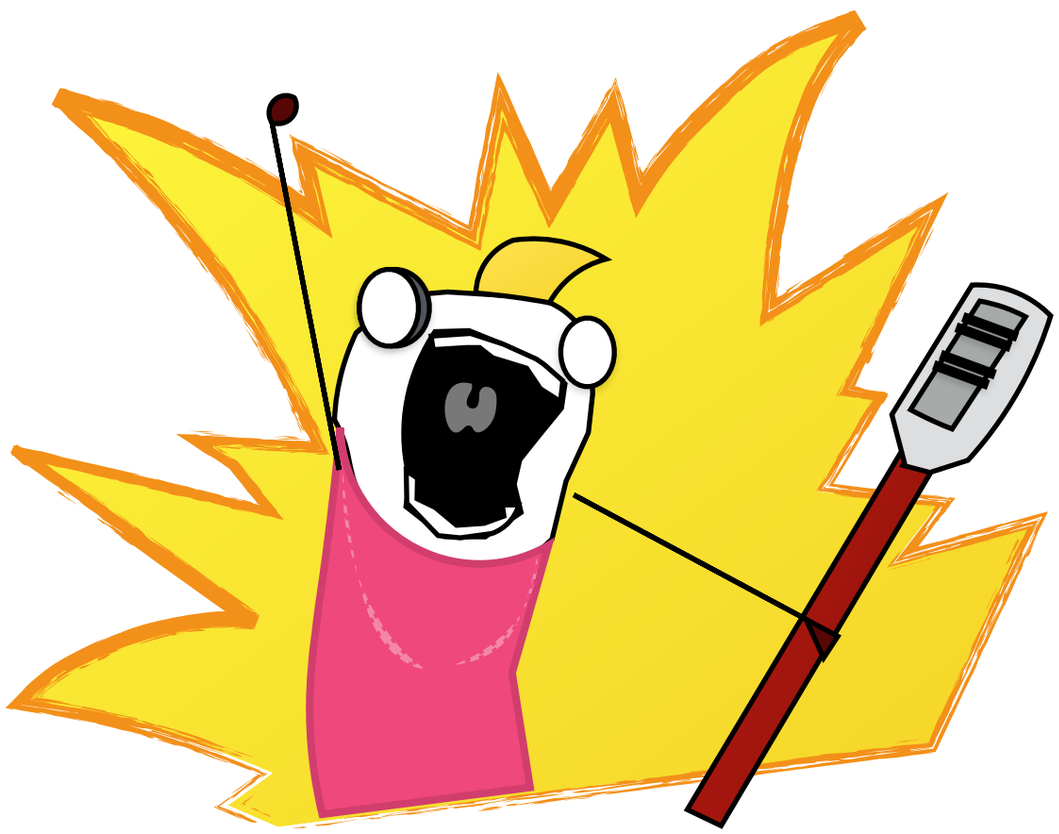
```c
#include <assert.h>
#include <stdbool.h>
#include <stdint.h>
```

```c
int database[] = {0, 1, 2, 34, 41, 50, 69, 77, 84, 99};

int main (int argc, char* argv[]) {
  assert(binary_search(database,  1, 0, sizeof(database)/sizeof(int)-1));
  assert(binary_search(database, 50, 0, sizeof(database)/sizeof(int)-1));
  assert(binary_search(database, 99, 0, sizeof(database)/sizeof(int)-1));
  assert(binary_search(database,  0, 0, sizeof(database)/sizeof(int)-1));
  assert(binary_search(database, 84, 0, sizeof(database)/sizeof(int)-1));
  assert(binary_search(database,  7, 0, sizeof(database)/sizeof(int)-1));

  return 0;
}
```

