程序设计
Programming

Lecture 5: 数组

1、程序的内存分布

C语言程序的内存分布（简化版）

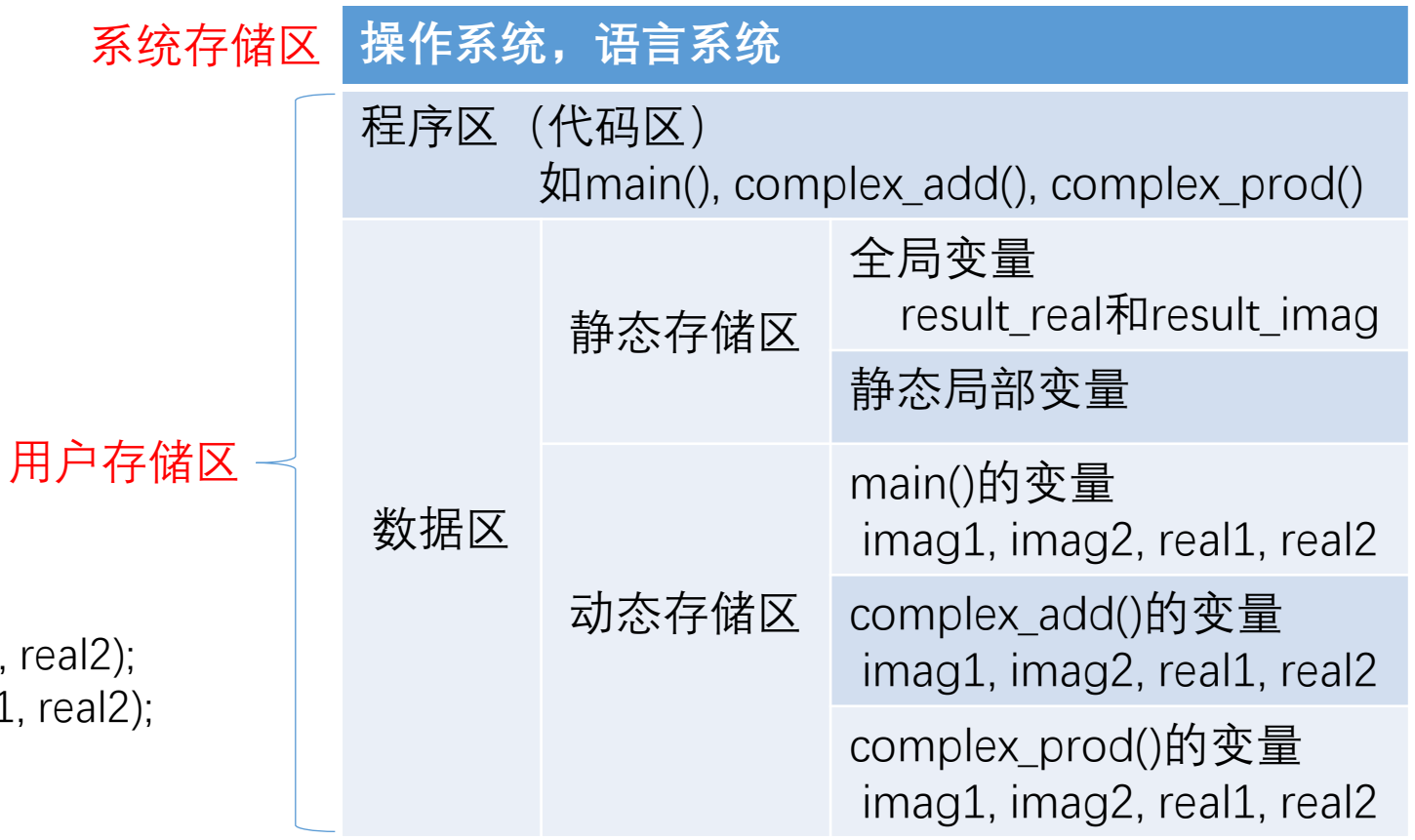
- 书本111页，例5-7
- 书本116页，图5.3

```
float result_real, result_imag;
void complex_add (……);
void complex_prod (……);

int main (void)
{
        float imag1, imag2, real1, real2;
        ……
        complex_add (imag1, imag2, real1, real2);
        complex_prod (imag1, imag2, real1, real2);
        return 0;
}

void complex_add (……) {}
void complex_prod (……) {}
```

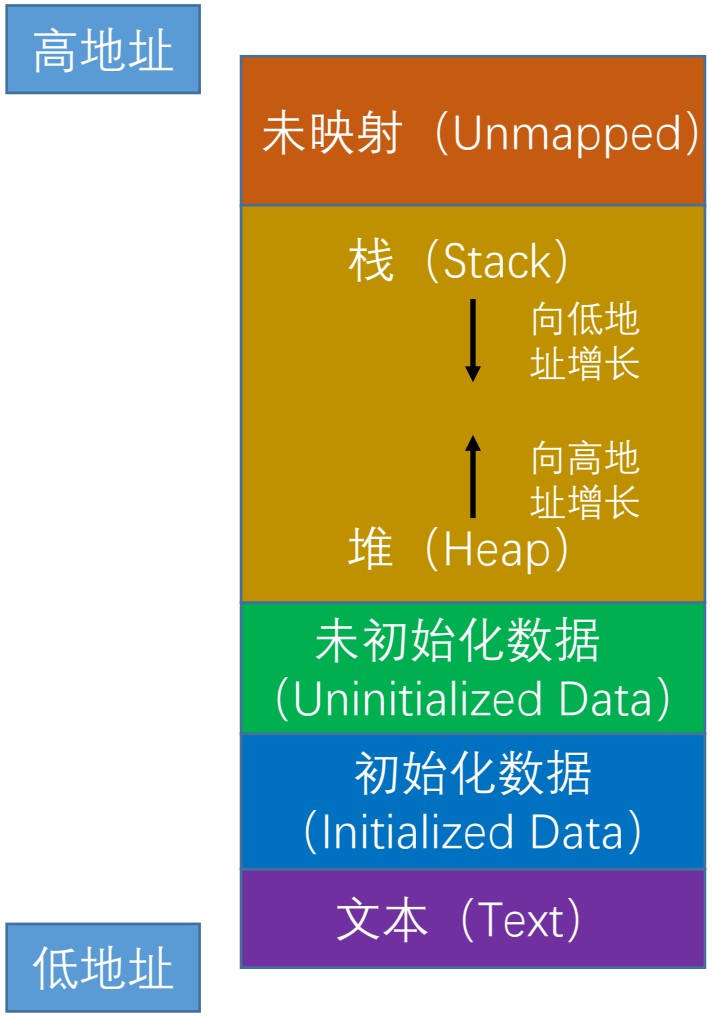
C语言程序的内存分布（简化版）

- 书本111页，例5-7
- 书本116页，图5.3

```
float result_real, result_imag;
void complex_add (……);
void complex_prod (……);

int main (void)
{
        float imag1, imag2, real1, real2;
        ……
        complex_add (imag1, imag2, real1, real2);
        complex_prod (imag1, imag2, real1, real2);
        return 0;
}

void complex_add (……) {}
void complex_prod (……) {}
```



C语言程序的内存分布（详细版，了解）



操作系统内核区，存放命令行参数，环境变量等，可以被用户进程通过系统调用访问。

堆栈区，动态分配内存和存放局部变量。

- ✓ 堆：程序申请的动态分配内存区域，如malloc(), calloc(), realloc()和C++的new。需要使用free()和C++的delete来释放内存。
- ✓ 栈：系统分配的连续内存区域。存放局部变量，包括参数、返回地址、局部变量等。当函数被调用时，创建一个stack frame并压栈。当函数调用结束时，对应的stack frame被销毁并返回（出栈）。由栈指针跟踪栈区域的内存地址。

存放未初始化的和初始化为0的全局变量和静态局部变量。又称为BSS段，Block Started by Symbol.

