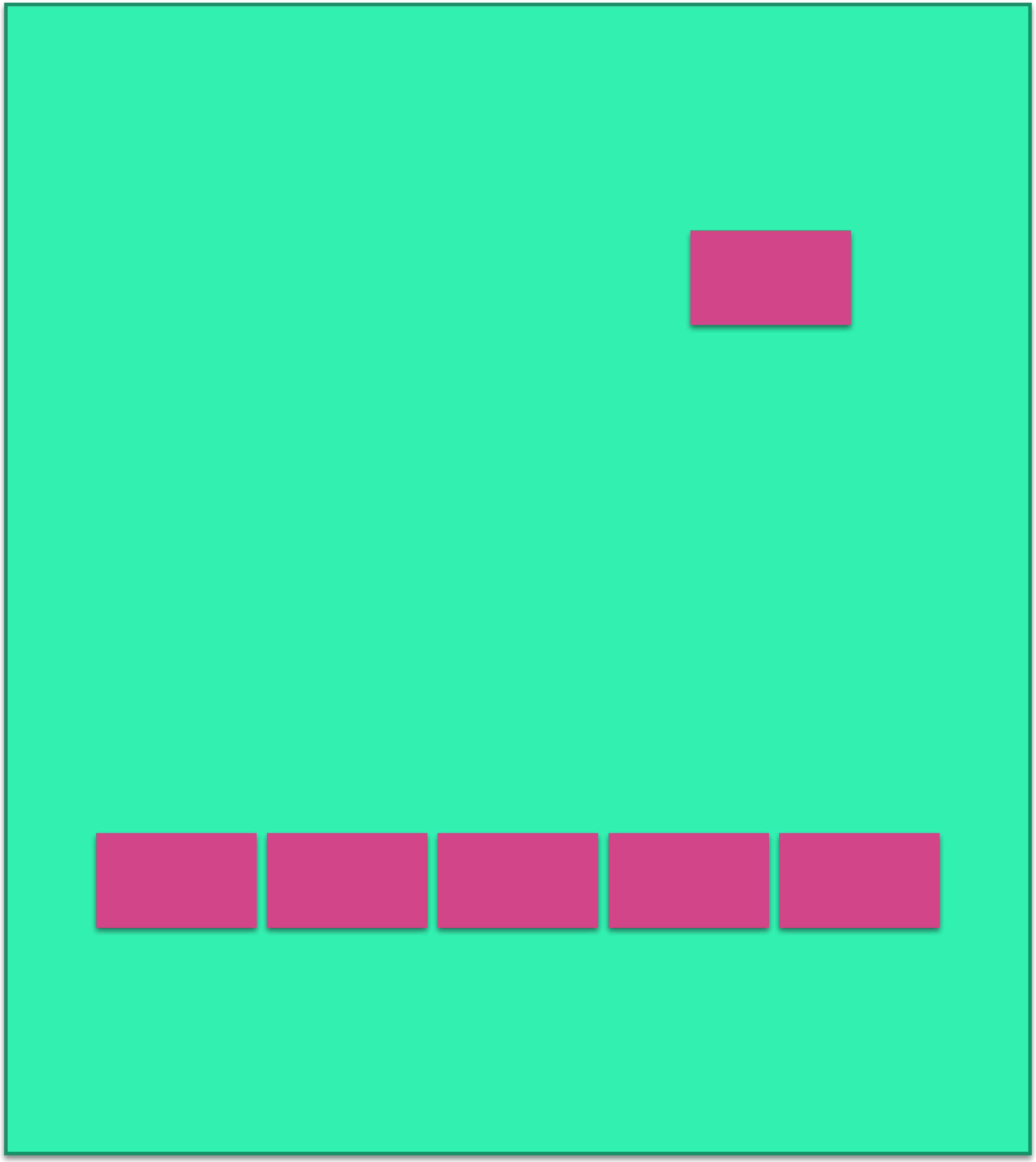
数组

huiw@suda.edu.cn

前面的学习中，我们处理的都是**标量**。

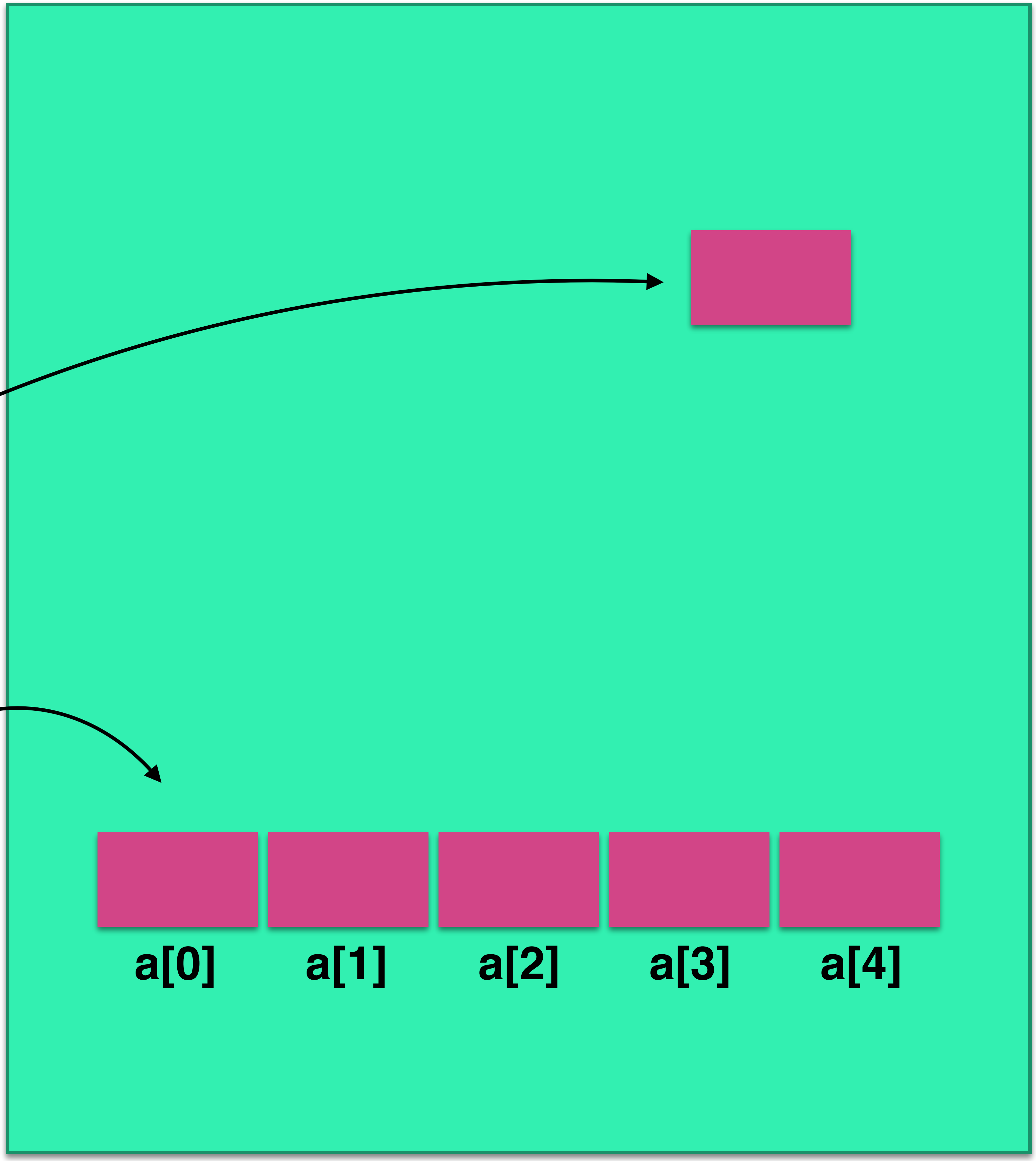
C语言为我们提供了相同类型数据的聚集能力。

```
int a;


int arr[5];
```