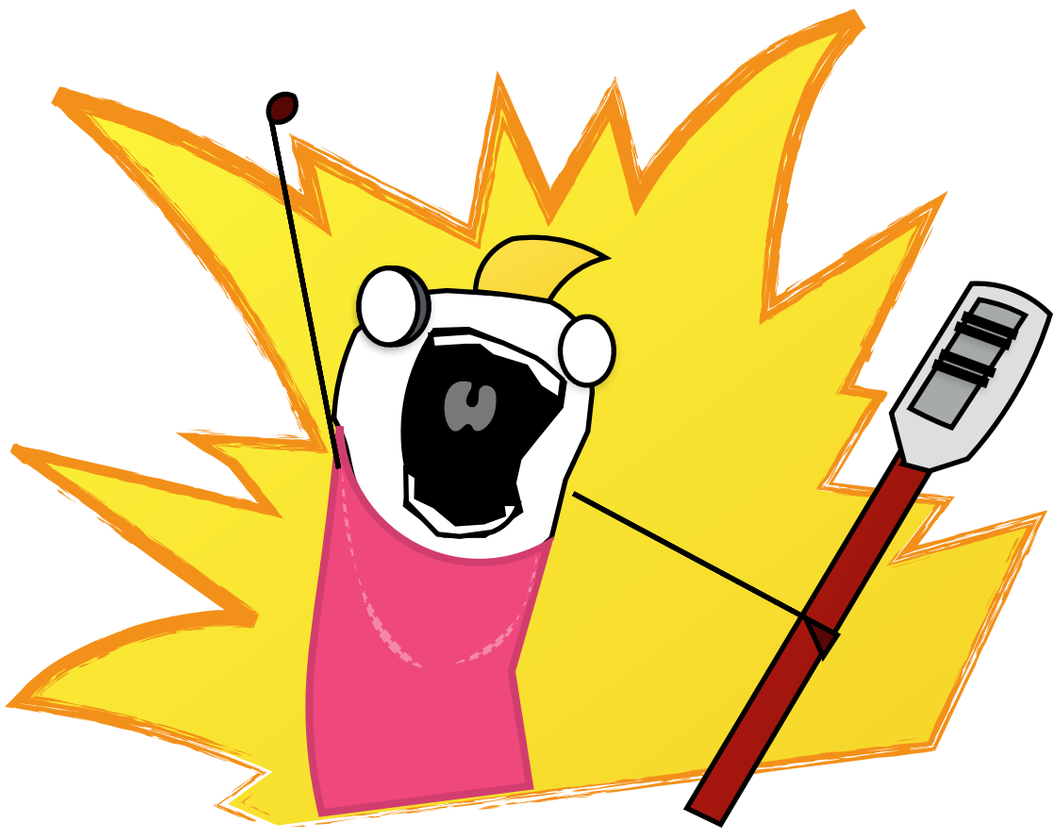
还有4周就考试了 …

必须打一针鸡血！！！

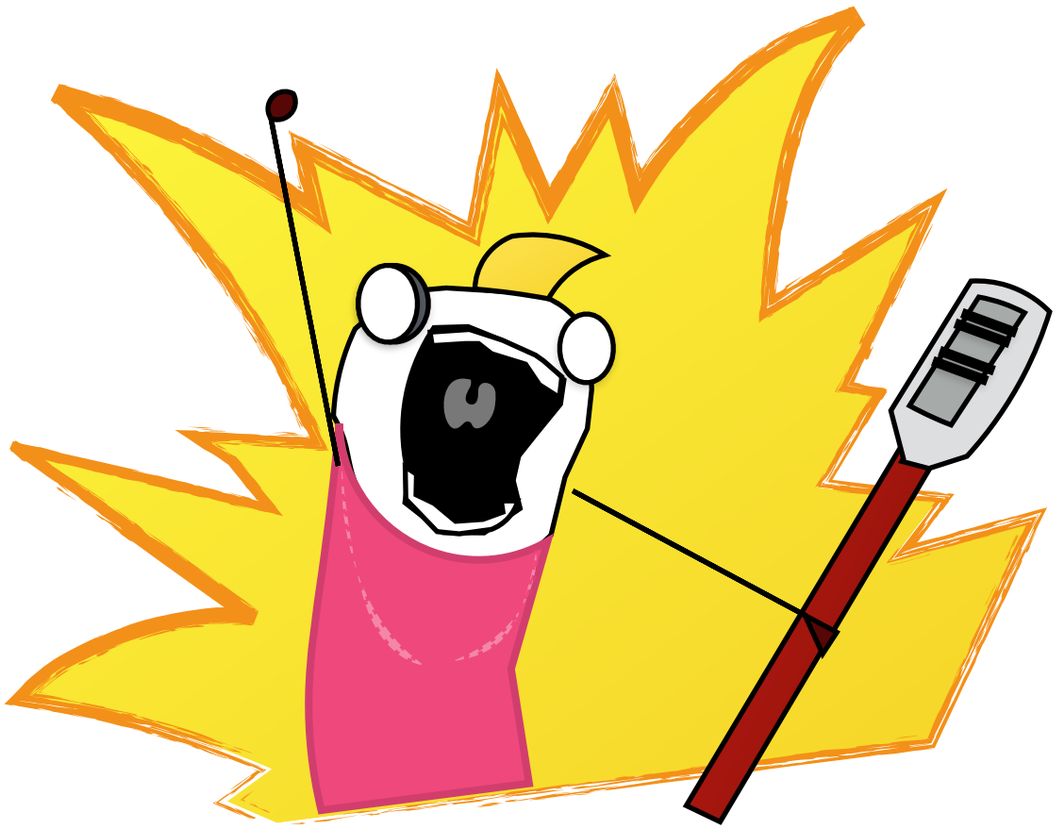
我 们 是 谁 ！

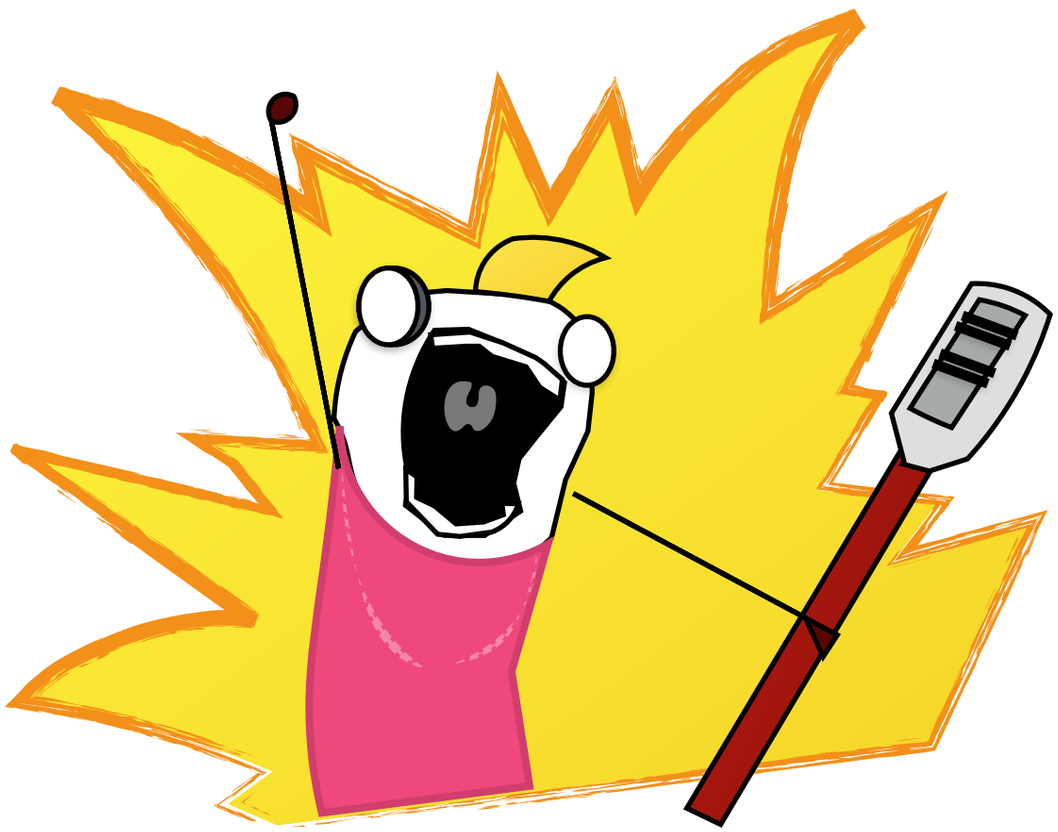


19计算机



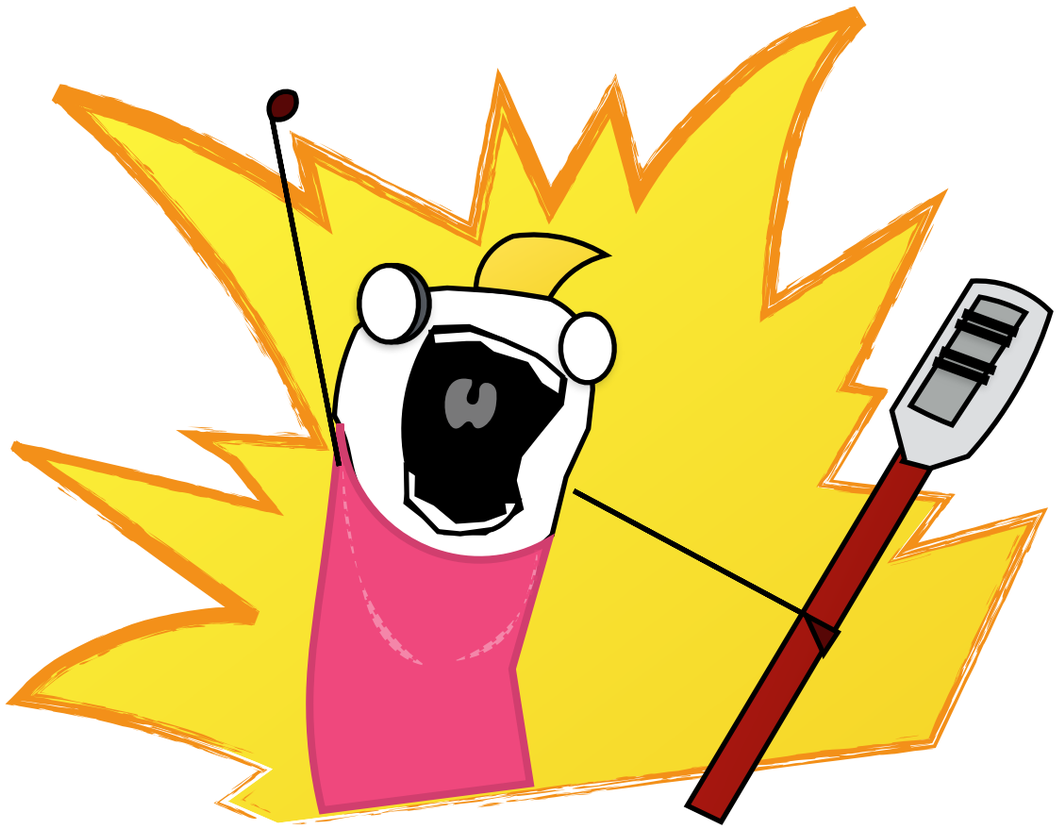
我们的目标是



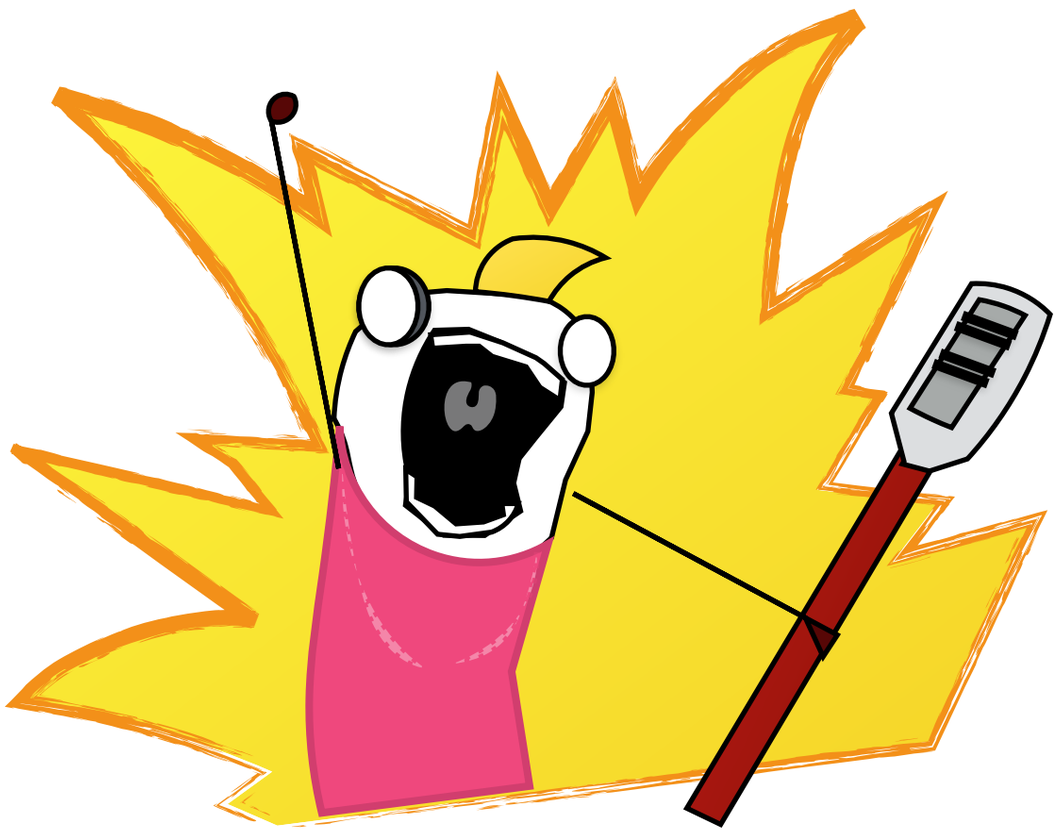


搞钱！

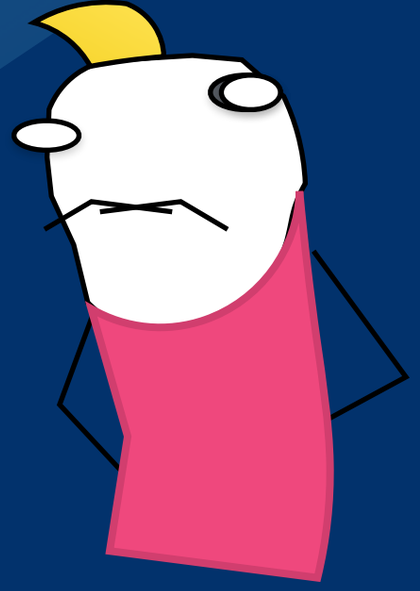
我 们 怎 么 办 ？

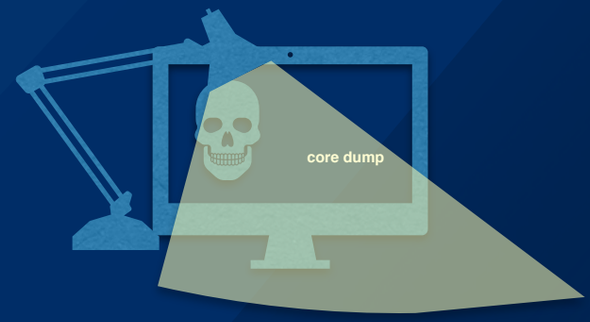


熬夜写程序！



秋 寒 窗 前 丁
风 扰 雪 花 屏
影 暗 虫 无 迹
孤 栈 斜 月 明





数组的数组

```
int i
```

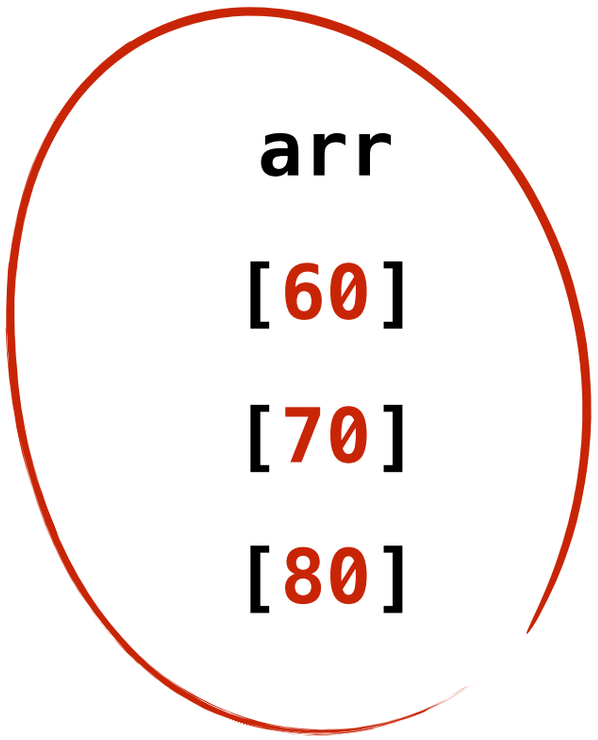
i

int

```
int  arr[100];
```

arr

```
int [100];
```




```
int [100];
```

```
arr
[60]
[70]
[80]

int [100];
```

arr

[60]