存放已初始化为非0的全局变量和静态局部变量。又分为只读区域和可读写区域。

存放程序代码，即可执行的机器指令。一般为只读区域。可以被多个进程共享。

2、数据类型与表达式回顾

C语言数据类型

- 基本数据类型
  - ✓整型、字符型、浮点型（单\双精度）
- 构造数据类型
  - ✓数组、结构（struct）、联合（union）、枚举（enum）
- 指针类型
- 空类型

各数据类型在内存中的表示

- L03-数据类型和控制语句
  - ✓有符号、无符号整型
  - ✓单精度、双精度浮点型

C语言整型常数的三种编写方法

- 十进制
  - ✓123，-2345

C语言整型常数的三种编写方法

- 十进制
  - ✓123，-2345
- 八进制
  - ✓以0开头，010，-065
  - ✓例如 int a = 010;

C语言整型常数的三种编写方法

- 十进制
  - ✓123，-2345
- 八进制
  - ✓以0开头，010，-065
  - ✓例如 int a = 010;
- 十六进制
  - ✓以0x开头，0x123，0XFFFFFF，-0x1abcd
  - ✓例如 int a = 0x123;

C语言整型常数的三种编写方法

- 十进制
  - ✓123，-2345
- 八进制
  - ✓以0开头，010，-065
  - ✓例如 int a = 010;
- 十六进制
  - ✓以0x开头，0x123，0XFFFFFF，-0x1abcd
  - ✓例如 int a = 0x123;

可以把十进制整数printf输出八进制/十六进制，反之亦然

利用转换说明符！！

转换说明符

C标准转换说明,表一

	C语言prinf转换说明及其打印结果
转换说明	输出
%a	浮点数,十六进制和p记数法(C99/C11)
%A	浮点数,十六进制和p记数法(C99/C11)
%c	单个字符
%d	有符号十进制
%e	浮点数, e记数法
%E	浮点数, e记数法
%f	浮点数, 十进制记数法
%g	根据值的不同,自动选择%f或%e, %e格式用于指数小于-4或大于或等于精度时
%G	根据值的不同,自动选择%f或%e, %e格式用于指数小于-4或大于或等于精度时
%i	有符号十进制(%d相同)
%o	无符号八进制
%p	指针
%s	字符串
%u	无符号十进制整数
%x	无符号十六进制整数
%X	无符号十六进制整数
%%	打印一个百分号

三种整型常数使用示例

```
 3  int main()
 4  {
 5      int a = 123;     //十进制123
 6      int b = 0123;    //八进制123
 7      int c = 0x123;   //十六进制123
 8
 9      printf("%d, %d, %d\n", a, b, c);  //打印成十进制形式
10      printf("%o, %o, %o\n", a, b, c);  //打印成八进制形式
11      printf("%x, %x, %x\n", a, b, c);  //打印成十六进制形式（字母小写）
12      printf("%X, %X, %X\n", a, b, c);  //打印成十六进制形式（字母大写）
13
14      return 0;
15  }
```

```
123, 83, 291
173, 123, 443
7b, 53, 123
7B, 53, 123
```

浮点型常数的表示

- 浮点表示法
  - ✓1.2，-3.56
- 科学计数法
  - ✓3.4e35，-2E-209
  - ✓转义字符为 %e 和 %E

浮点型常数使用示例

```
3  int main()
4  {
5      float a = 1.2;        //浮点表示法
6      float b = 3.4e-3;     //科学计数法
7
8      printf("%f, %f\n", a, b);  //打印成浮点形式
9      printf("%e, %e\n", a, b);  //打印成科学计数形式
10     printf("%E, %E\n", a, b);  //打印成科学计数形式
11
12     return 0;
13 }
```

```
1.200000, 0.003400
1.200000e+00, 3.400000e-03
1.200000E+00, 3.400000E-03
```

字符型常数的表示

- 字符表示
  - ✓ ‘a’ ， ‘A’

- 可以使用ASCII码对应的八进制或十六进制整数进行转义表示
  - ✓如字母B的ASCII码为66，则B的转义表示为\102或者\x42（102和42分别是66的八进制和十六进制值）

- 转换说明符为%c

字符型常数使用示例

```
 3  int main()
 4  {
 5      char ch1 = 'B';          //字符表示法
 6      char ch2 = '\102';       //转义表示法（八进制）
 7      char ch3 = '\x42';       //转义表示法（十六进制）
 8      char ch4 = 66;           //ASCII码表示
 9
10      printf("%c, %c, %c, %c\n", ch1, ch2, ch3, ch4);  //打印成字符表示
11
12      return 0;
13  }
```

```
B, B, B, B
```

字符型数据的输入和输出

- 输入
  - ✓scanf()，把stdin输入字符与转换说明符逐个匹配
  - ✓getchar()，每次从stdin输入一个字符

- 输出
  - ✓printf()，将输出数值按转换说明逐个打印到stdout
  - ✓putchar(char)，每次输出一个字符到stdout

- stdin：standard input
- stdout：standard output

字符型数据的输入和输出

- 输入
  - ✓scanf()，把stdin输入字符与转换说明符逐个匹配
  - ✓getchar()，每次从stdin输入一个字符
- 输出
  - ✓printf()，将输出数值按转换说明逐个打印到stdout
  - ✓putchar(char)，每次输出一个字符到stdout

对于字符处理很方便

- stdin：standard input
- stdout：standard output

字符型数据的输入和输出

```c
#include <stdio.h>

int main () {
    char c;

    printf("Enter character: ");
    c = getchar();

    printf("Character entered: ");
    putchar(c);

    return(0);
}
```

换
个

明
字

```c
#include <stdio.h>

int main () {
    char ch;

    for(ch = 'A' ; ch <= 'Z' ; ch++) {
        putchar(ch);
    }

    return(0);
}
```

getchar()的返回值类型

- 如果读取成功，则返回读取的字符
- 如果读取失败，则返回EOF（通常是遇到文件末尾或者error）

使用getchar循环读取，直至遇到'0'

```c
#include <stdio.h>

int main()
{
    char c;
    while((c = getchar()) != '0')
    {
        putchar(c);
        putchar('\n');
    }
    return 0;
}
```

getchar读取文件中的字符，直至文件末尾

- 从键盘输入字符时，可以使用ctrl+d模拟EOF

```
3  int main()
4  {
5      char ch;
6      while((ch = getchar()) != EOF)
7      {
8          putchar(ch);
9          putchar('\n');
10     }
11     return 0;
12 }
```

C语言中的表达式

- 算术表达式，常用于计算
- 赋值表达式，常用于给变量赋值
- 关系表达式，常用于比较，返回值为1（代表true）或0（代表false）
- 逻辑表达式，常用于逻辑运算，返回值为1（代表true）或0（代表false）
- 条件表达式，C语言唯一的三目运算符构成的表达式
- 逗号表达式，用逗号分隔的一组表达式，最后一个表达式的值作为整个表达式的值
- 位运算，用于二进制内存表示的运算
- 其他，sizeof()返回值类型为unsigned long