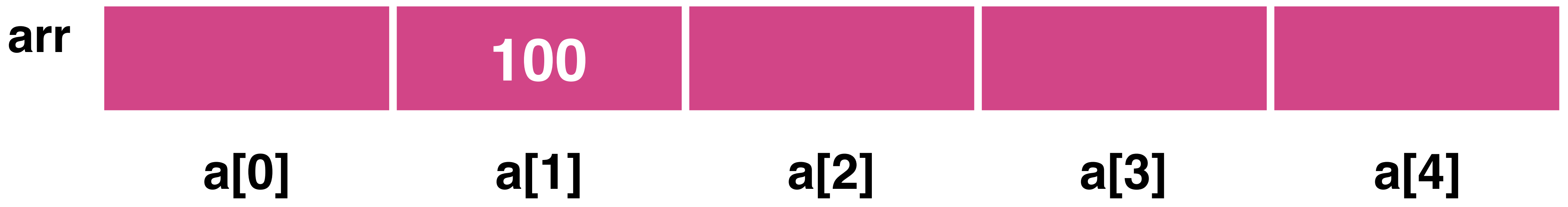
```
int a;
```

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



数组元素的赋值

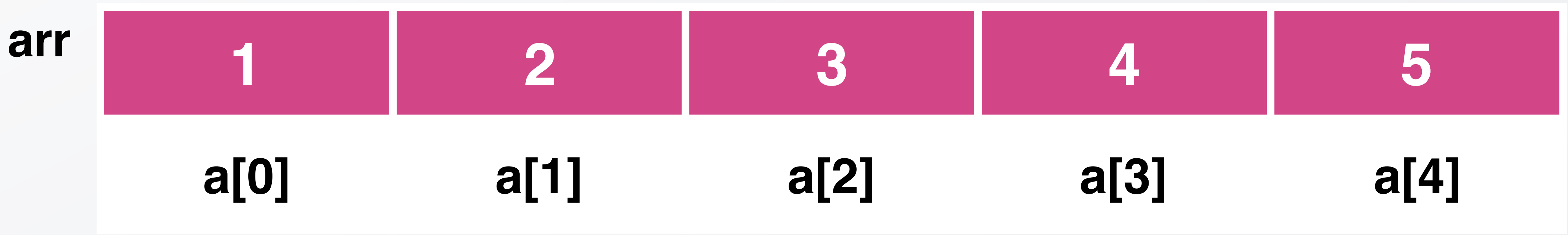
```
arr[1] = 100;
```



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};
```

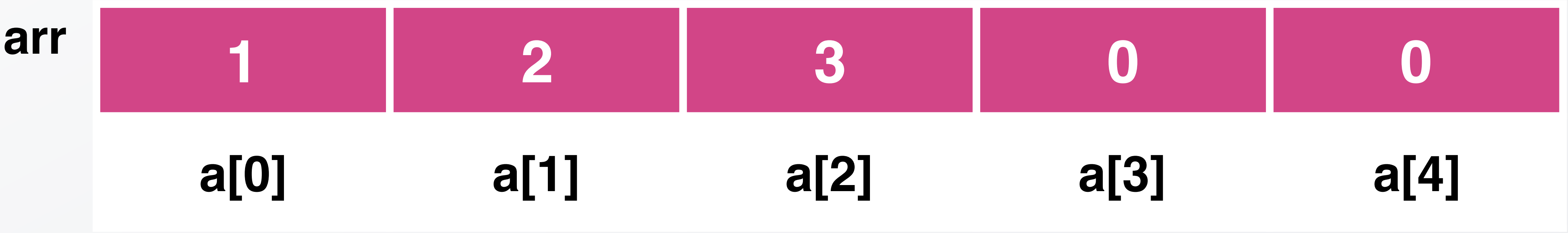


arr

1	2	3	4	5
a[0]	a[1]	a[2]	a[3]	a[4]

数组元素的初始化

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



arr	1	2	3	0	0
	a[0]	a[1]	a[2]	a[3]	a[4]

```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} */
```

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

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

```
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}.
 *.
 */
```

```c
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;
```

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.

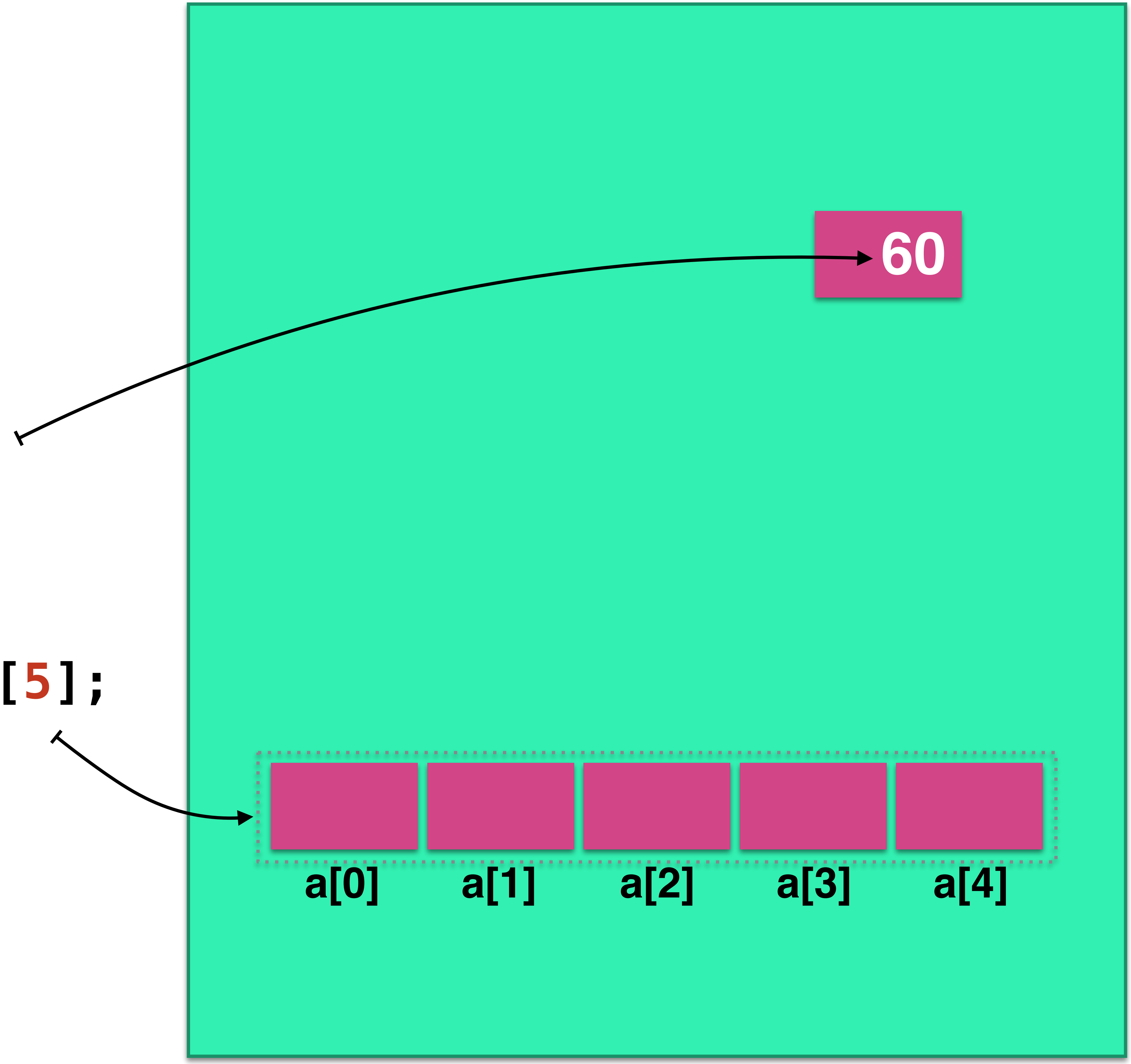
```
int arr[5] = {5,6};
```

✸ arr 的右值是?

arr在内存中的 **位置** .