[70]

int[80][100];

arr

[60]

int [70][80][100];

arr

```
int  [60][70][80][100];
```

```
int   arr[60][70][80][100];
```

```
arr
[60]
[70]
[80]

int [100];
```

arr

[60]

[70]

int[80][100];

arr

[60]

int [70][80][100];

arr

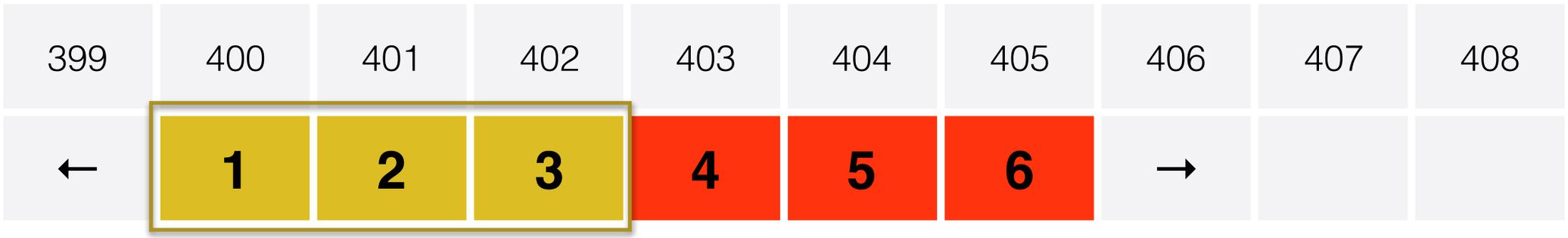
```
int  [60][70][80][100];
```

```
// array of integers
int a[3];

// array of array of integers
int b[2][3] = {{1, 2, 3}, {4, 5, 6}};
```

399	400	401	402	403	404	405	406	407	408
←	1	2	3	4	5	6	→		

```
// array of array of integers
int b[2][3] = {{1, 2, 3}, {4, 5, 6}};
```

b[0]




```
// array of array of integers
int b[2][3] = {{1, 2, 3}, {4, 5, 6}};
```

```
b[0]
```

399	400	401	402	403	404	405	406	407	408
←	1	2	3	4	5	6	→		

```
// array of array of integers
int b[2][3] = {{1, 2, 3}, {4, 5, 6}};
```

b[1]

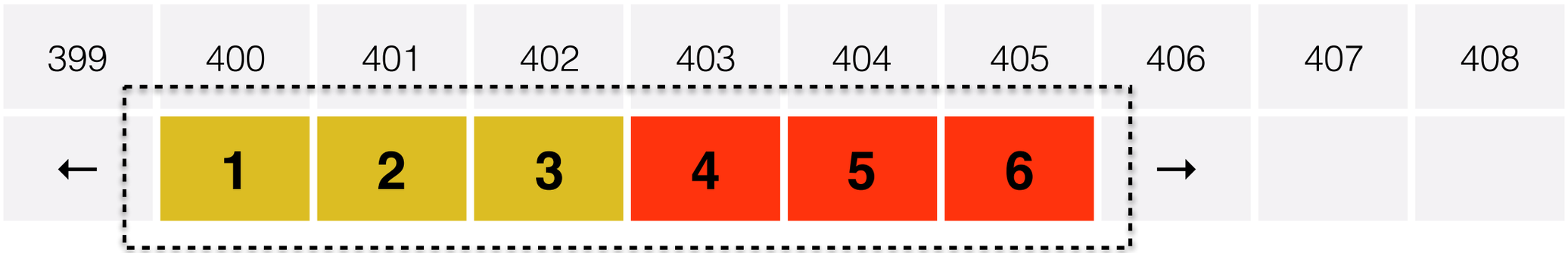

```
// array of array of integers
int b[2][3] = {{1, 2, 3}, {4, 5, 6}};
```

```
b[1]
```

399	400	401	402	403	404	405	406	407	408
←	**1**	**2**	**3**	**4**	**5**	**6**	→		

```
// array of array of integers
int b[2][3] = {{1, 2, 3}, {4, 5, 6}};
```

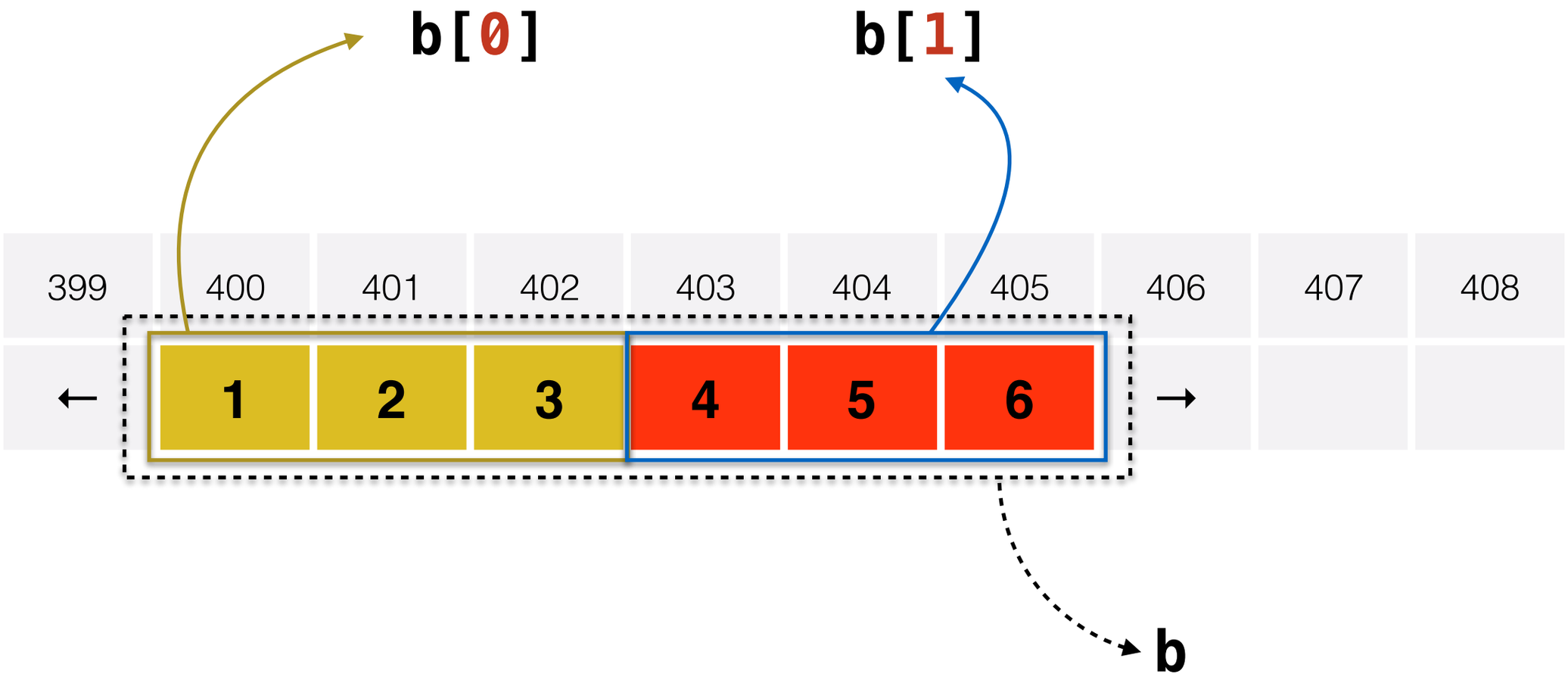
b



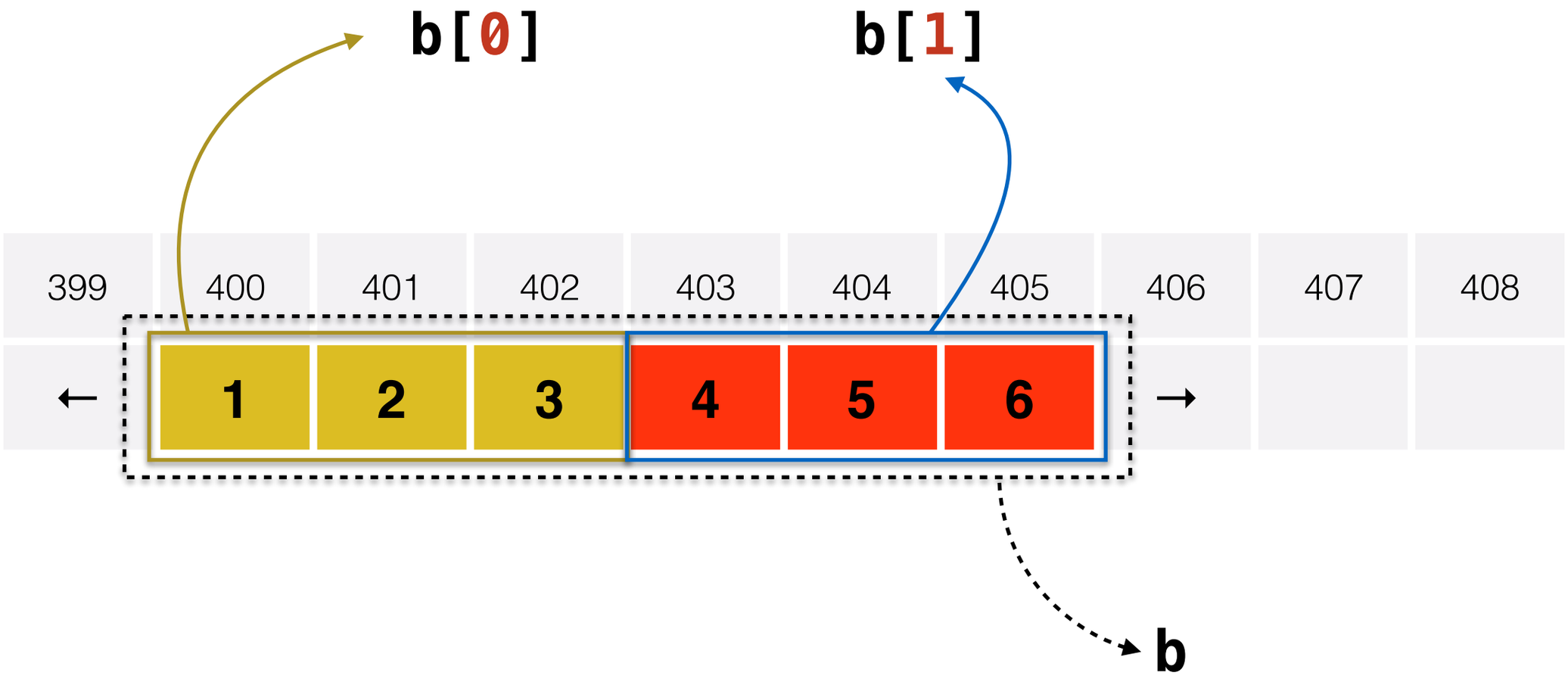

```
// array of array of integers
int b[2][3] = {{1, 2, 3}, {4, 5, 6}};
```