常用表达式中的运算符

运算符种类	运算符	结合方向	优先级
逻辑运算符	!	从右向左（右结合）	高
算术运算符	++ -- + - *（单目）		
	* / %（双目）	从左向右（左结合）	
	+ -（双目）		
关系运算符	< <= > >=		
	== ! =		
逻辑运算符	&&		
	||		
条件表达式	?:	从右向左（右结合）	
赋值运算符	= += -= *= /= %=		
逗号运算符	,	从左向右（左结合）	低

赋值表达式

- a = b，将b的值赋给a

运算符	等价运算
+=	a += b 等价于 a = a + b
-=	a -= b 等价于 a = a - b
*=	a *= b 等价于 a = a * b
/=	a /= b 等价于 a = a / b
%=	a %= b 等价于 a = a % b

关系表达式

- 表示运算符两边的关系，一般用作分支和循环的条件语句

运算符	表示关系
a < b	a小于b
a <= b	a小于等于b
a > b	a大于b
a >= b	a大于等于b
a == b	a等于b
a != b	a不等于b

逻辑表达式

- 表示运算符两边的关系，一般用作分支和循环的条件语句

运算符	名称	简明含义
!	逻辑非	!a表示a取反，即非0变0，0变非0
&&	逻辑与	a && b，a并且b
||	逻辑或	a || b，a或者b

- 实际过程：
  - ✓a && b是对a和b的值做“与”运算，即a和b都是非0，则a && b的值为非0，否则a && b的值就是0；
  - ✓同理，a || b是对a和b的值做“或”运算，即a和b都是0，则a || b的值为0，否则a || b的值就是非0；

逻辑表达式

- 表示运算符两边的关系，一般用作分支和循环的条件语句

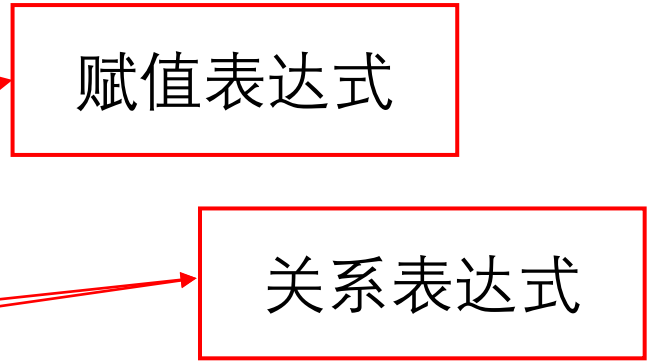
运算符	名称	简明含义
!	逻辑非	!a表示a取反，即非0变0，0变非0
&&	逻辑与	a && b，a并且b
||	逻辑或	a || b，a或者b

- 实际过程：
  - ✓a && b是对a和b的值做“与”运算，即a和b都是非0，则a && b的值为非0，否则a && b的值就是0；
  - ✓同理，a || b是对a和b的值做“或”运算，即a和b都是0，则a || b的值为0，否

可以使用%d格式转换符打印关系、逻辑表达式的值

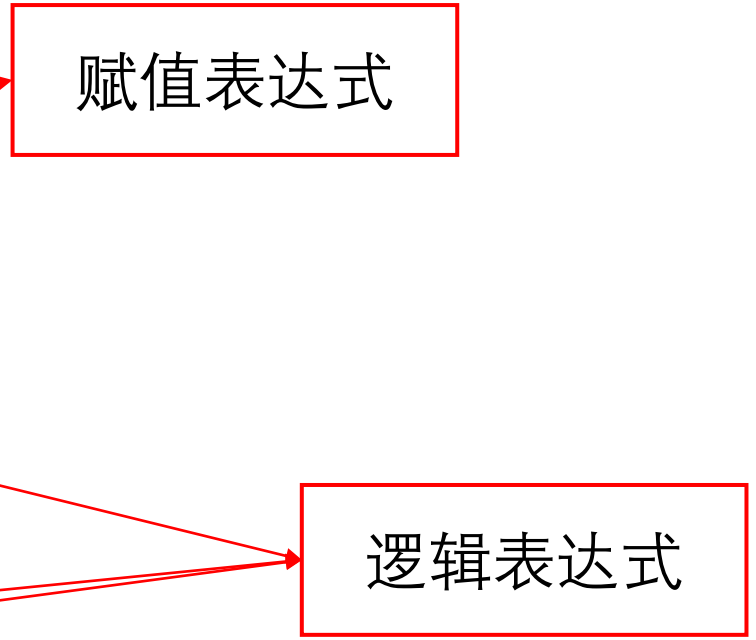
表达式示例

```
 3  int main()
 4  {
 5      int x = 1;
 6      int y = 2;
 7
 8      if (x > 1 && y == 2)
 9          printf("yes!");
10      else
11          printf("no!");
12
13      if ( !x || y == 2)
14          printf("yes!");
15      else
16          printf("no!");
17
18      return 0;
19  }
```



表达式示例

```
3  int main()
4  {
5      int x = 1;
6      int y = 2;
7
8      if (x > 1 && y == 2)
9          printf("yes!");
10     else
11         printf("no!");
12
13     if ( !x || y == 2)
14         printf("yes!");
15     else
16         printf("no!");
17
18     return 0;
19 }
```



条件表达式：唯一的三目运算符

```
 3  int main()
 4  {
 5      int a = 3;
 6      int b = 5;
 7      int c = a > b ? a : b; //找出a和b中较大的数
 8      printf("%d\n", c);
 9
10      return 0;
11  }
```

条件表达式：唯一的三目运算符

a > b 吗？

```
3  int main()
4  {
5      int a = 3;
6      int b = 5;
7      int c = a > b ? a : b; //找出a和b中较大的数
8      printf("%d\n", c);
9
10     return 0;
11 }
```

条件表达式：唯一的三目运算符

若是，则返回a（执行 ：前的表达式）

a > b 吗？

```
 3  int main()
 4  {
 5      int a = 3;
 6      int b = 5;
 7      int c = a > b ? a : b; //找出a和b中较大的数
 8      printf("%d\n", c);
 9
10      return 0;
11  }
```

条件表达式：唯一的三目运算符

若是，则返回a（执行 ：前的表达式）

a > b 吗？

```
3  int main()
4  {
5      int a = 3;
6      int b = 5;
7      int c = a > b ? a : b; //找出a和b中较大的数
8      printf("%d\n", c);
9
10     return 0;
11 }
```

否则返回b（执行：后的表达式）

条件表达式：唯一的三目运算符

```
 3  int main()
 4  {
 5      int a = 3;
 6      int b = 5;
 7      a > b ? printf("a is larger.") : printf("b is larger.");
 8
 9      return 0;
10  }
```

sizeof的用法

- 返回类型、常量、变量占据的内存字节数
- 返回值类型为unsigned long

```
 3  int main()
 4  {
 5      char ch = 'A';
 6      double a = 1.2;
 7      printf("%lu, %lu, %lu, %lu\n", sizeof(ch), sizeof('A'), sizeof(int), sizeof(a));
 8
 9      return 0;
10  }
```

3、数组

C语言数据类型

- 基本数据类型
  - ✓整型、字符型、浮点型（单\双精度）
- 构造数据类型
  - ✓数组、结构（struct）、联合（union）、枚举（enum）
- 指针类型
- 空类型

数组

- 一组类型相同的、在内存中连续存放的数据的集合

- 数组的定义
  - ✓一维数组：类型名 数组名 [数组长度]，如int a[10];
  - ✓二维数组：类型名 数组名 [第1维长度][第2维长度]，如int a[2][3];
  - ✓N维数组：类型名 数组名 [第1维长度][第2维长度]…[第N维长度]