```
int a;
```

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



循环语句 for

```
iteration_statement
   :for '(' expression_statement expression_statement ')' statement
   | for '(' expression_statement expression_statement expression ')'
statement
```

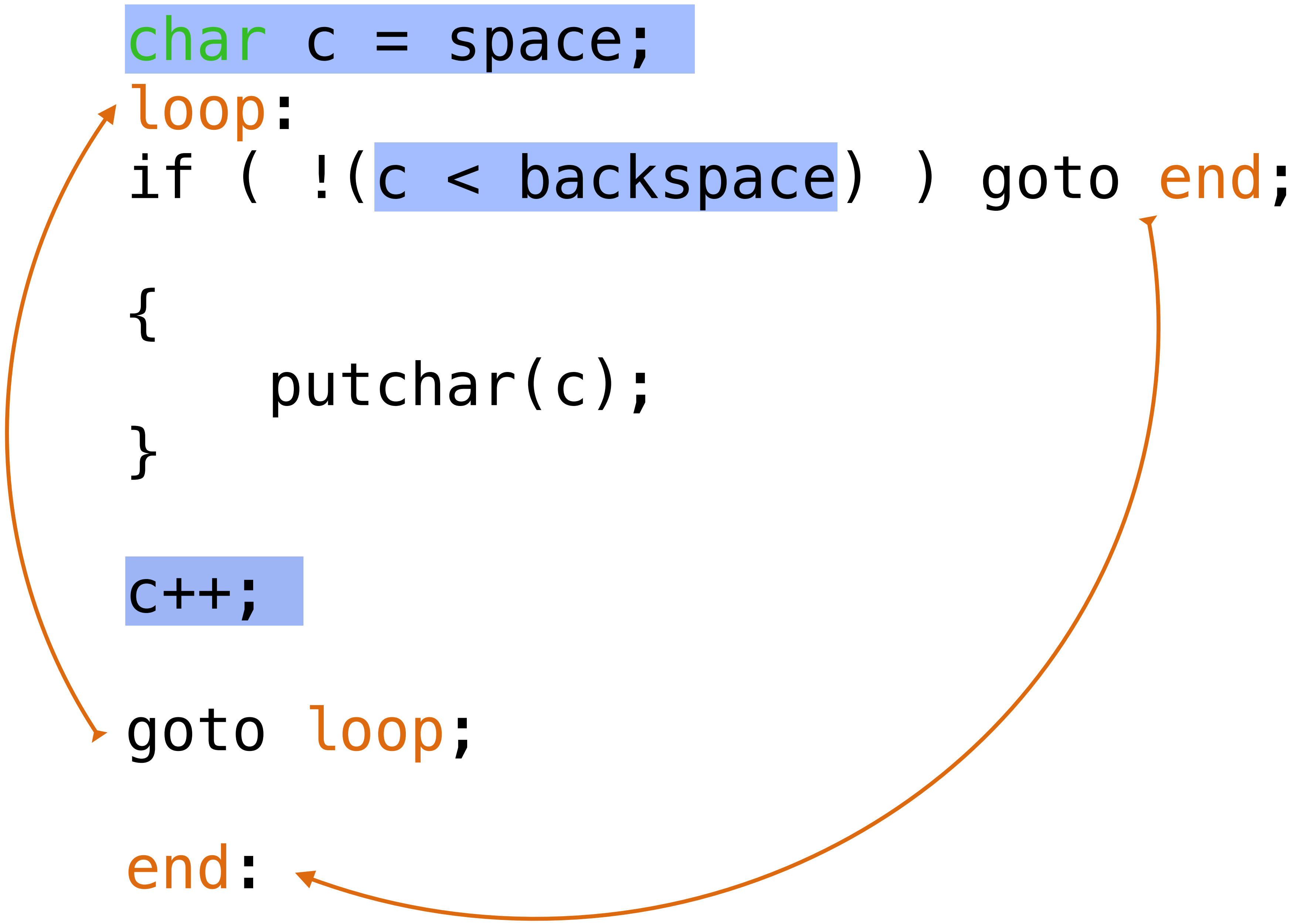
```
for (;true;) {
  putchar('A');
}
```

死循环

```
for (;;) {
  putchar('A');
}
```

```c
const char space     = '\x20';
const char backspace = '\x7F';

for (char c = space; c < backspace; c++) {
  putchar(c);
}
```

```
 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
```



```
char c = space;
loop:
if ( !(c < backspace) ) goto end;

{
    putchar(c);
}

c++;

goto loop;

end:
```

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
```

判断一个正整数是否有相同数字

```
bool
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;
}
```

编程计算e的值

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

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; 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;
}
```

```
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;
}
```

```
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;
}
```

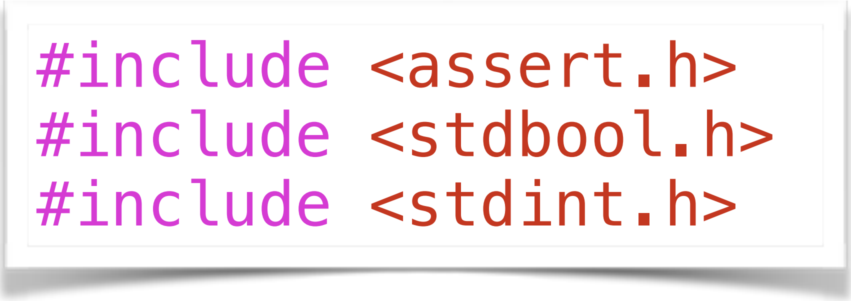
二分查找（递归实现）