b

```
// array of array of integers
int b[2][3] = {{1, 2, 3}, {4, 5, 6}};
```

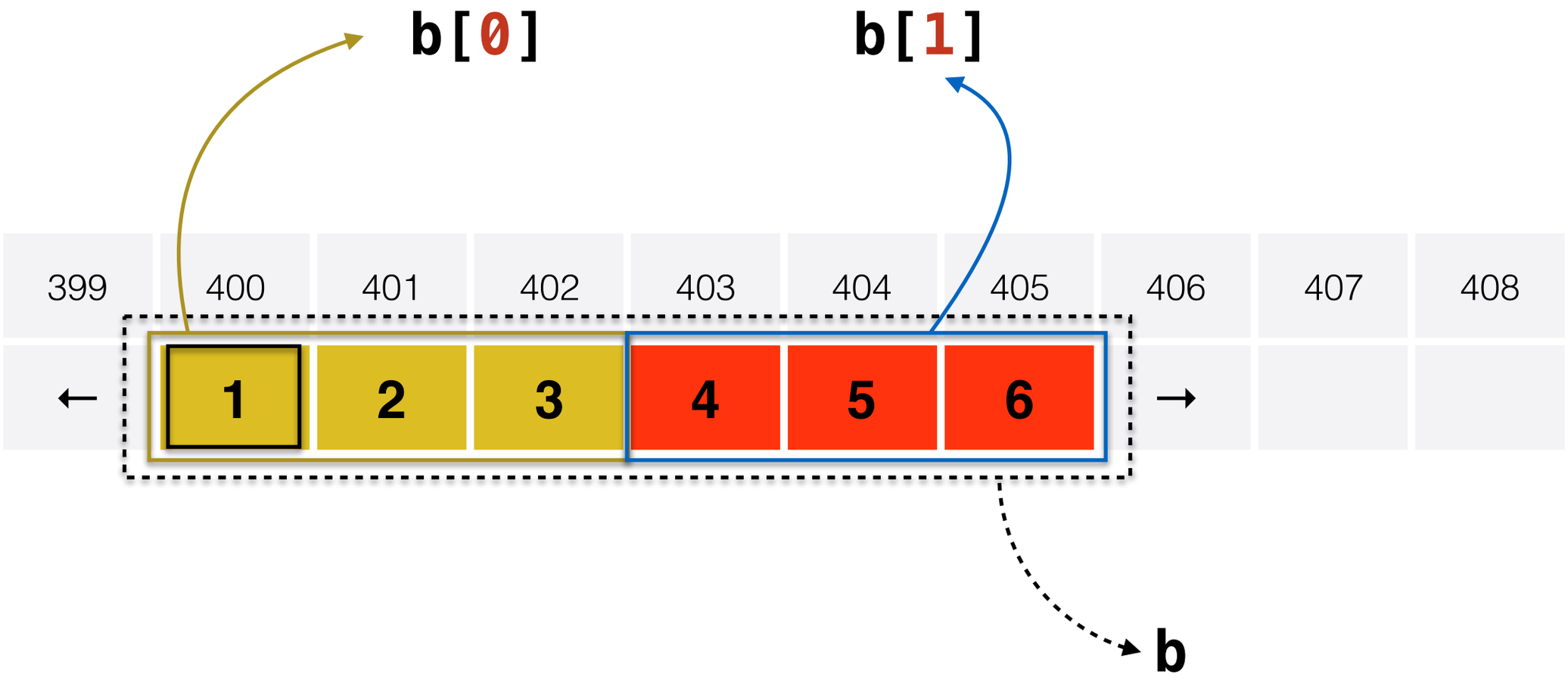


```
// array of array of integers
int b[2][3] = {{1, 2, 3}, {4, 5, 6}};
```



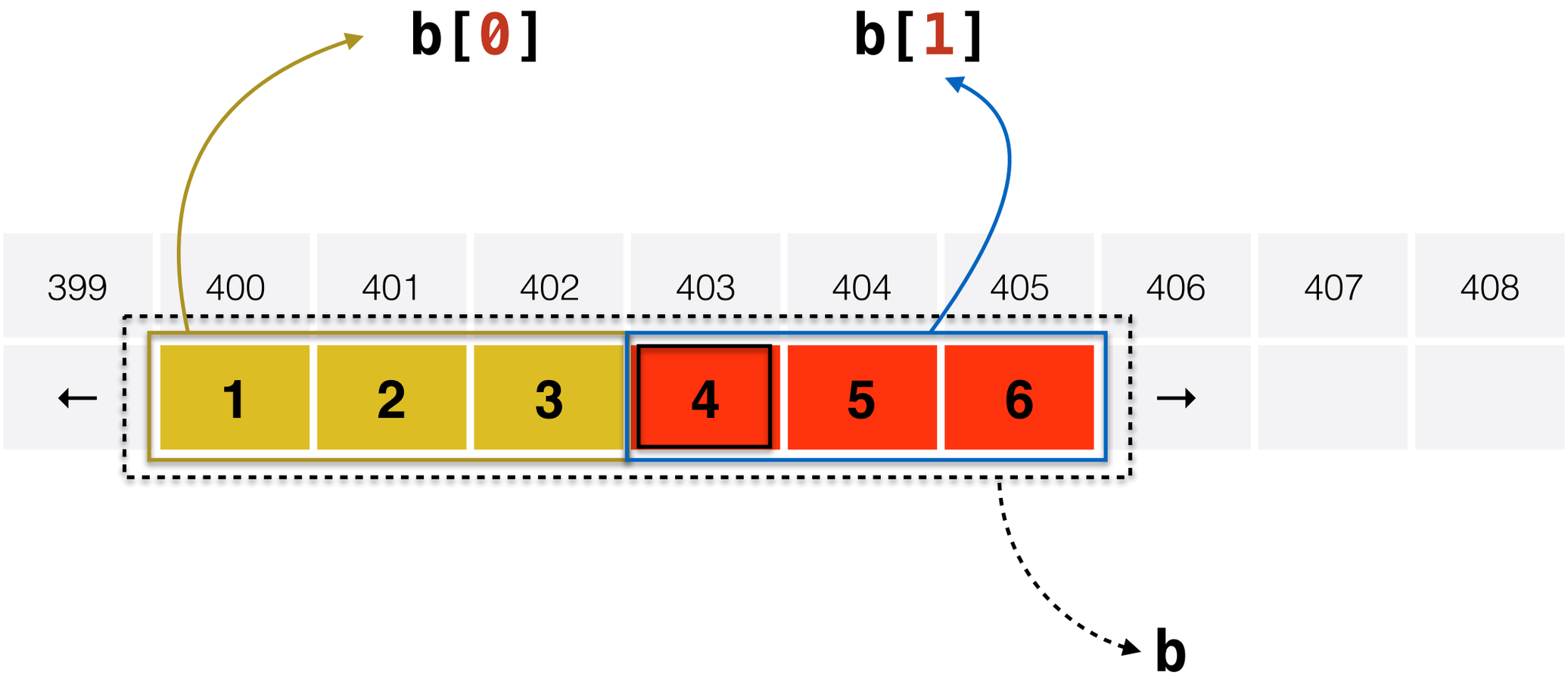
b是个二维整数数组，其右值是 **400**

```
// array of array of integers
int b[2][3] = {{1, 2, 3}, {4, 5, 6}};
```



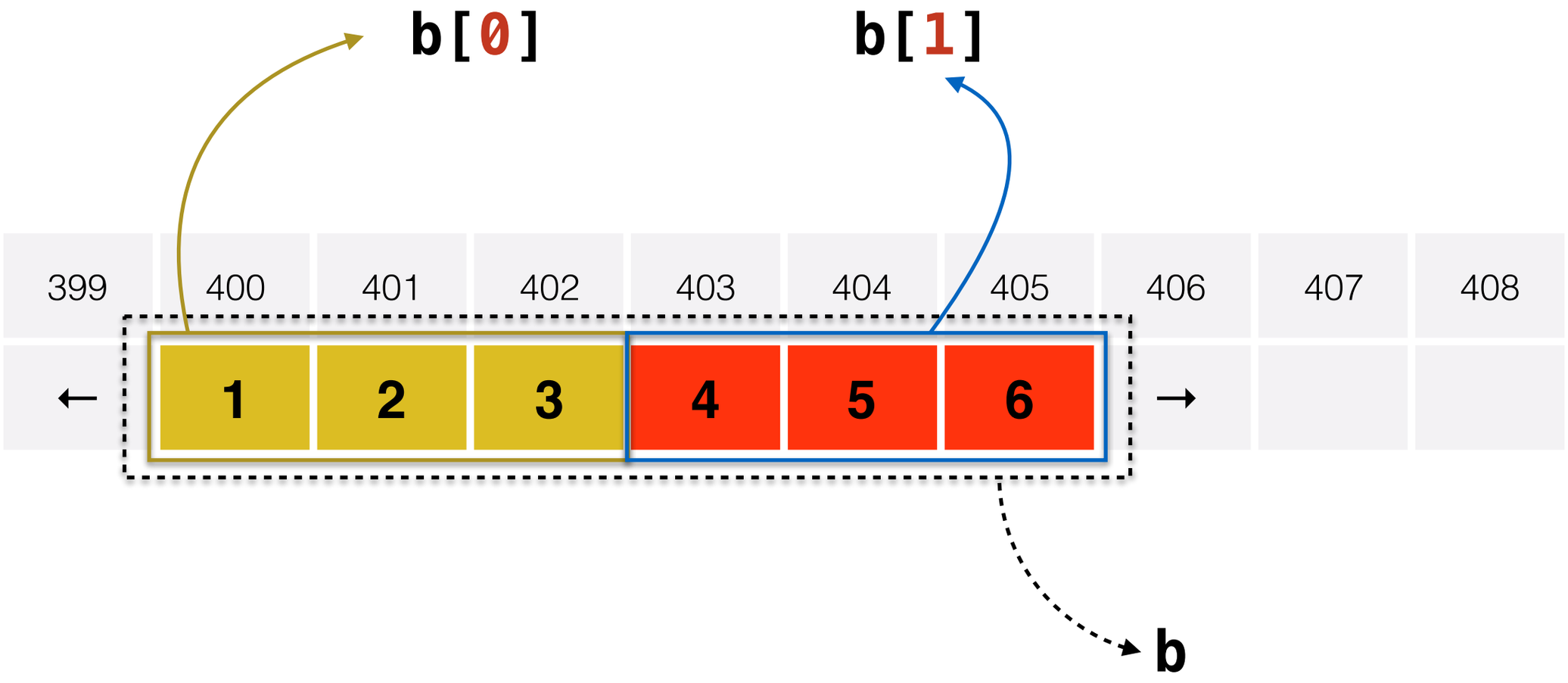
b[0]是个一维整数数组，其右值是 **400**

```
// array of array of integers
int b[2][3] = {{1, 2, 3}, {4, 5, 6}};
```



b[1]是个一维整数数组，其右值是 **403**

```
// array of array of integers
int b[2][3] = {{1, 2, 3}, {4, 5, 6}};
```