- 数组的长度可以使用常量或者变量（C99以后，但不建议）
  - ✓int a[10] 或者 int a[n]

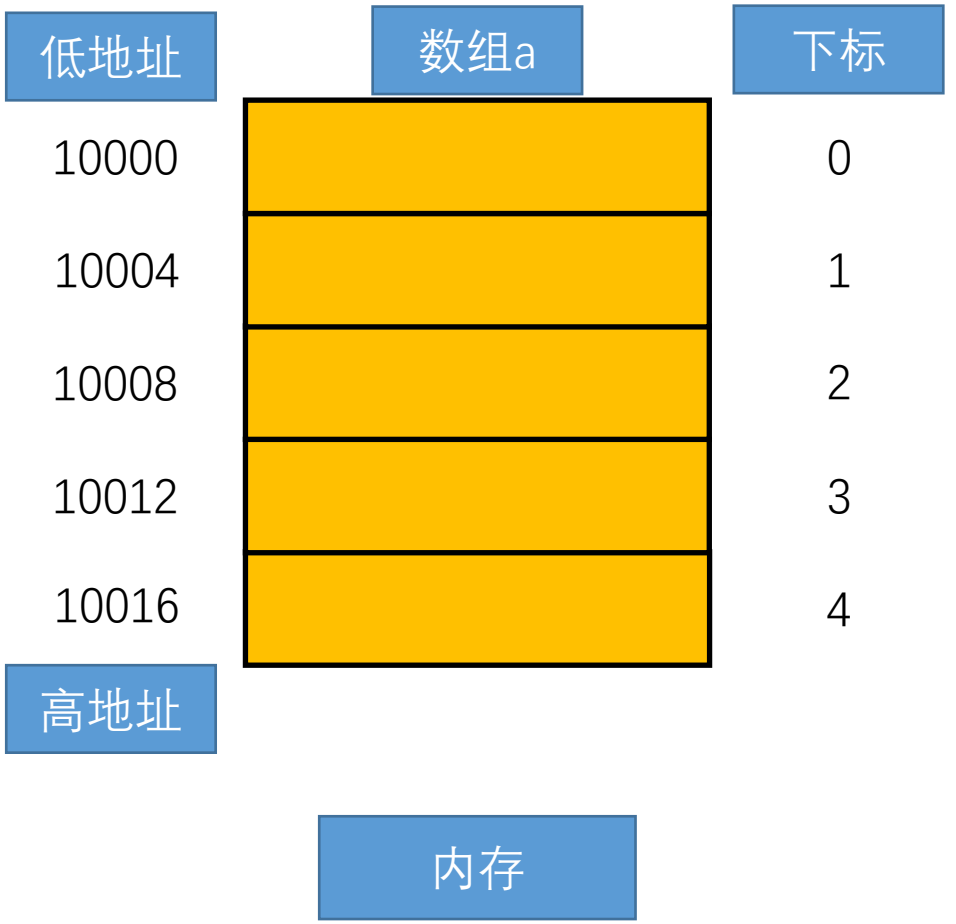
一维数组的定义和引用

- 定义：int a[5];
  - ✓定义了长度为5的一维数组

- 引用元素：a[0], a[1], …, a[4]
  - ✓a[-1], a[5]?
  - ✓注意不要越界！！
  - ✓如何引用整个数组？
  - ✓a

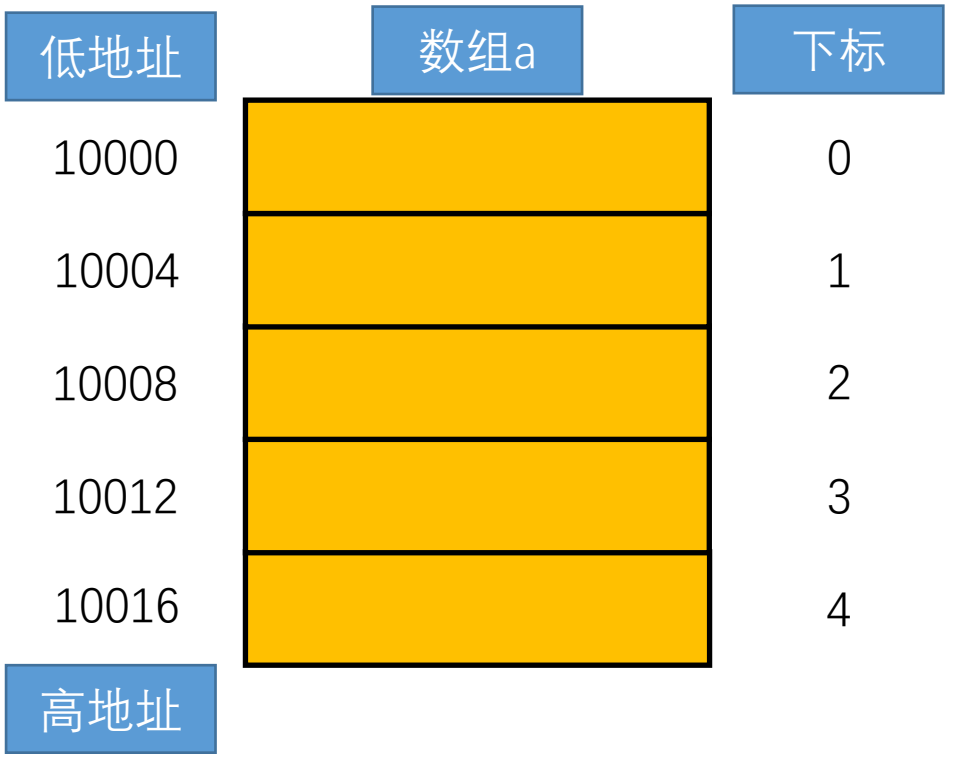
一维数组的定义和引用

- 定义：int a[5];
  - ✓定义了长度为5的一维数组
- 引用元素：a[0], a[1], …, a[4]
  - ✓a[-1], a[5]?
  - ✓注意不要越界！！
  - ✓如何引用整个数组？
  - ✓a
- 数组在内存中存放
  - ✓从a[0]开始，依次存放（假设从低地址到高地址）
  - ✓为什么地址之间差是4？