```
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;
}
```

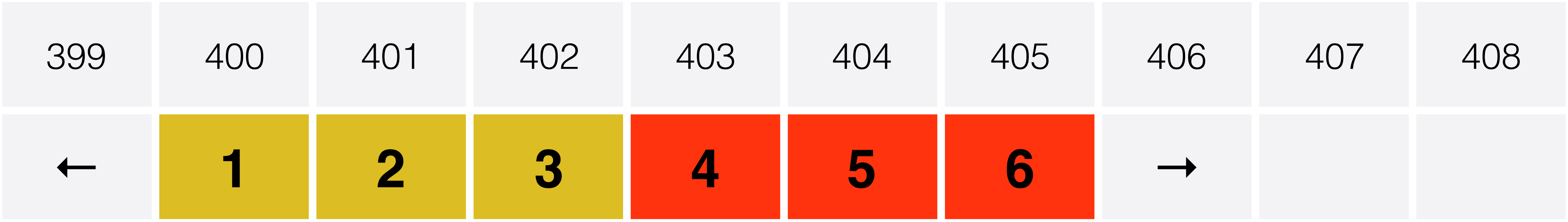


```c
#include <assert.h>
#include <stdbool.h>
#include <stdint.h>
```

```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;
}
```

数组的数组

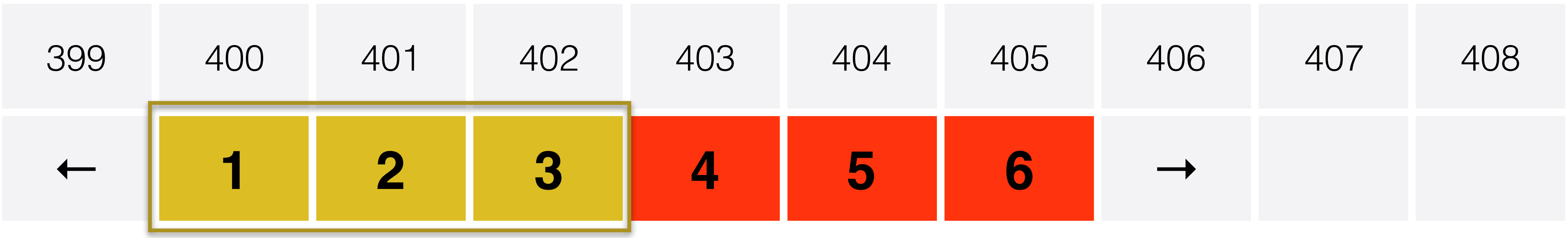
```
// array of integers
int a[3];.

// 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}};
```

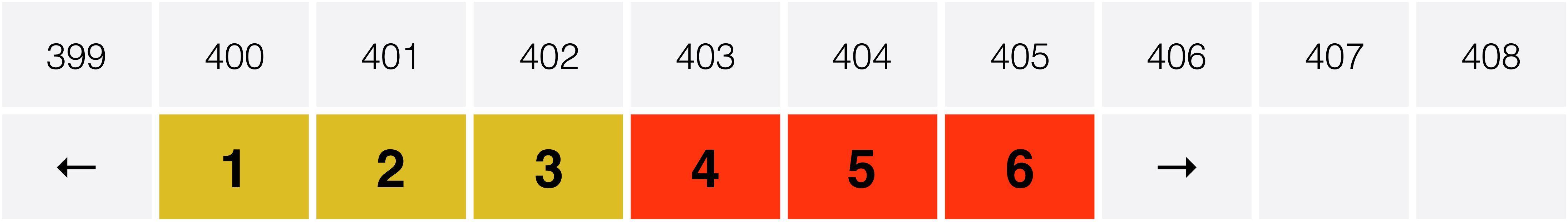
b[0]



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}};
```

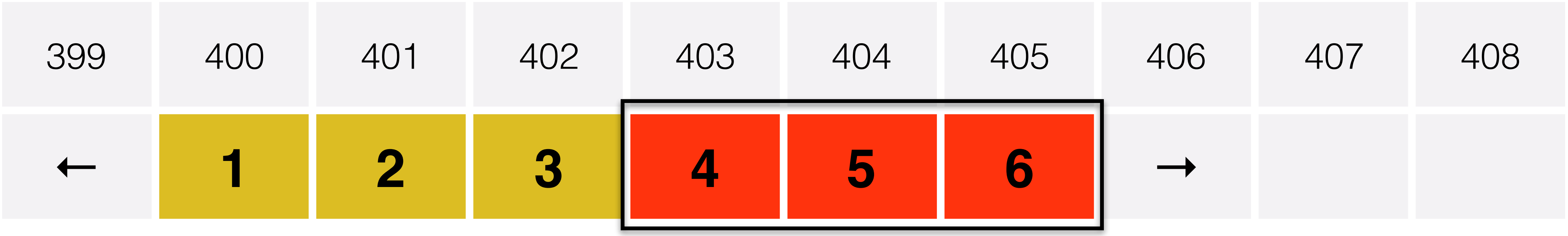
```
    b[0]
```



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]



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

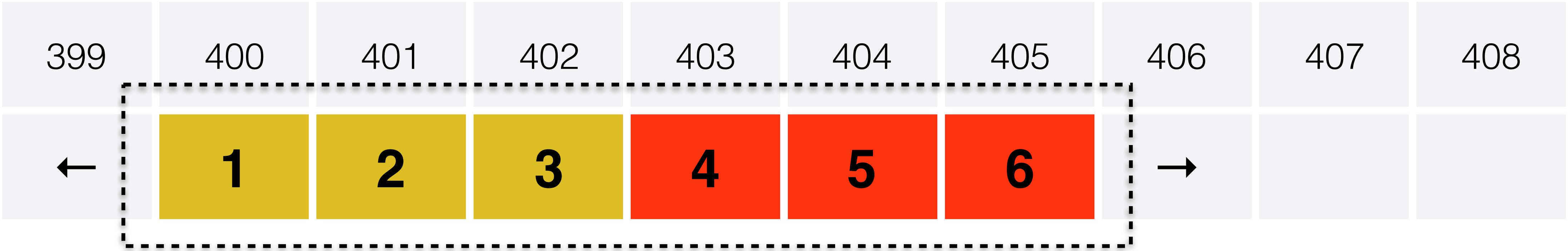
```
// array of array of integers
int b[2][3] = {{1, 2, 3}, {4, 5, 6}};
```

```
b[1]
```