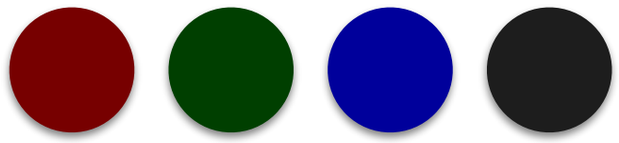
b[0][0]是个整数变量，其右值是 1 ， 其左值为400.

```
assert(sizeof(b) == sizeof(int) * 6);
assert(sizeof(b[0]) == sizeof(int) * 3);
assert(sizeof(b[1]) == sizeof(int) * 3);
assert(sizeof(b[2][3]) == sizeof(int));
```

声明定义的数据对象，在存储器中都有位置，
所以它们都是有左值的。

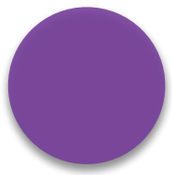
数组对象的左值，**不允许使用**。

有了数组，我们描述数据对象的能力迅速增强...

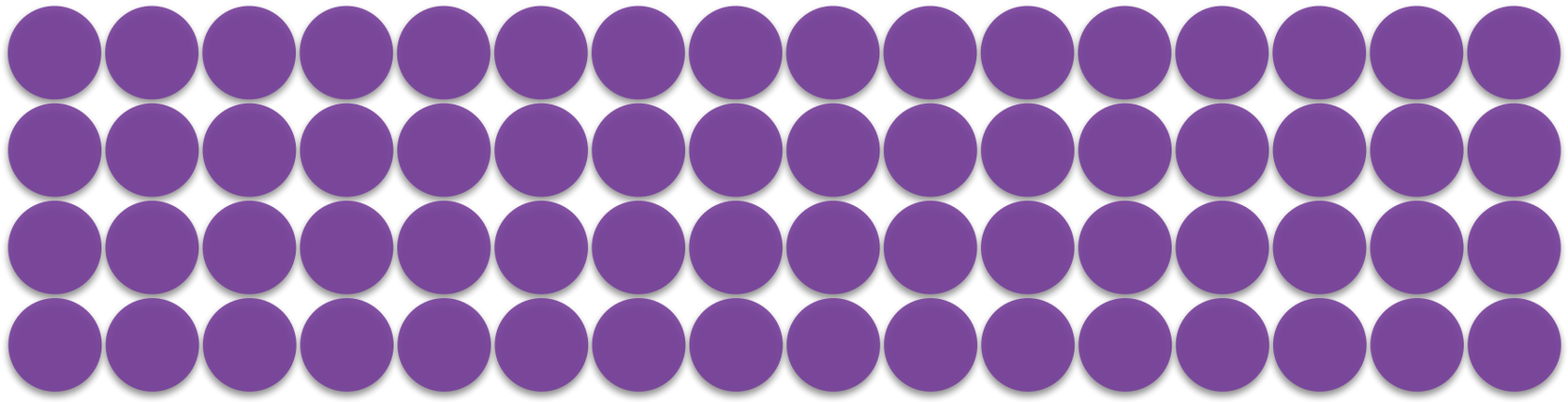


```
typedef unsigned char color_t[4];
```

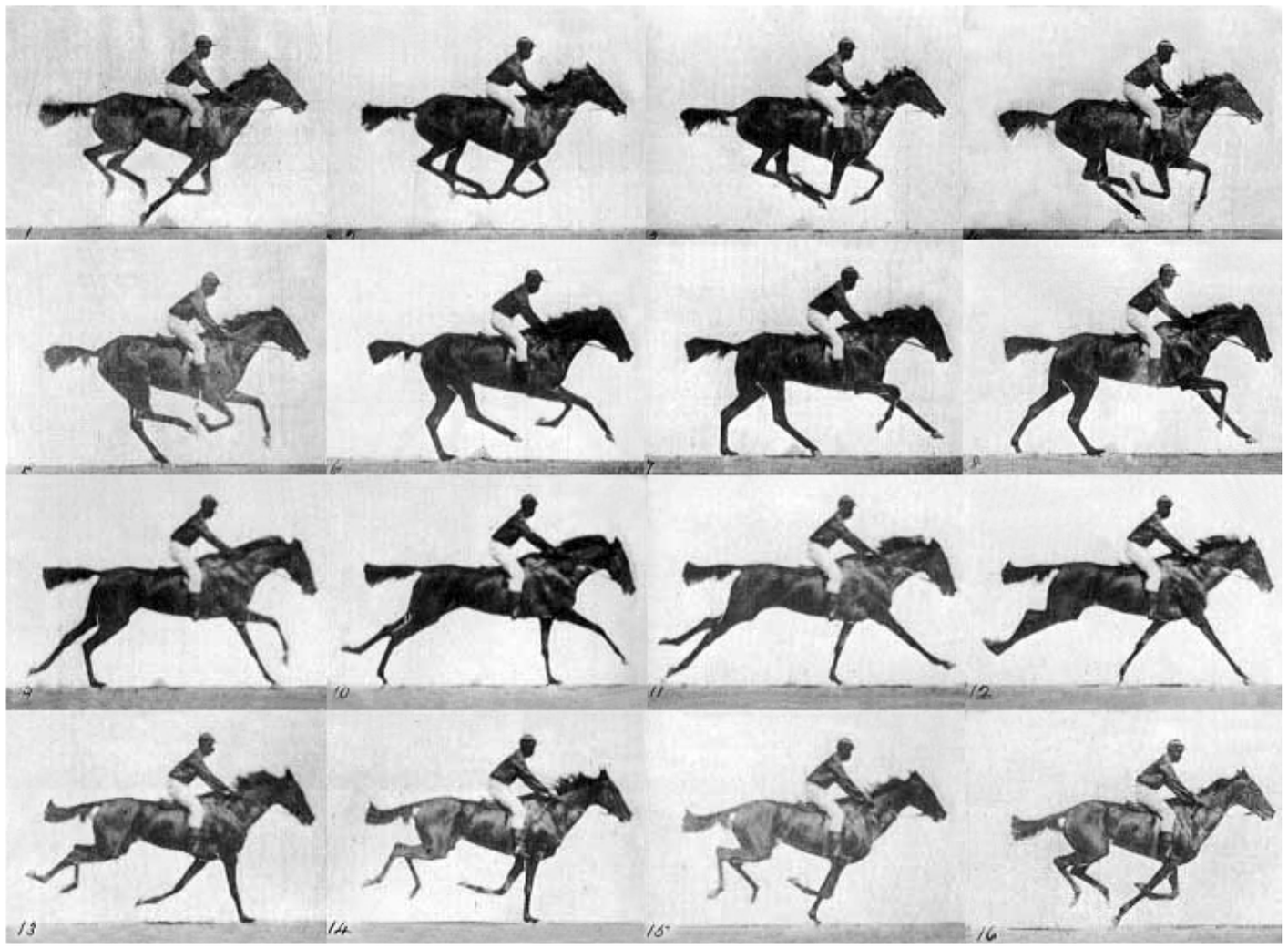
```
typedef color_t pixel_t;
```



```
#define WIDTH  400
#define HEIGHT 300
#define MOVIE_SZ_FRAME 9999

typedef pixel_t frame_t[HEIGHT][WIDTH];
```



```
typedef frame_t movie_t[MOVIE_SZ_FRAME];

extern movie_t horse_riding;
```



```
void
play_movie(movie_t movie) {
    for (int i = 0; i < MOVIE_SZ_FRAME; i++) {
        copy_frame_to_display(movie[i]);
    }
}
```

计算矩阵的和

matadd.c

```c
#define ROW_ELEMENTS 3
#define COL_ELEMENTS 2

typedef int mat_t[COL_ELEMENTS][ROW_ELEMENTS];

// array of array of integers
mat_t m = {{1, 2, 3}, {4, 5, 6}};
mat_t n = {{10, 20, 30}, {40, 50, 60}};
mat_t r;

void mat_add(mat_t m, mat_t n, mat_t r) {
  for (int column = 0; column < COL_ELEMENTS; column++)
    for (int row = 0; row < ROW_ELEMENTS; row++) {

      r[column][row] = m[column][row] + n[column][row];
    }
}
```

matadd.c

```
void print_mat(mat_t r) {
  for (int column = 0; column < COL_ELEMENTS; column++) {
    for (int row = 0; row < ROW_ELEMENTS; row++) {

      printf("%4d\t", r[column][row]);
    }
    putchar('\n');
  }
}

int main(int argc, char *argv[]) {.
  mat_add(m, n, r);
  print_mat(r);
  return 0;.
}
```

课堂练习

二阶矩阵求逆

计算3阶矩阵的逆阵

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 1 & 0 & 6 \end{bmatrix}$$

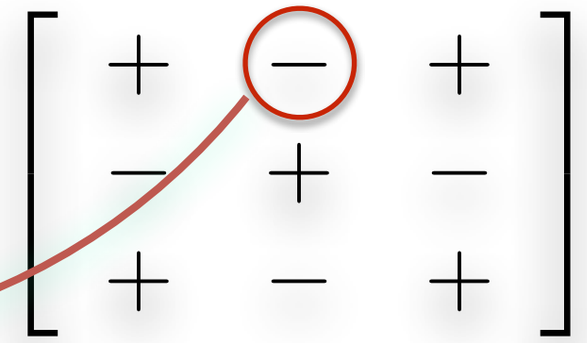
$$A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 1 & 0 & 6 \end{bmatrix}$$