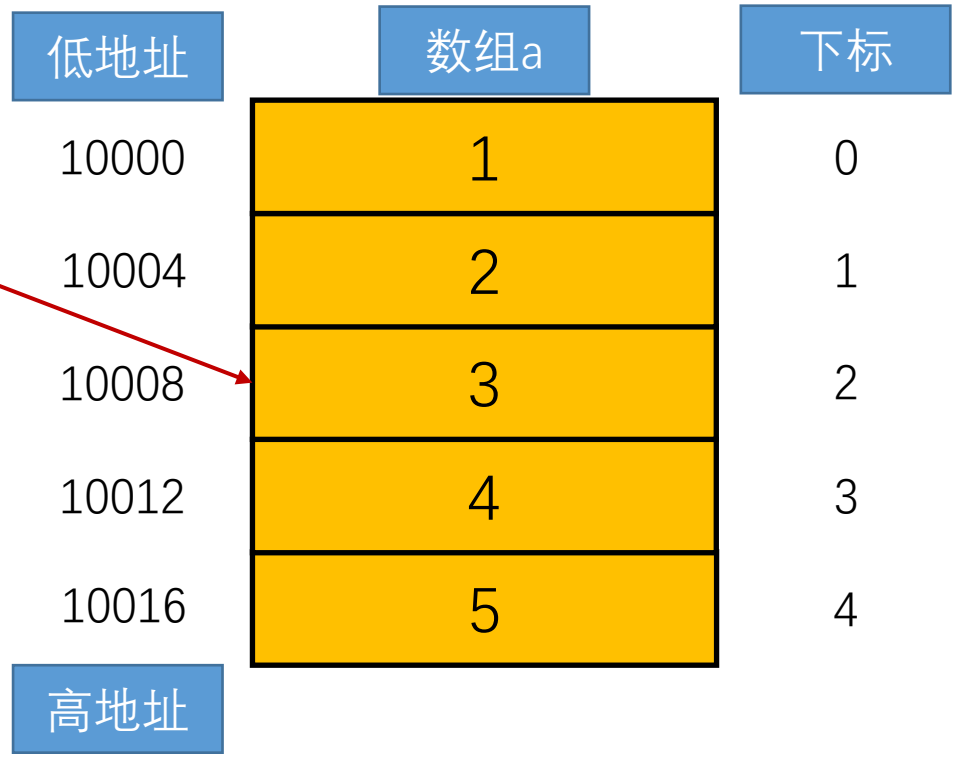
一维数组的初始化

- 类型名 数组名[数组长度]={初值表};



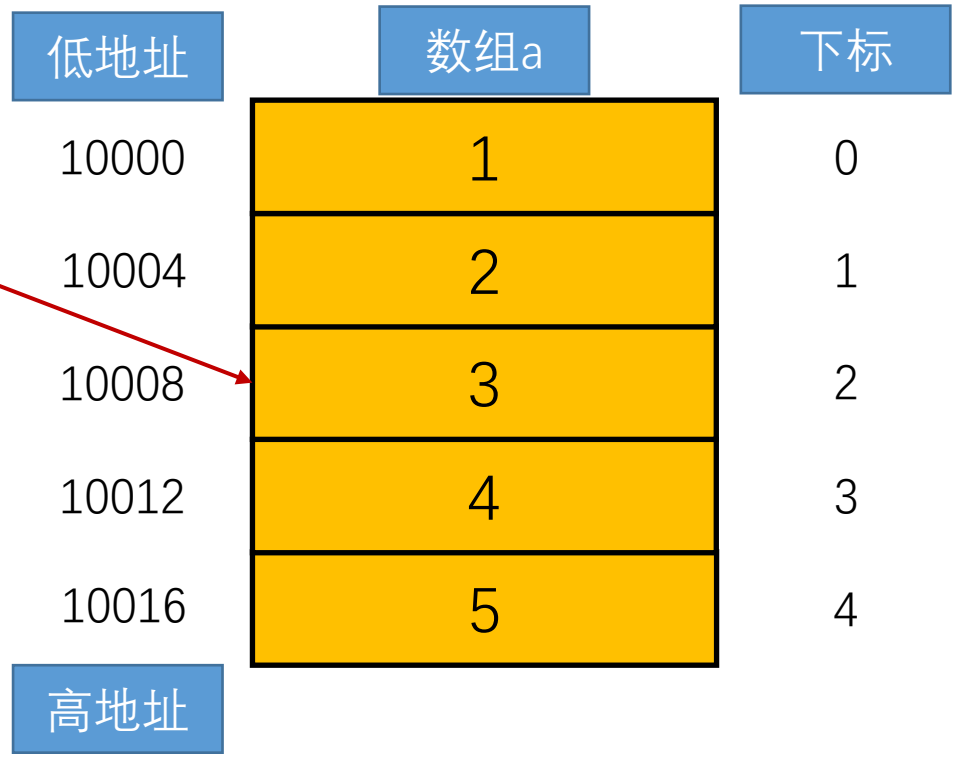
一维数组的初始化

- 类型名 数组名[数组长度]={初值表};
  - ✓int a[5] = {1, 2, 3, 4, 5};



一维数组的初始化

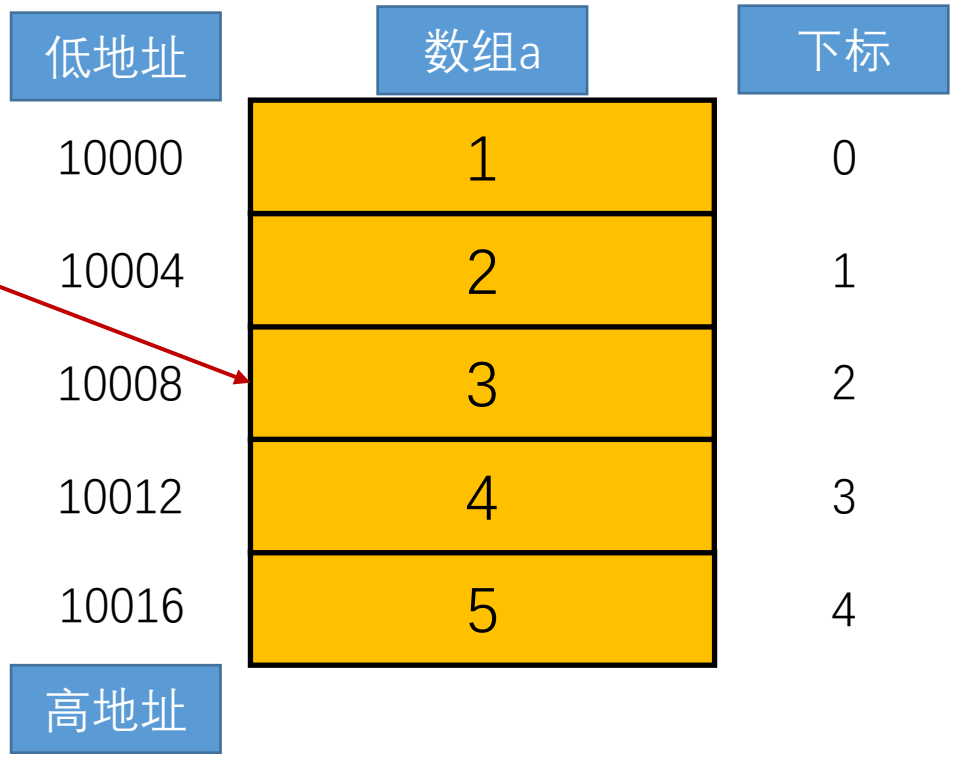
- 类型名 数组名[数组长度]={初值表};
  - ✓int a[5] = {1, 2, 3, 4, 5};
  - ✓int a[5] = {1, 2, 3};