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[2][3]
```

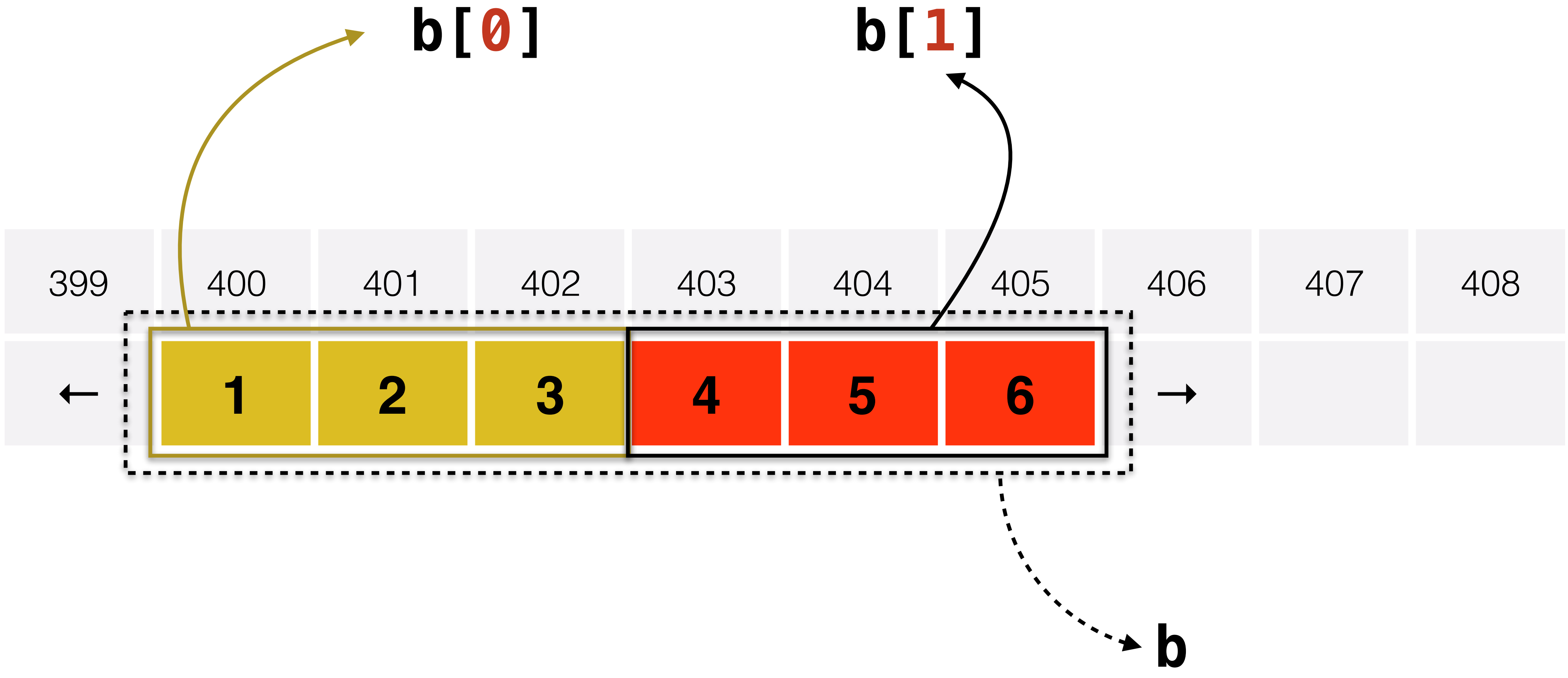


399	400	401	402	403	404	405	406	407	408
←	1	2	3	4	5	6	→		

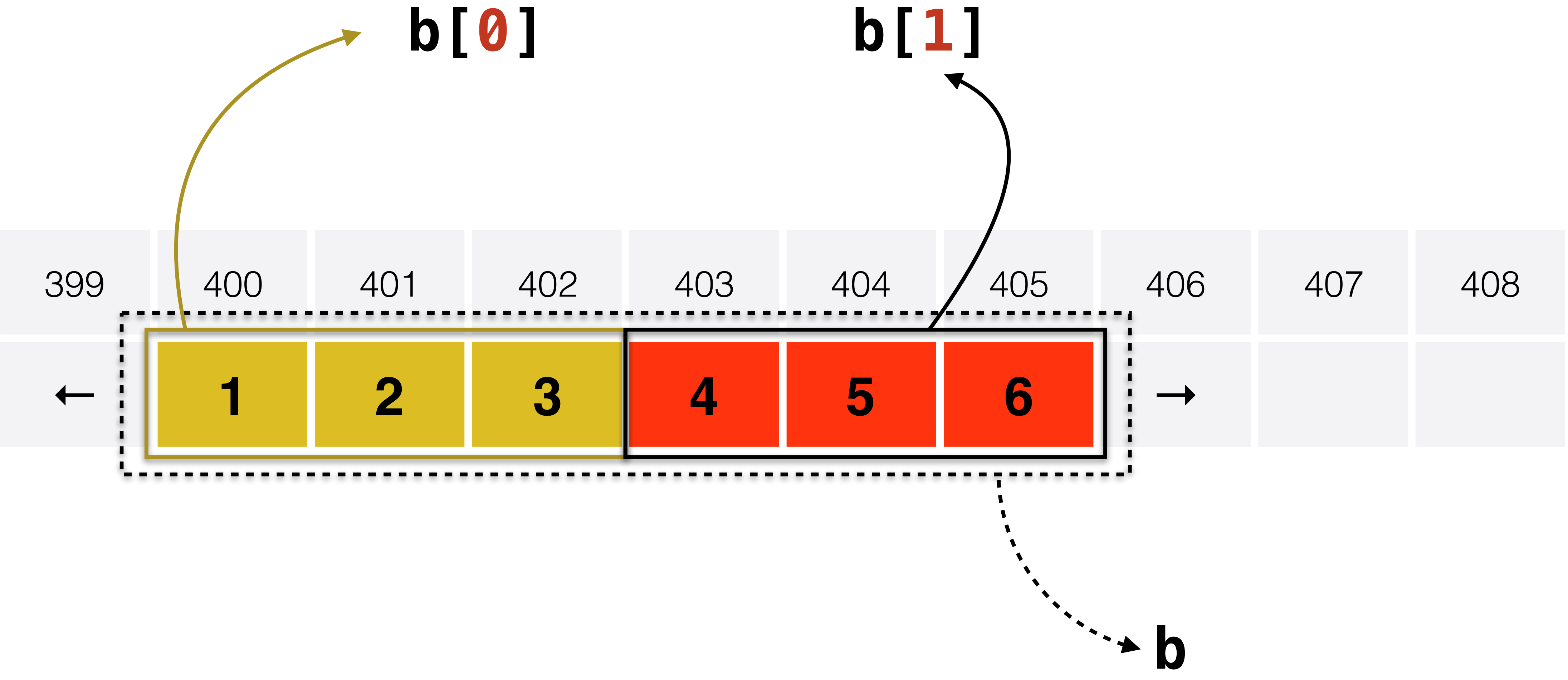
```
// array of array of integers
int b[2][3] = {{1, 2, 3}, {4, 5, 6}};
```