A_{00}的余子式

$$A_{00} = \begin{bmatrix} 4 & 5 \\ 0 & 6 \end{bmatrix}$$

A_{00}的余子式

矩阵A的代数余子式



$$A_{00} = \begin{vmatrix} 4 & 5 \\ 0 & 6 \end{vmatrix} = 24 \qquad A_{01} = -\begin{vmatrix} 0 & 5 \\ 1 & 6 \end{vmatrix} = 5 \qquad A_{02} = \begin{vmatrix} 0 & 4 \\ 1 & 0 \end{vmatrix} = -4$$

$$A_{10} = -\begin{vmatrix} 2 & 3 \\ 0 & 6 \end{vmatrix} = -12 \quad A_{11} = \begin{vmatrix} 1 & 3 \\ 1 & 6 \end{vmatrix} = 3 \qquad A_{12} = -\begin{vmatrix} 1 & 2 \\ 1 & 0 \end{vmatrix} = 2$$

$$A_{20} = \begin{vmatrix} 2 & 3 \\ 4 & 5 \end{vmatrix} = -2 \qquad A_{21} = -\begin{vmatrix} 1 & 3 \\ 0 & 5 \end{vmatrix} = -5 \quad A_{22} = \begin{vmatrix} 1 & 2 \\ 0 & 4 \end{vmatrix} = 4$$

确定代数余子式符号

```
typedef int mat3x3_t[3][3];

bool is_odd(int i) {.
   return (i % 2 == 1) ? true : false;.
}
```

```
bool is_even(int i) {.
   return !is_odd(i);.
}

int sign_cofactor(unsigned int i, unsigned int j) {
   if (is_even(i+j)) return 1;

   return -1;
}
```

matinverse.c

计算代数余子式的值

```c
int det2(int a, int b, int c, int d) {
  return a * d - b * c;
}

int calculate_cofactor(mat3x3_t m, unsigned int i, unsigned int j) {
  assert(i<3 && j<3);
  if (0==i && 0==j) return det2(m[1][1],m[1][2],m[2][1],m[2][2]);
  if (0==i && 1==j) return det2(m[1][0],m[1][2],m[2][0],m[2][2]);
  if (0==i && 2==j) return det2(m[1][0],m[1][1],m[2][0],m[2][1]);

  if (1==i && 0==j) return det2(m[0][1],m[0][2],m[2][1],m[2][2]);
  if (1==i && 1==j) return det2(m[0][0],m[0][2],m[2][0],m[2][2]);
  if (1==i && 2==j) return det2(m[0][0],m[0][1],m[2][0],m[2][1]);

  if (2==i && 0==j) return det2(m[0][1],m[0][2],m[1][1],m[1][2]);
  if (2==i && 1==j) return det2(m[0][0],m[0][2],m[1][0],m[1][2]);

  // if (2==i && 2==j).
  return det2(m[0][0],m[0][1],m[1][0],m[1][1]);
}
```

matinverse.c

计算代数余子式矩阵

```
void
calculate_cofactor_mat(mat3x3_t a, mat3x3_t cofactor_mat)
{
  for (int col=0; col<3; col++)
    for (int row=0; row<3; row++){
      cofactor_mat[col][row] = sign_cofactor(col,row) *.
        calculate_cofactor(a, col, row);
    }

}
```

matinverse.c

矩阵转置

```
void
transpose_swap(mat3x3_t m, unsigned int i, unsigned int j) {
  unsigned int t = m[i][j];
  m[i][j] = m[j][i];

  m[j][i] = t;
}
```

matinverse.c

```
void
transpose(mat3x3_t m) {
  for (int i=0; i<3; i++)
    for (int j=i+1; j<3; j++){
      transpose_swap(m, i, j);
    }
}
```

矩阵列印

```
void
print_mat(mat3x3_t a)
{
   for (int col=0; col<3; col++) {
      for (int row=0; row<3; row++)
         printf("%5i\t",a[col][row]);
      putchar('\n');
   }

}
```

matinverse.c

三阶行列式计算

```
int
det3x3(mat3x3_t m)
{
int r = m[0][0] * m[1][1] * m[2][2] +.
        m[0][1] * m[1][2] * m[2][0] +
        m[0][2] * m[1][0] * m[2][1] -

        m[0][2] * m[1][1] * m[2][0] -
        m[1][2] * m[2][1] * m[0][0] -
        m[2][2] * m[0][1] * m[1][0];

  return r;
}
```

matinverse.c

主函数

```
mat3x3_t a = {{1,2,3},
              {0,4,5},
              {1,0,6}};

mat3x3_t cofactor_mat = {{0,0,0},
                         {0,0,0},
                         {0,0,0}};
```

```
int main (int argc, char* argv[]) {

  calculate_cofactor_mat(a, cofactor_mat);
  transpose(cofactor_mat);
  print_mat(cofactor_mat);
  printf("* (1/%d)", det3x3(a));
  return 0;
}
```

matinverse.c

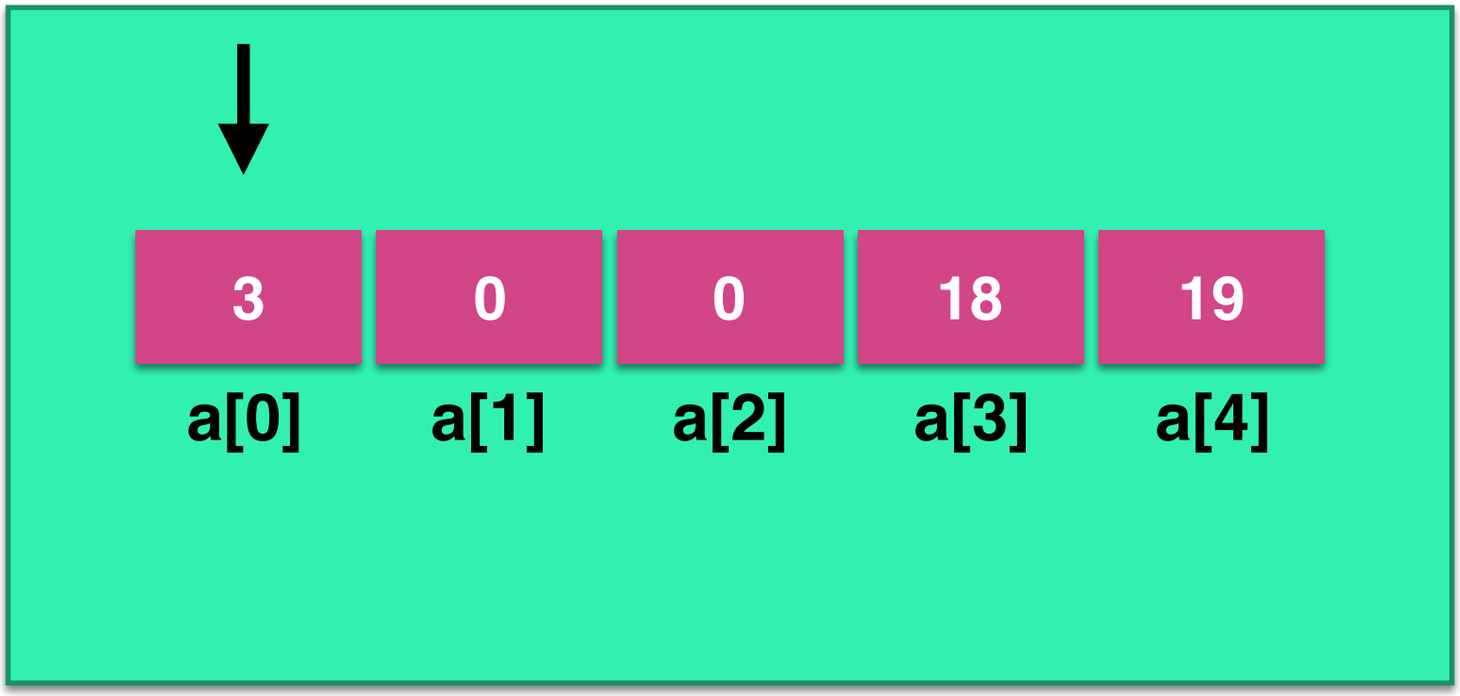
课堂练习

写出程序的执行结果

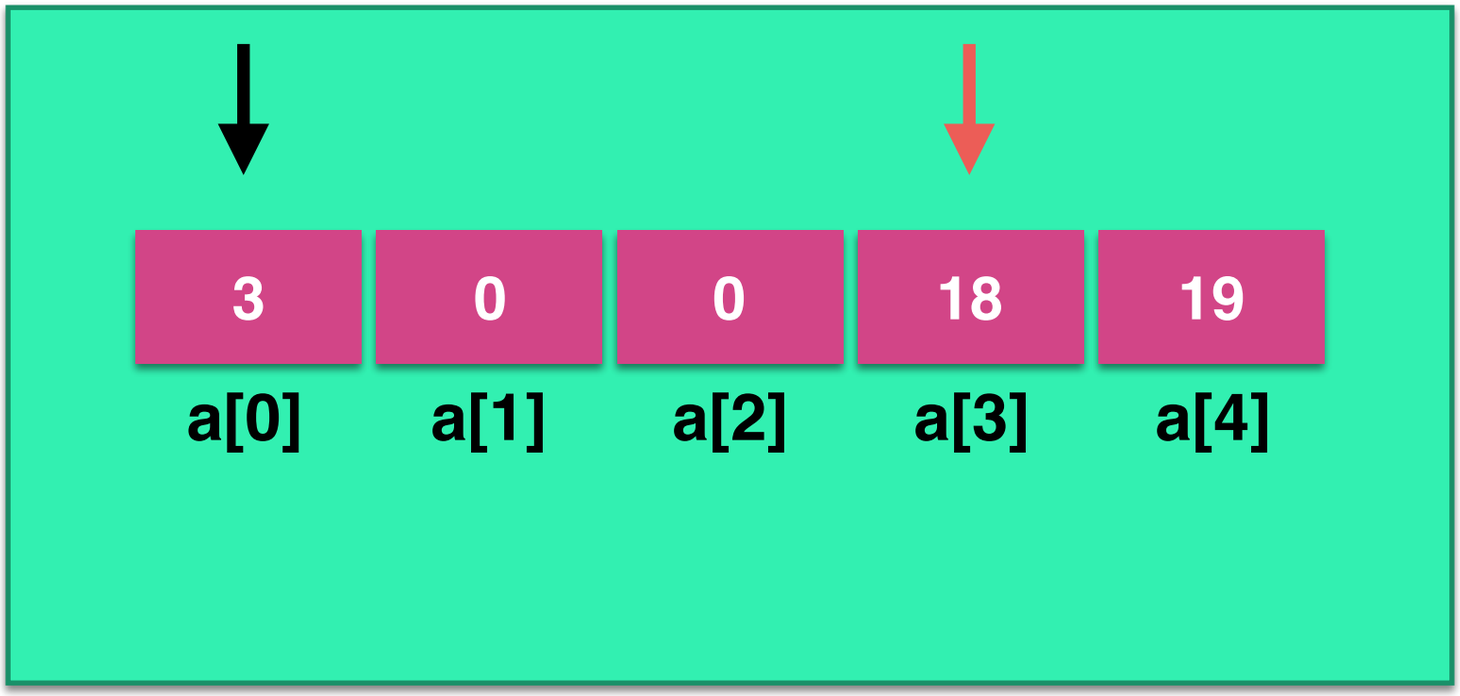
```
int a[100];
int b[100];

void fun(int b[100]) {
    printf("%ld\n", sizeof(b));
}

int main (int argc, char* argv[]) {
    printf("%lu\n", sizeof (a) );
    printf("%lu\n", sizeof(a[100]));
    printf("%lu\n", sizeof(&a));
    printf(“%lu\n", sizeof(&a[0]));
    fun(b);

    return 0;
}
```