一维数组的初始化

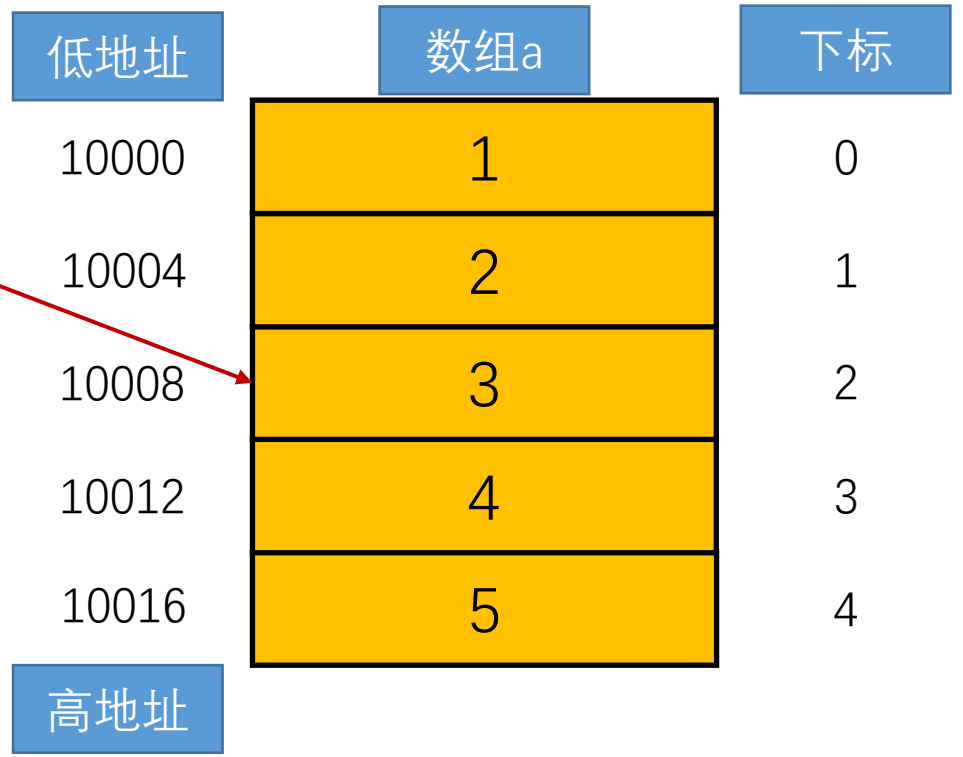
- 类型名 数组名[数组长度]={初值表};
  - ✓int a[5] = {1, 2, 3, 4, 5};
  - ✓int a[5] = {1, 2, 3};

- 未初始化的数组元素
  - ✓全局数组/静态数组：0
  - ✓局部数组：不确定
  - ✓（但现在好像都会初始成0）



一维数组的初始化

- 类型名 数组名[数组长度]={初值表};
  - ✓int a[5] = {1, 2, 3, 4, 5};
  - ✓int a[5] = {1, 2, 3};



- 未初始化的数组元素
  - ✓全局数组/静态数组：0
  - ✓局部数组：不确定
  - ✓（但现在好像都会初始成0）

- 如果初值表元素个数等于想定义的数组的长度，则数组长度可以省略
  - ✓不推荐！！（如果这样定义了，如何知道数组的长度？使用sizeof！）

一维数组的长度

```c
#include<stdio.h>

int main()
{
    int a[] = {1,2,3};
    printf("%lu\n", sizeof(a) / sizeof(int));

    return 0;
}
```

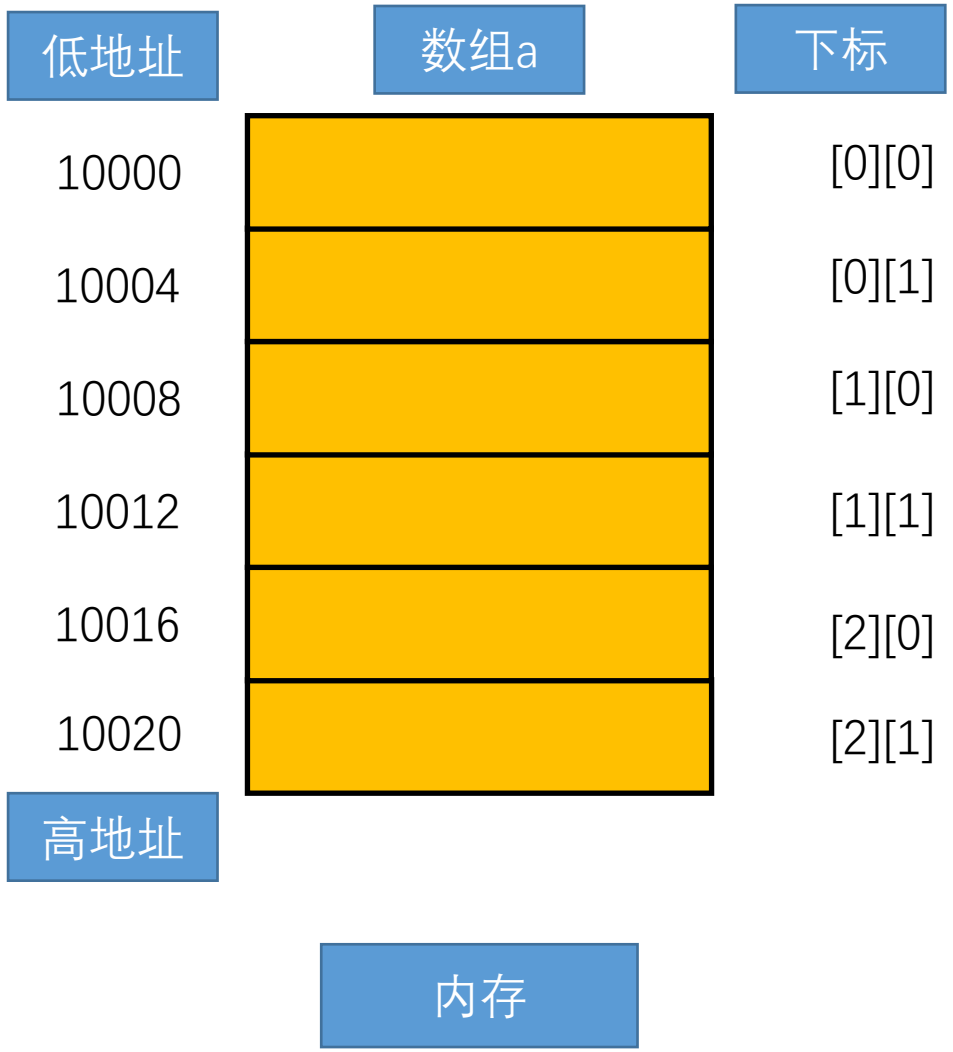
二维数组的定义和引用

- 定义：int a[3][2];
  - ✓想象成3行2列的矩阵
  - ✓第一维代表行数，第二维代表列数

- 引用元素：a[0][0], a[2][1]
  - ✓a[0][-1], a[2][2], a[3][1]?
  - ✓注意不要越界！！

二维数组的定义和引用

- 定义：int a[3][2];
  - ✓想象成3行2列的矩阵
  - ✓第一维代表行数，第二维代表列数
- 引用元素：a[0][0], a[2][1]
  - ✓a[0][-1], a[2][2], a[3][1]?
  - ✓注意不要越界！！
- 数组在内存中存放
  - ✓从a[0][0]开始，依次存放（假设从低地址到高地址）



二维数组的初始化

- 分行赋初值
  - ✓int a[3][2] = {{1, 2}, {3, 4}, {5, 6}};
  - ✓int a[3][2] = {{1, 2}, {}, {5}};
- 顺序赋初值
  - ✓int a[3][2] = {1, 2, 3, 4, 5, 6};
  - ✓int a[3][2] = {1, 2, 5};

- 数组行数明确时，第一维长度可以省略
  - ✓不推荐！！

向量点乘示例

- 用一维数组a和一维数组b代表两个向量，进行向量点乘运算

```c
#include<stdio.h>

int main()
{
    int a[3] = {1,2,3};
    int b[3] = {2,3,4};
    int sum = 0;
    for(int i = 0; i < 3; i++)
    {
        sum += a[i] * b[i];
    }
    printf("%d\n", sum);
    return 0;
}
```