```
   b[2][3]
```

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

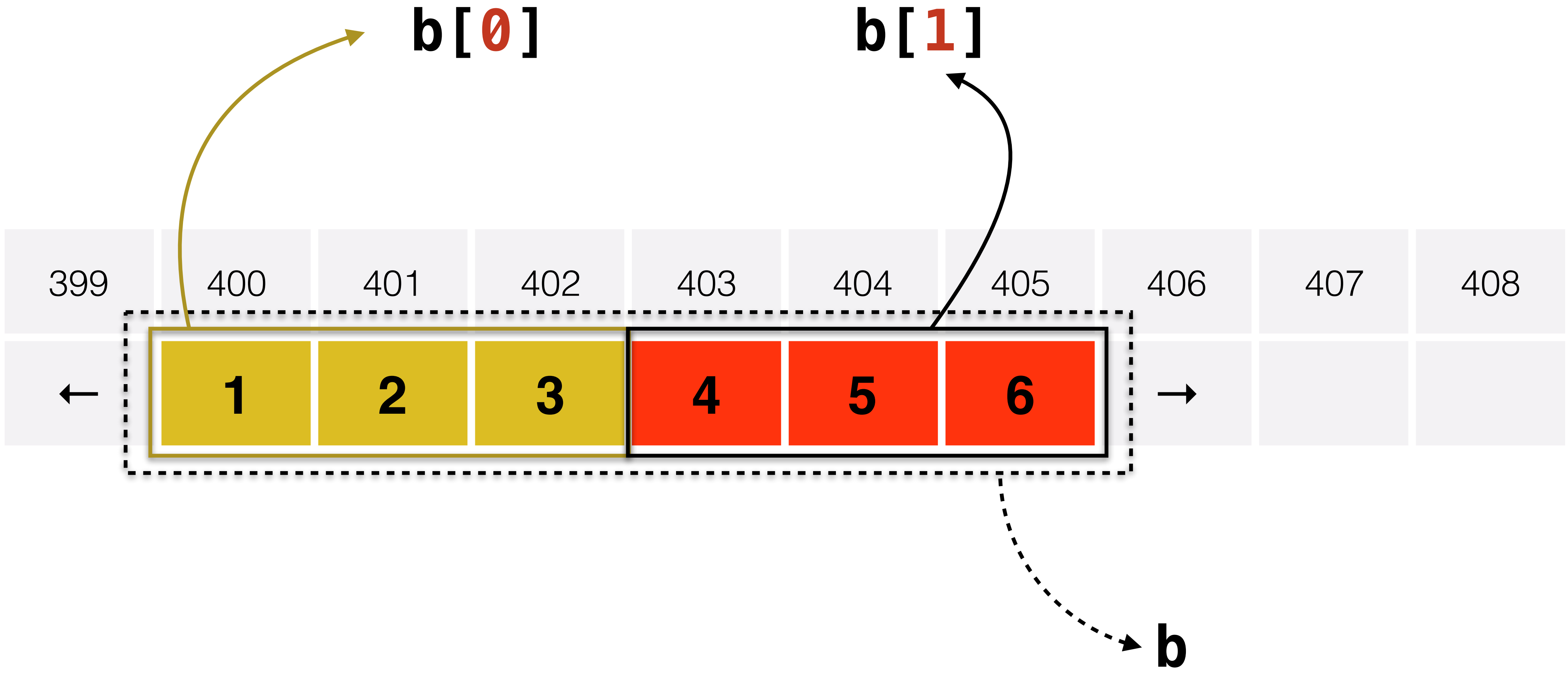


```
// array of array of integers
int b[2][3] = {{1, 2, 3}, {4, 5, 6}};
```



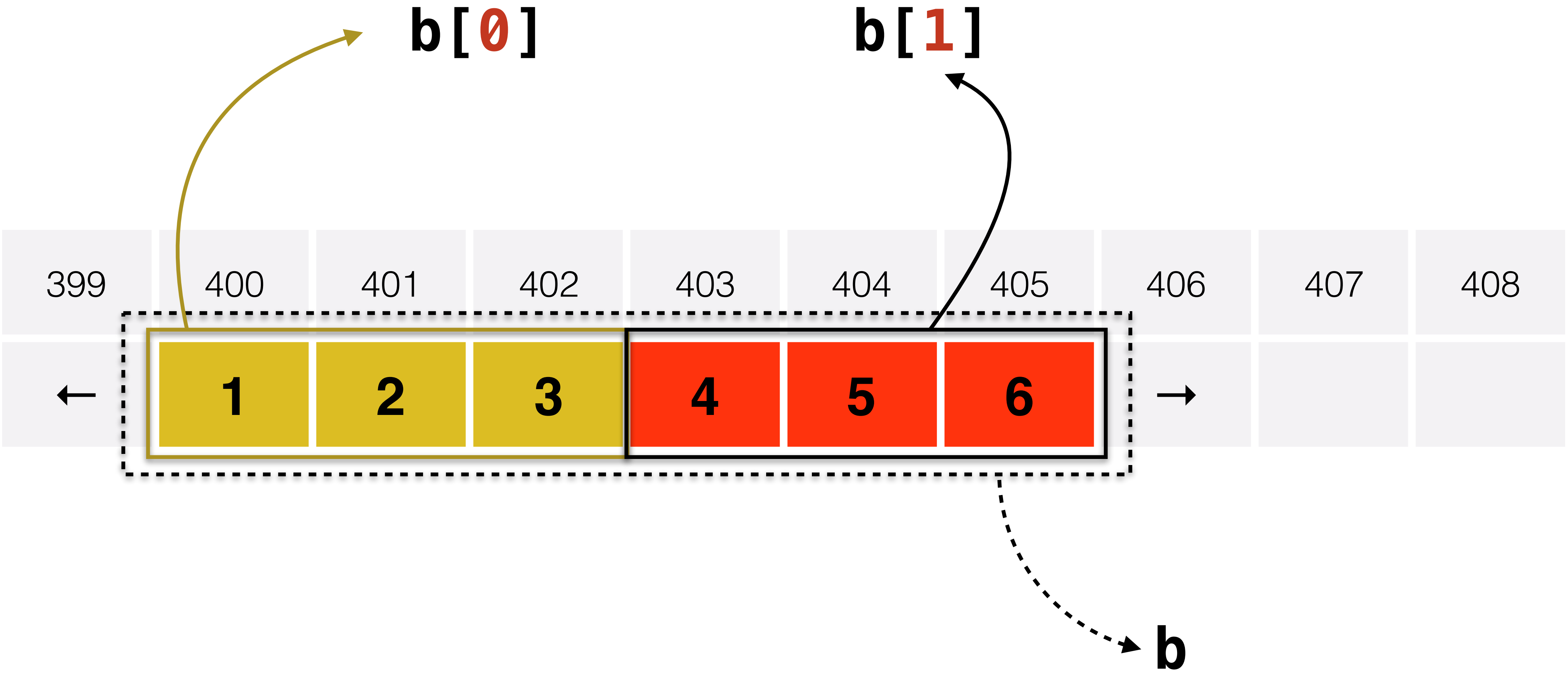
b是个二维整数数组，其右值是400

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



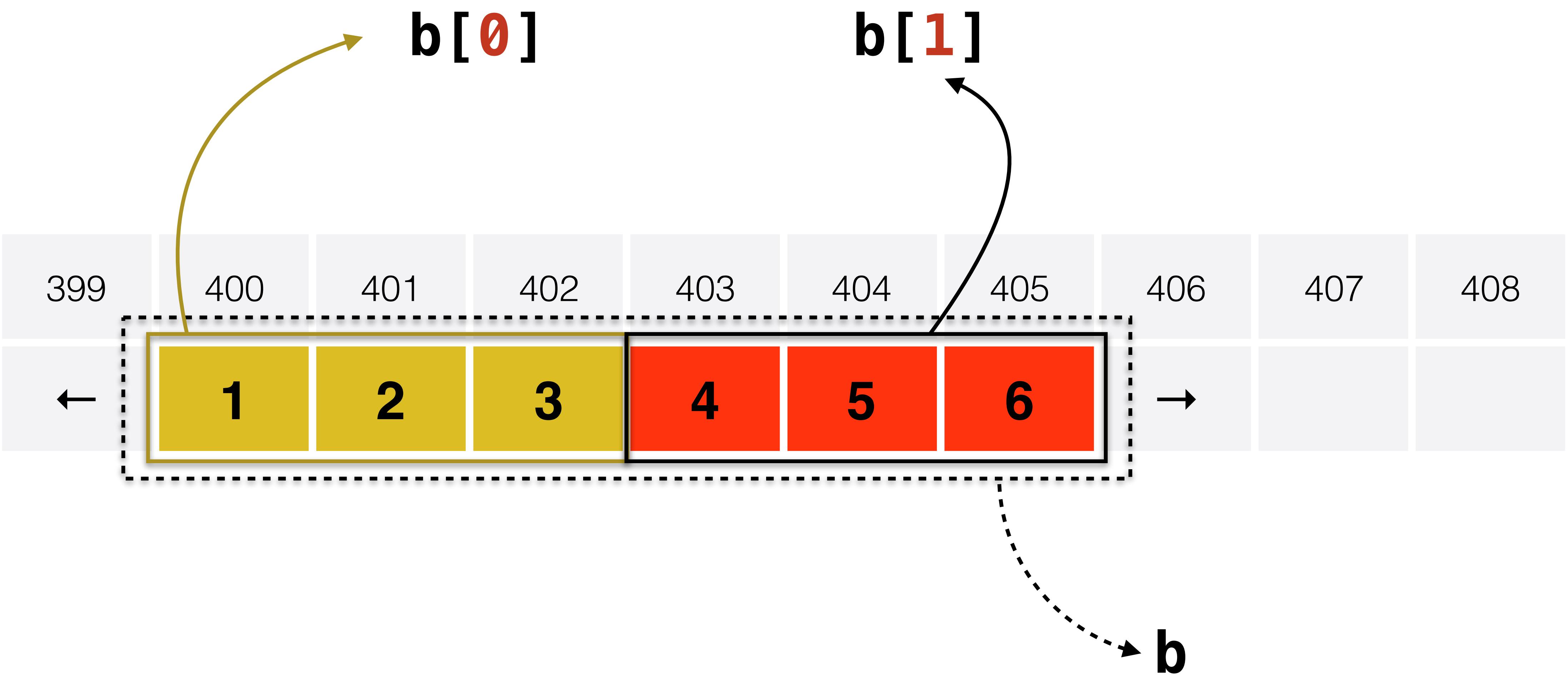
b[0]是个一维整数数组，其右值是 400