素数筛

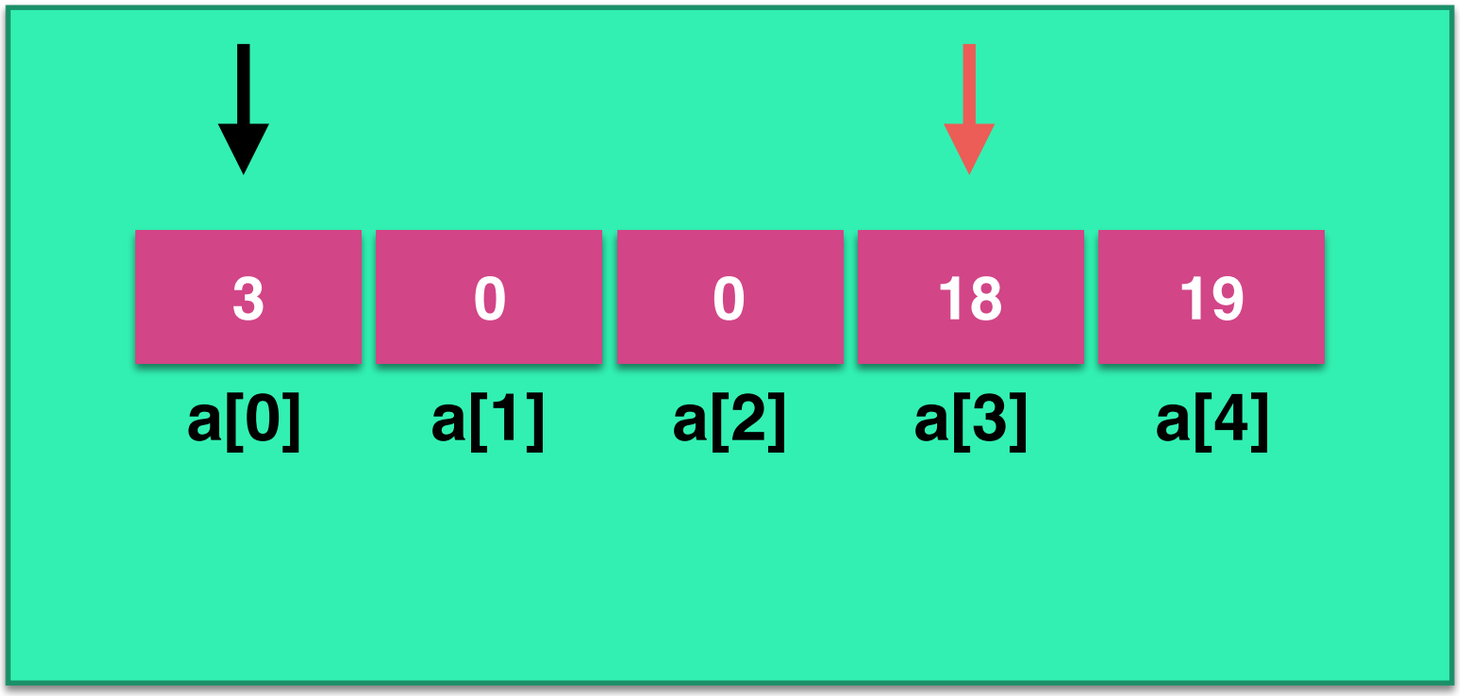
编写函数，从当前位置开始，找出数组中下一个非零元素，返回其下标。



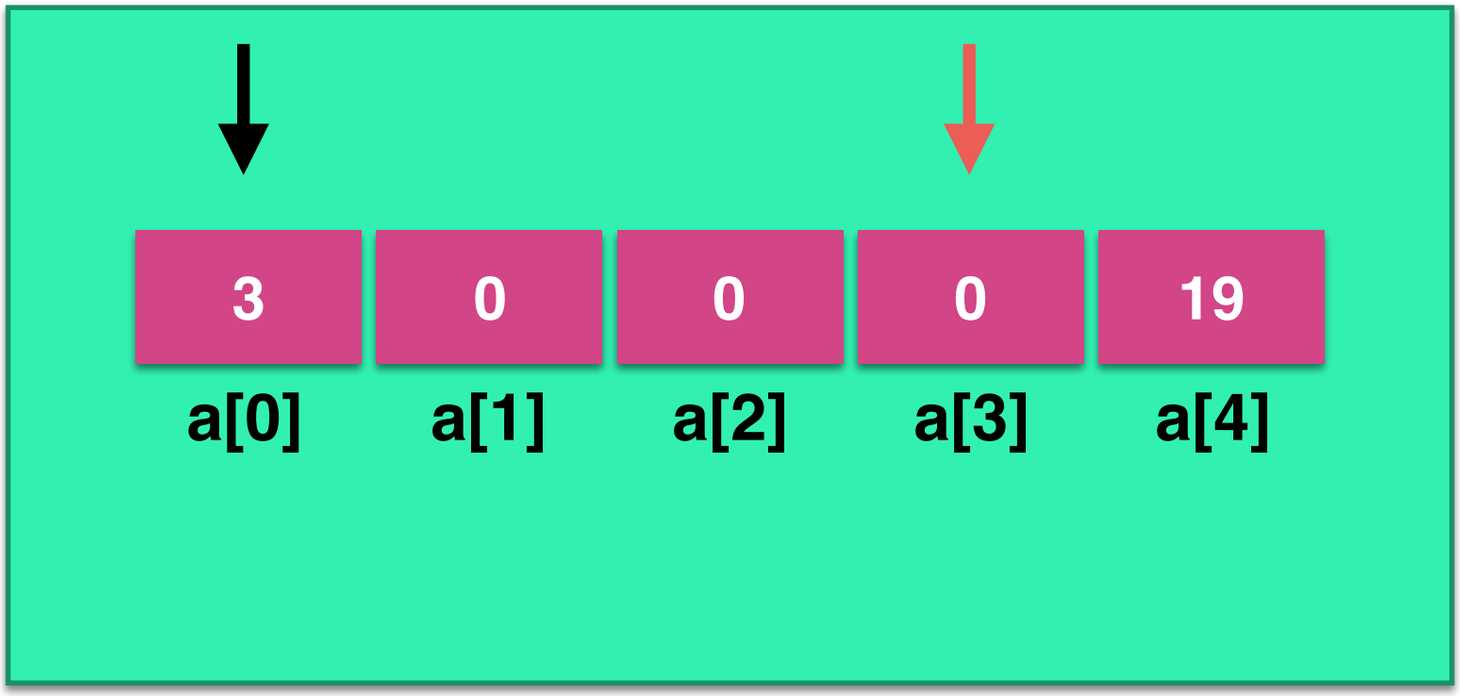
编写函数，从当前位置开始，找出数组中下一个非零元素，返回其下标。



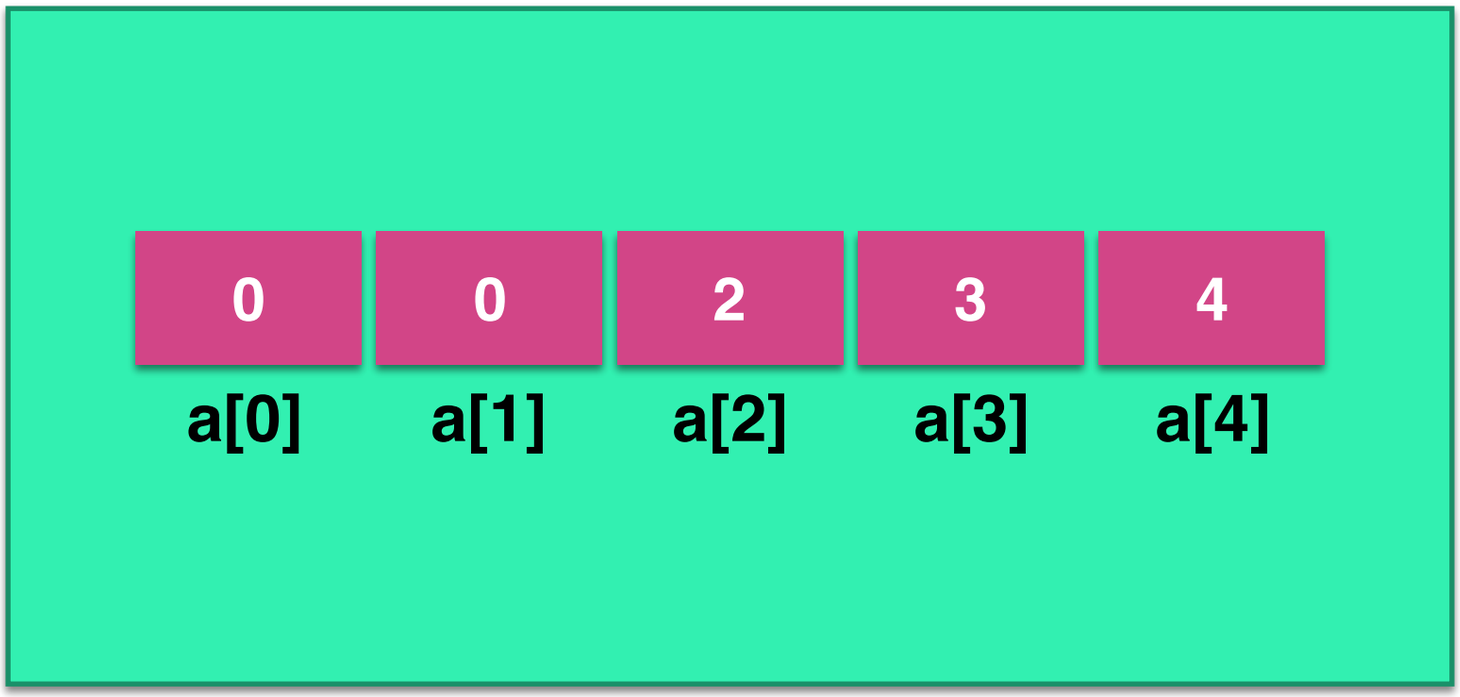
编写函数，从当前位置x开始扫描， 将a[x]可以整除后续数组元素的值置0（标记为删除）。