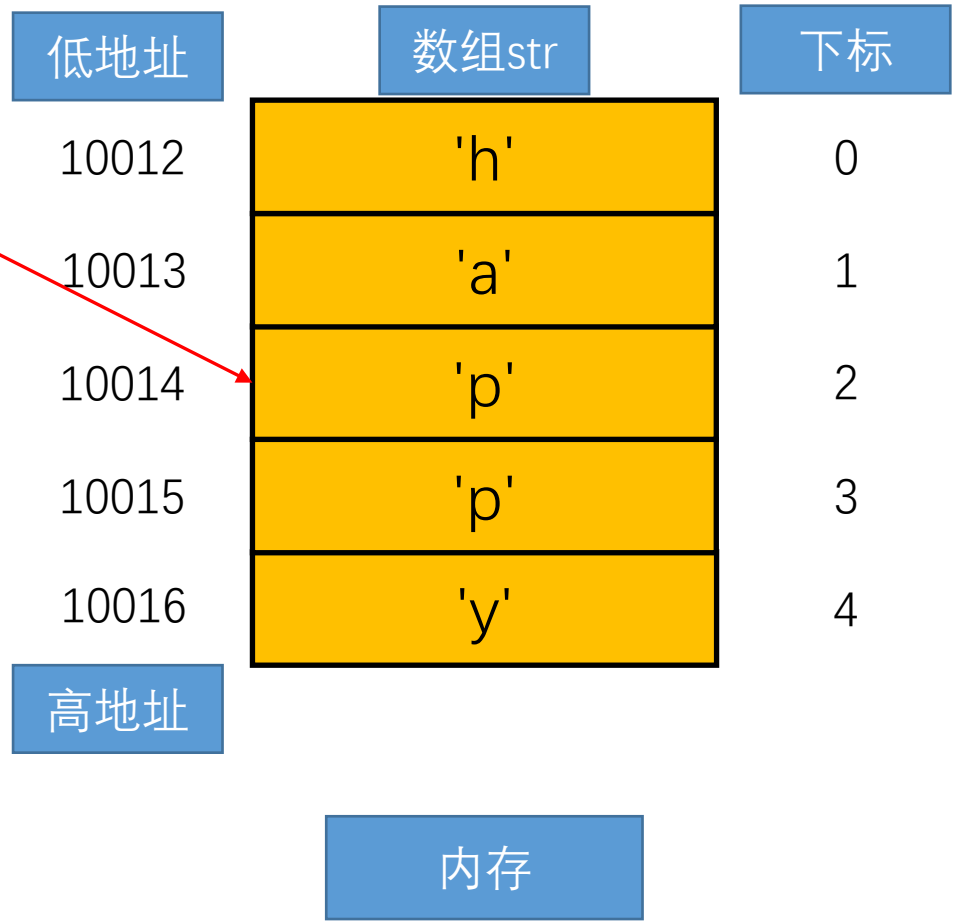
矩阵乘法示例

- 用二维数组a和二维数组b代表两个矩阵，进行矩阵乘法运算

```c
#include <stdio.h>

int main()
{
    int a[2][2] = {{0,1}, {0,2}};   /*  0, 1
                                        0, 2  */

    int b[2][2] = {{1,2}, {3,4}};   /*  1, 2
                                        3, 4  */

    for(int i = 0; i < 2; i++)  //数组a每一行
    {
        for(int j = 0; j < 2; j++)  //数组b每一列
        {
            int res = 0;
            for(int k = 0; k < 2; k++)  //数组a的i行和数组b的j列，逐元素相乘再求和
            {
                res += a[i][k] * b[k][j];
            }
            printf("%d", res);
            if(j == 1)
                printf("\n");
            else
                printf(" ");
        }
    }
    return 0;
}
```

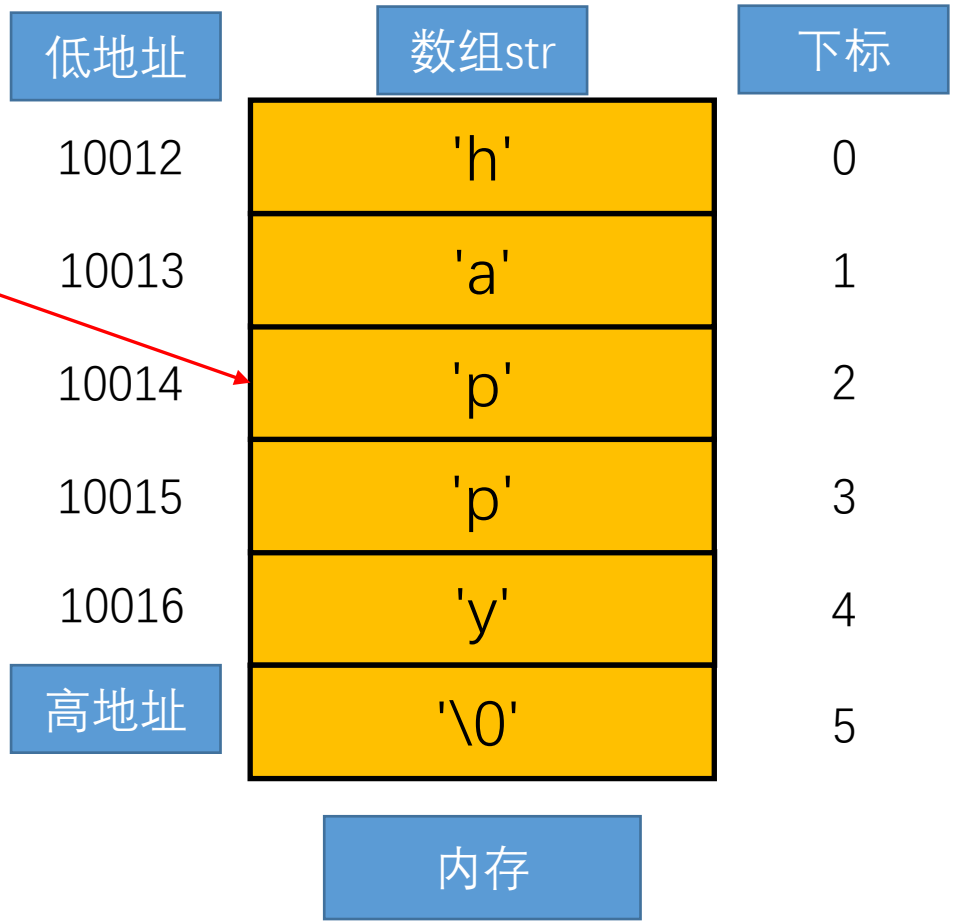
字符数组与字符串

- 字符数组
  - ✓ char str[5] = {'h', 'a', 'p', 'p', 'y'};