```
// 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

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



b[0]

b[1]

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

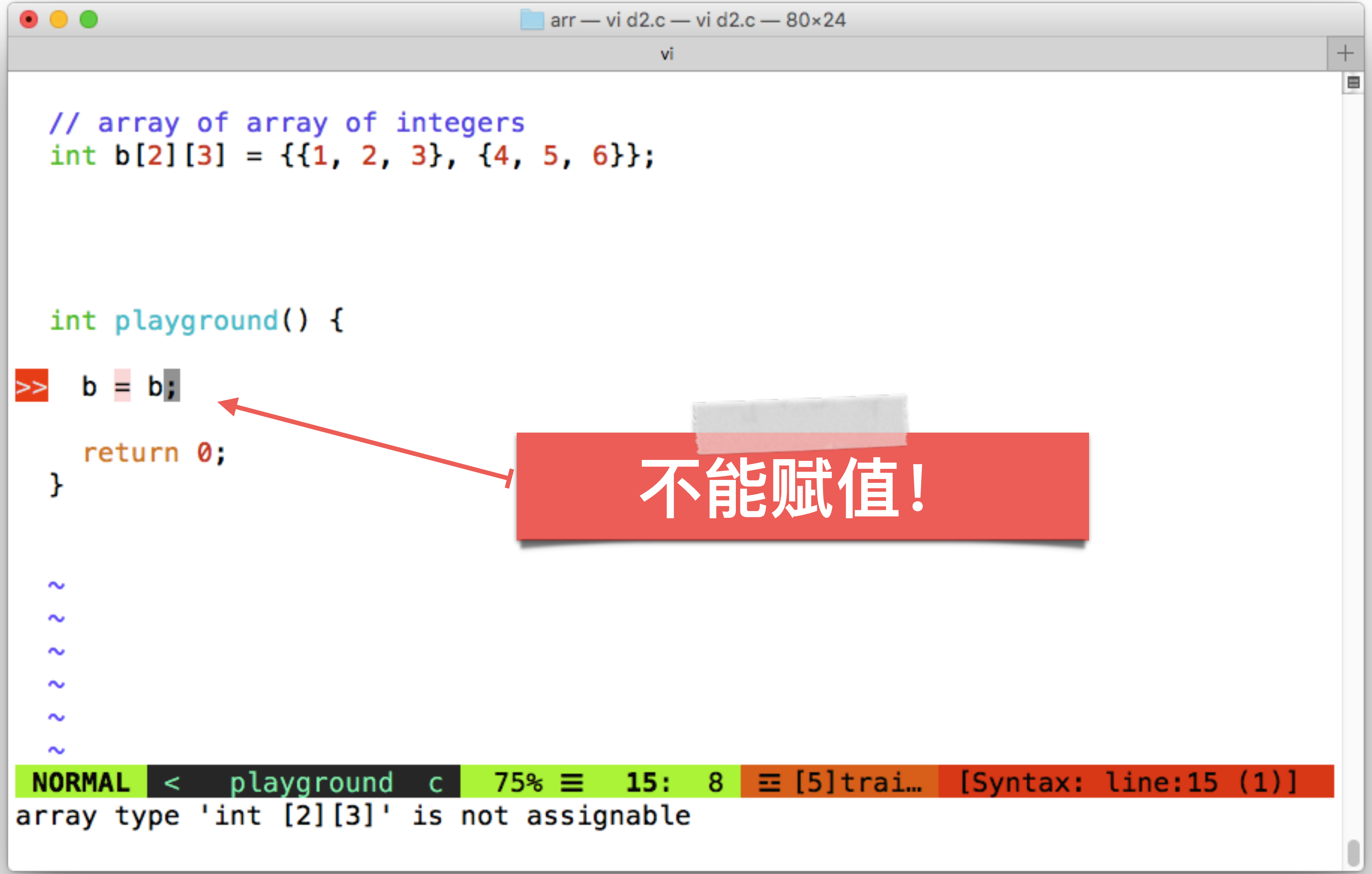
b

b[0][0]是个整数变量，其右值是 1 ，其左值为400.

```c
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**。



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



int playground() {

>>  b = b;

    return 0;
}

~
~
~
~
~
~
 NORMAL  <   playground  c   75% ≡  15:  8  ☲ [5]trai…  [Syntax: line:15 (1)]
array type 'int [2][3]' is not assignable
```

不能赋值！

计算矩阵的和

```
#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];
    }
}
```

```
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;.
}
```