编写函数，从当前位置x开始扫描， 将a[x]可以整除后续数组元素的值置0（标记为删除）。



编写函数，将输入数组初始化为成连续的整数序列。并赋 a[1] 为0值。



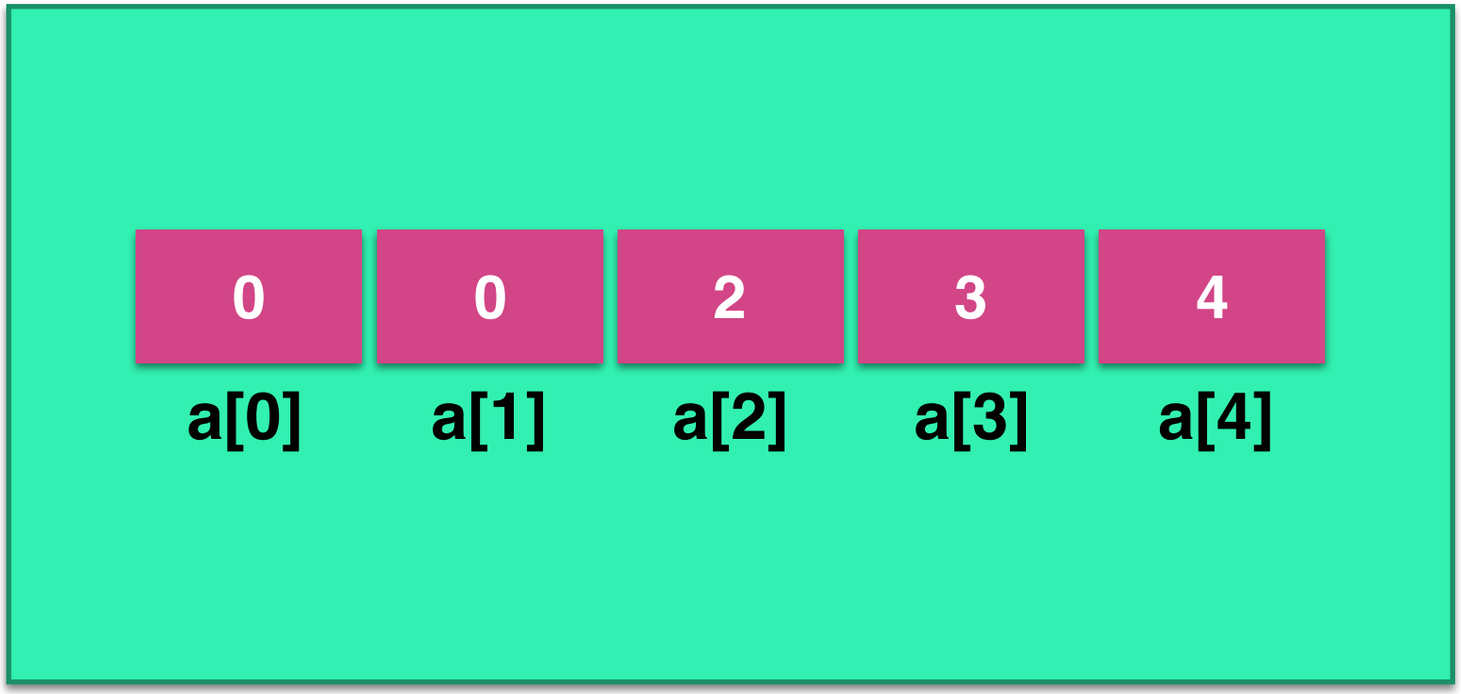
repeat

从当前位置c开始，找出第一个非零元素，记录其位置为loc。

从loc+1位置开始， 标记a[loc]倍数的数组元素。

更新当前位置 c = loc。

until 完成整个数组的扫描。



prime_sieve.c

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

#define SIEVED_FLAG 0
#define MAX_SIZE 10000

const int size = MAX_SIZE;

int arr[MAX_SIZE + 1];

void initial_arr(int arr[], int size) {
    for (int i=0; i<size; i++) {
        arr[i] = i;
    }

    arr[1] = SIEVED_FLAG;  // drop 1
}
```

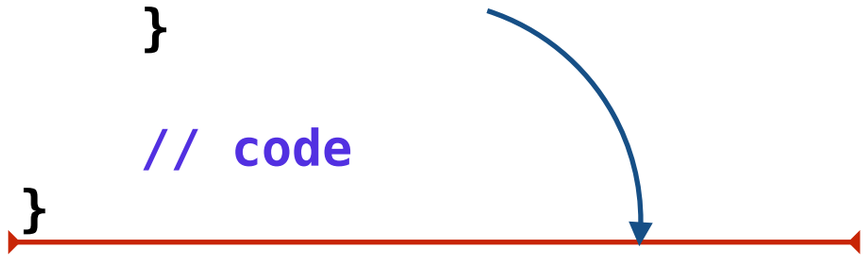
prime_sieve.c

```
void sieve (int arr[], int current_sieve_hole, int from, int size) {
    assert(current_sieve_hole != 0);

    for (int i=from; i<size; i++) {
        if (arr[i] == SIEVED_FLAG) continue;
        if (arr[i] % current_sieve_hole == 0)
            arr[i] = SIEVED_FLAG;  // drop through
    }

}
```

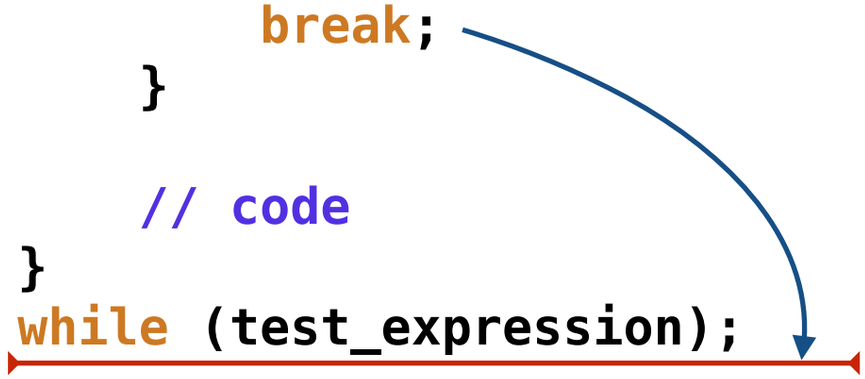
prime_sieve.c

```
void prime_sieve(int arr[], int size) {
    int indicator = 0; // pointing the 1st element in arr, which
                       //
                       // starts from 2, 3, ....
    while (indicator < size) {
        while (arr[indicator] == SIEVED_FLAG ) indicator++;

        if (indicator >= size) break;
        sieve(arr, arr[indicator], indicator+1, size);

        indicator++;
    }
}
```

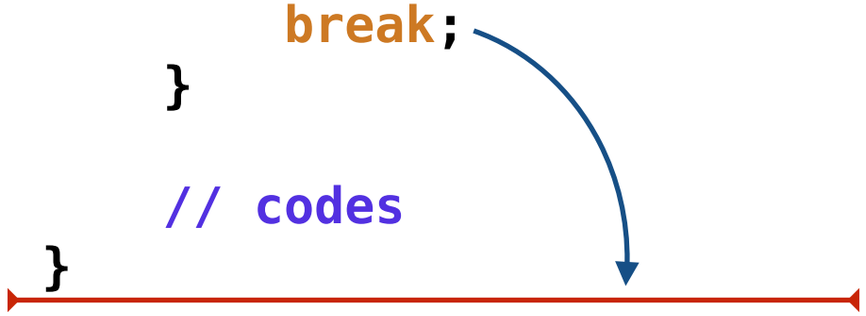
```c
while (test_expression) {
    // codes
    if (condition to break) {
        break;
    }

    // code
}
```



```c
do {
    // codes
    if (condition to break) {
        break;
    }

    // code
}
while (test_expression);
```



```c
for (init; test_expression; update) {
    // codes
    //
    if (condition to break) {
        break;
    }

    // codes
}
```



break

```c
while (test_expression) {
    // codes
    if (condition to break) {
        continue;
    }

    // code
}
```

```c
do {
    // codes
    if (condition to break) {
        continue;
    }

    // code
}
while (test_expression);
```

```c
for (init; test_expression; update) {
    // codes
    //
    if (condition to break) {
        continue;
    }

    // codes
}
```

continue

break 还可以跟 switch 语句配合使用。

prime_sieve.c

```
44 void print_prime_tbl(int arr[], int size) {
45     for(int i=0; i<size; i++) {
46         if (arr[i] == SIEVED_FLAG) continue;  // skip 0-val elements
47 
48         if (i % 10 == 0) putchar('\n');
49         printf("%5d", arr[i]);
50     }
51 }
52 
53 
54 int main(int argc, char* argv[]) {
55     initial_arr(arr, size);
56     prime_sieve(arr, size);
57     print_prime_tbl(arr, size);
58     return 0;
59 }
```

抽象数据类型

函数 与 复合数据类型

array[]

func()

unsigned double

char signed

float long short

存储操作

int

= unsigned int

& ~ -(单目)

运算符与表达式

+ - * / % ! || && == != < > <= >=

continue

break

程序流程控制

goto if…else… for

cos
fabs

putchar()

printf()

库函数

WENZHENG COLLEGE OF SOOCHOW UNIVERSITY

2018.11.29

Soochow University