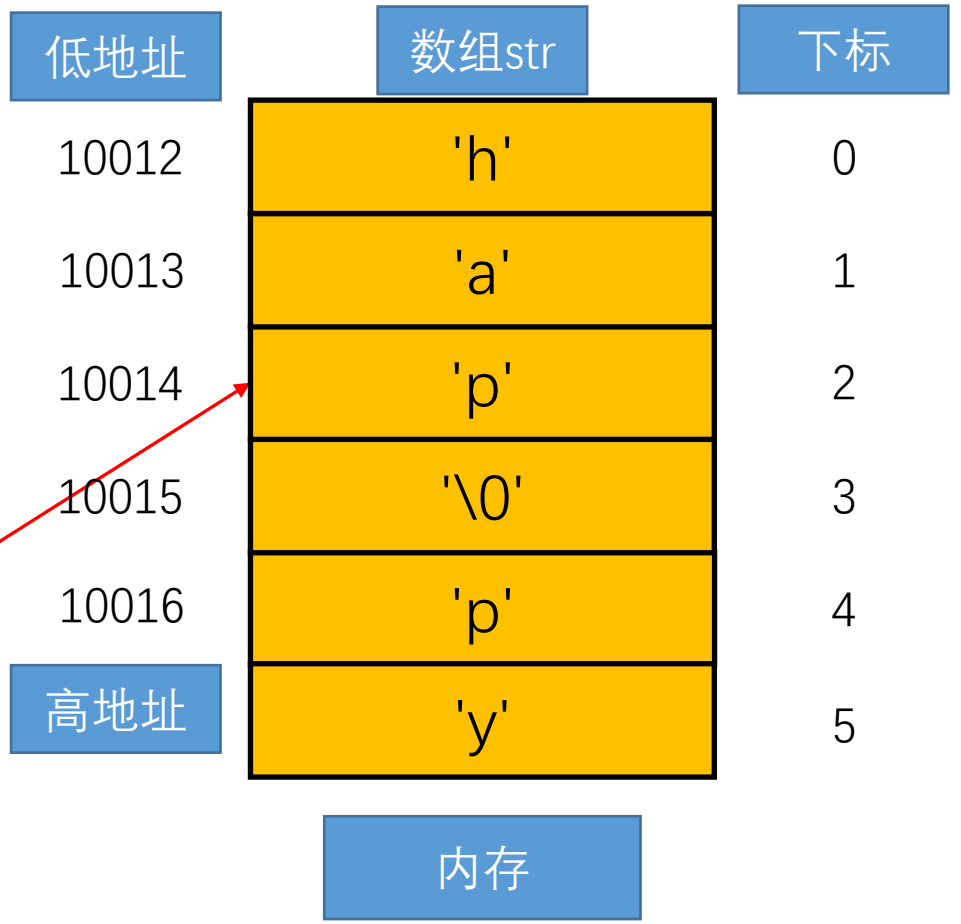
字符数组与字符串

- 字符数组
  - ✓char str[5] = {'h', 'a', 'p', 'p', 'y'};
  - ✓char str[6] = {'h', 'a', 'p', 'p', 'y'};
  - ✓如果某元素未赋值，则默认值为0，等价于'\0'



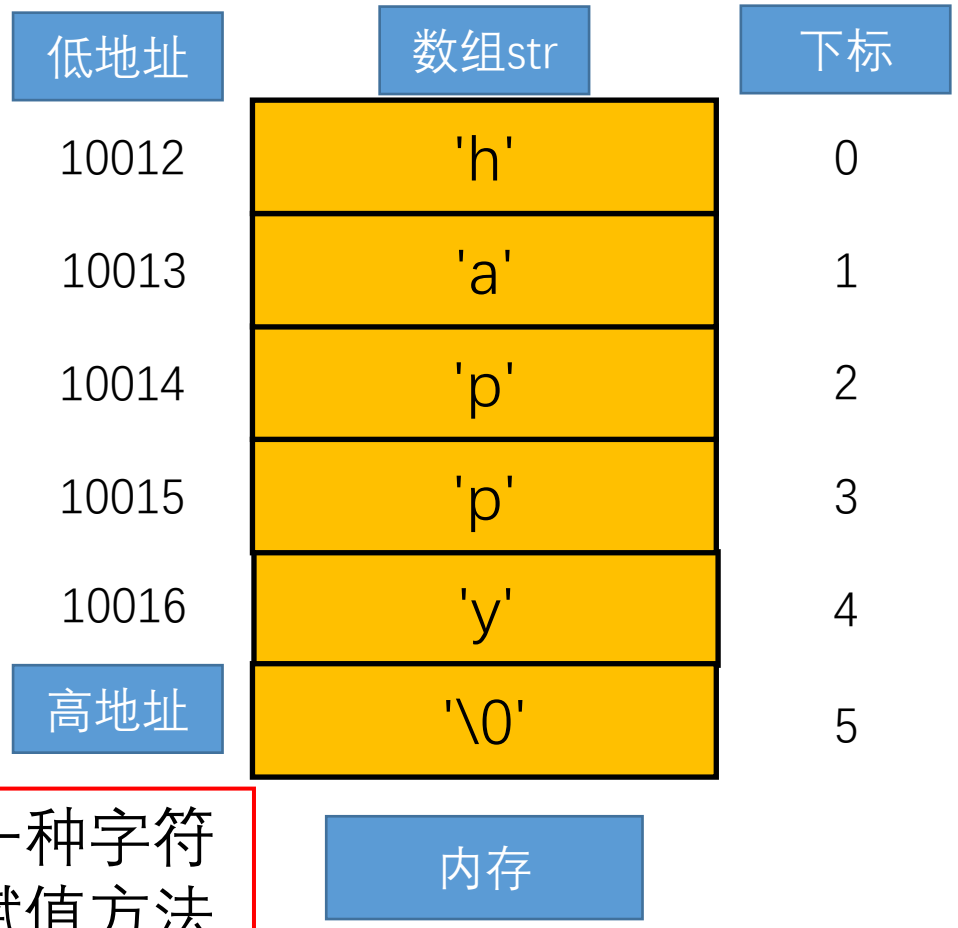
字符数组与字符串

- 字符数组
  - ✓char str[5] = {'h', 'a', 'p', 'p', 'y'};
  - ✓char str[6] = {'h', 'a', 'p', 'p', 'y'};
  - ✓如果某元素未赋值，则默认值为0，等价于'\0'
- 字符串
  - ✓char str[6] = {'h', 'a', 'p', 'p', 'y', '\0'};
  - ✓char str[6] = {'h', 'a', 'p', '\0', 'p', 'y'};
  - ✓字符数组+结束标志'\0'，形成字符串
  - ✓结束标志之前为字符串的有效字符！



字符数组与字符串

- 字符数组
  - ✓char str[5] = {'h', 'a', 'p', 'p', 'y'};
  - ✓char str[6] = {'h', 'a', 'p', 'p', 'y'};
  - ✓如果某元素未赋值，则默认值为0，等价于'\0'
- 字符串
  - ✓char str[6] = {'h', 'a', 'p', 'p', 'y', '\0'};
  - ✓char str[6] = {'h', 'a', 'p', '\0', 'p', 'y'};
  - ✓字符数组+结束标志'\0'，形成字符串
  - ✓结束标志之前为字符串的有效字符！
  - ✓char str[6] = "happy";
  - ✓注意双引号和单引号的使用区别！！



另一种字符串赋值方法

字符数组和字符串示例

```
 3  int main()
 4  {
 5      char str[6] = {'h', 'a', 'p', '\0', 'p', 'y'};
 6      for(int i = 0; i < 6; i++)
 7          printf("%c", str[i]);
 8      printf("\n");
 9      printf("%s\n", str);
10
11      return 0;
12  }
```

字符数组和字符串示例

```
 3  int main()
 4  {
 5      char str[6] = {'h', 'a', 'p', '\0', 'p', 'y'};
 6      for(int i = 0; i < 6; i++)
 7          printf("%c", str[i]);
 8      printf("\n");
 9      printf("%s\n", str);
10
11      return 0;
12  }
```

打印字符

打印字符串

```
happy
hap
```

小结

- 程序运行的内存分布
  - ✓程序区
  - ✓数据区：静态存储区和动态存储区
- 各数据类型的常数表示和打印方法
- 字符的输入输出
- 常用表达式及其返回值
- 数组的基本概念

L06：指针