计算3阶矩阵的逆阵

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

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

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

$$\begin{bmatrix} + & - & + \\ - & + & - \\ + & - & + \end{bmatrix}$$

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

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

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

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;
}
```

计算代数余子式的值

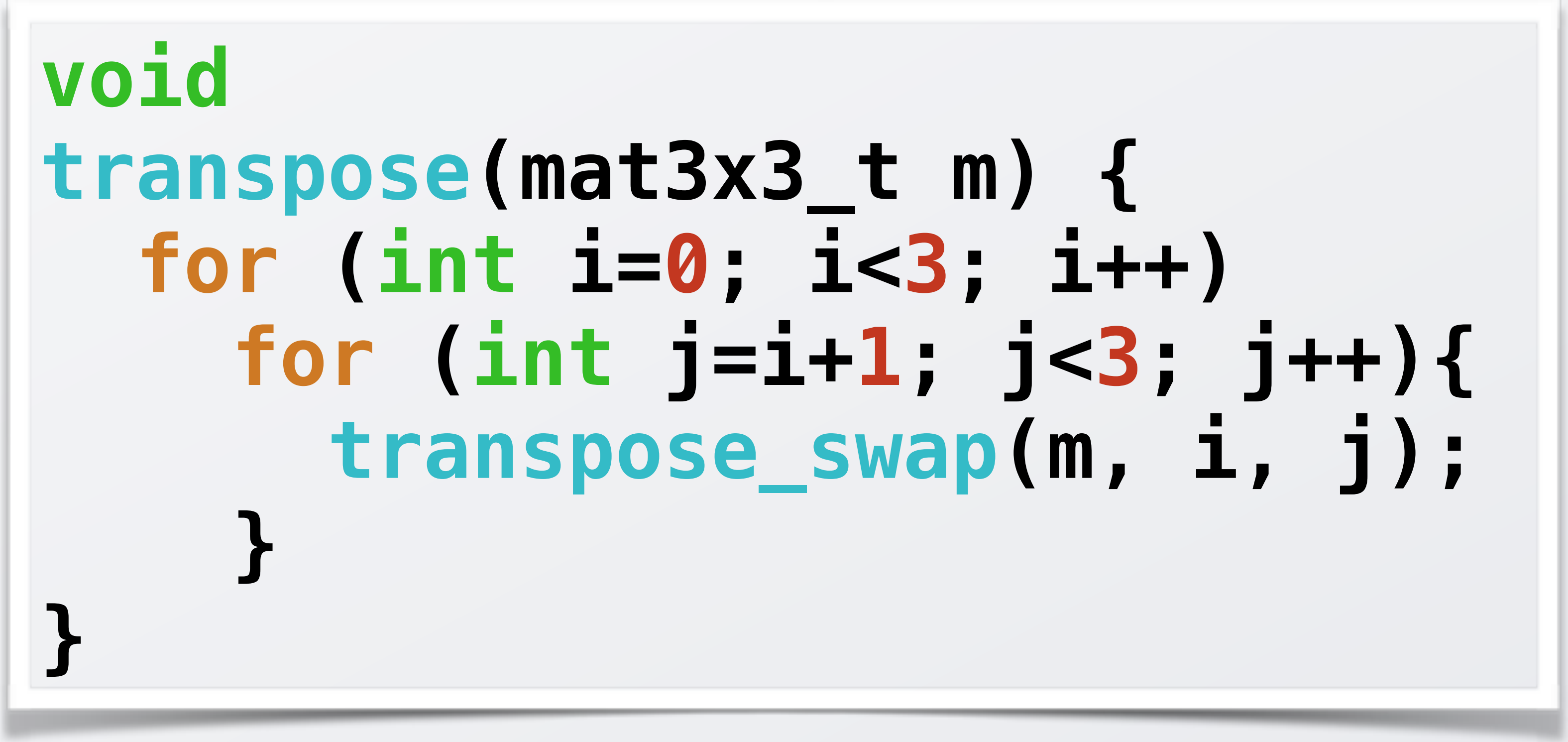
```
int det2(int a, int b, int c, int d) {
  return a * d - b * c;
}

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]);
}
```

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);
    }

}
```

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;
}
```



```
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');
  }

}
```

三阶行列式计算

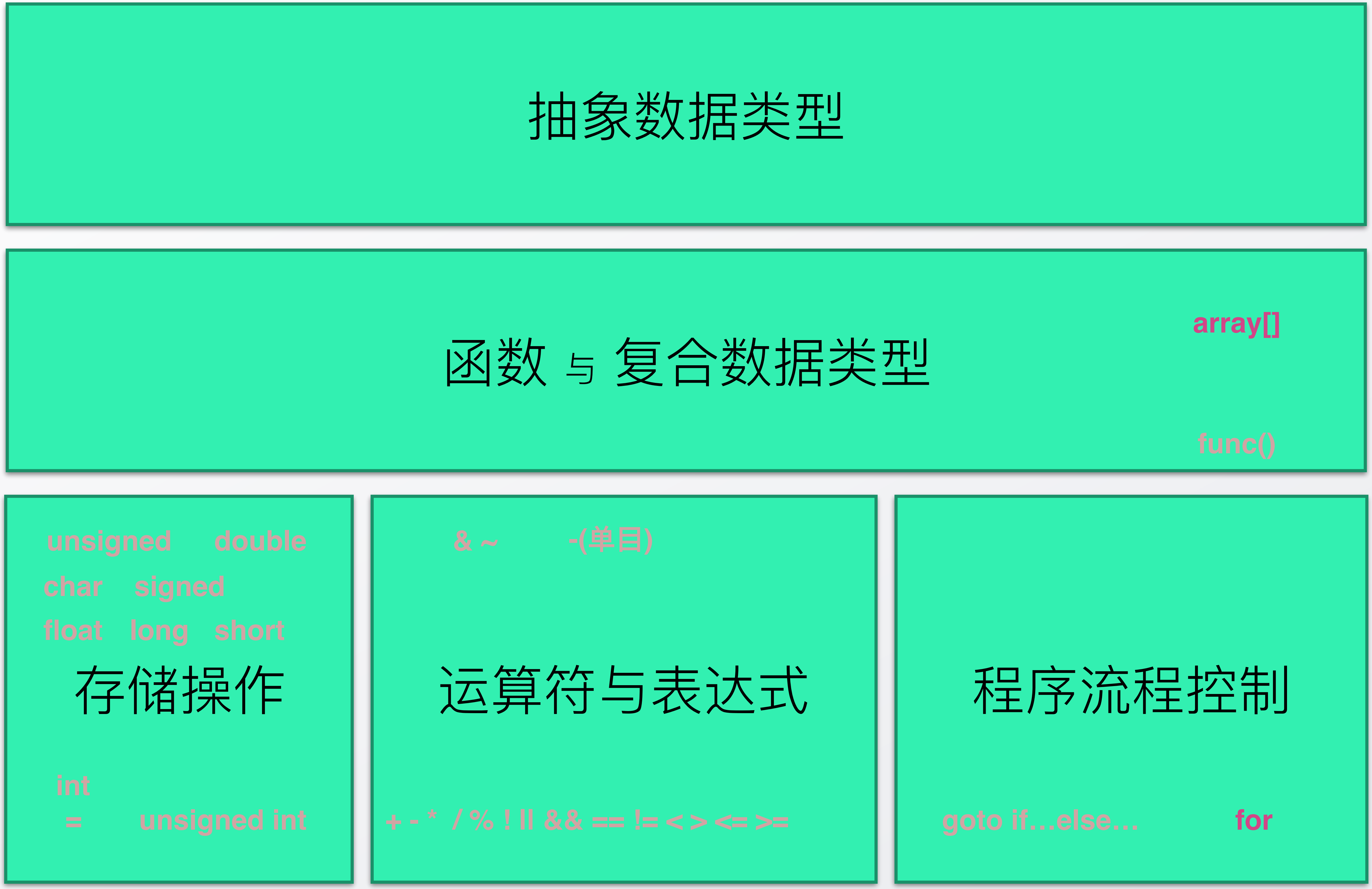
```
int
det3x3(mat3x3_t m)
{
int r = m[0][0] * m[1][1] * m[2][2] +.
        m[0][1] * m[1][2] * m[2][0] +
        m[0][2] * m[1][0] * m[2][1] -

        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;
}
```

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;
}
```

课堂练习

二阶矩阵求逆



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

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

程序流程控制

goto if…else… for

cos
fabs

putchar()

printf()

库函数

WENZHENG COLLEGE OF SOOCHOW UNIVERSITY

2017.3.29

Soochow University

附录