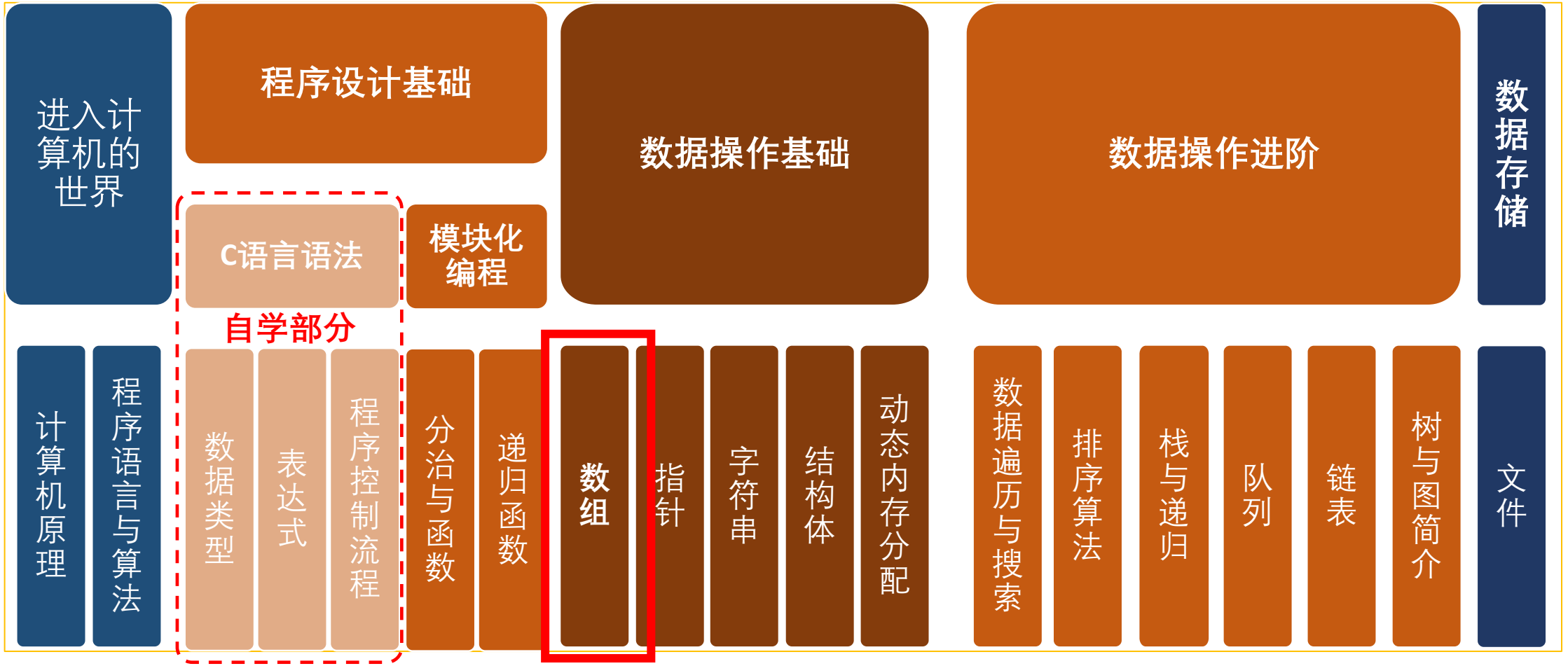
计算思维与实践

哈尔滨工业大学（深圳）
计算机科学与技术学院
大数据技术中心
张保权

课程内容安排



程序设计思想与数据操作方法融会贯通，内容由浅入深

第六讲 数组——学习内容

- 为什么使用数组(Array)?
- 数组的定义和初始化
- 向函数传递一维数组
- 向函数传递二维数组
- 数组综合应用案例

第六讲 数组——学习内容

- 为什么使用数组(Array)?
- 数组的定义和初始化
- 向函数传递一维数组
- 向函数传递二维数组
- 数组综合应用案例

6.1 为什么使用数组(Array)?

- 【例】要读入并存储10人的成绩，然后求平均成绩

- 需定义10个不同名的整型变量，需使用多个**scanf()**

```
int score1, score2, … score10;
  scanf("%d", &score1);
  scanf("%d", &score2);
  ......
```

- 而数组仅用一个**scanf()**并利用循环语句读取

```
int score[10],i;
  for (i=0; i<10; i++)
  {
       scanf("%d", &score[i]);
  }
```

第六讲 数组——学习内容

- 为什么使用数组(Array)?
- **数组的定义和初始化**
- 向函数传递一维数组
- 向函数传递二维数组
- 数组综合应用案例

6.2 数组的定义和初始化——一维数组的定义和初始化

- 一维数组的定义

```
int a[10];
```

数据类型

代表元素个数

下标从0开始
数组名a代表首地址

- 定义一个有10个int型元素的一维数组
  - 系统分配连续的10个int型存储空间给此数组
- 为什么数组下标从0开始？
  - 使编译器简化，且运算速度少量提高
- 如果希望下标从1到10而非从0到9，怎么办？

6.2 数组的定义和初始化——一维数组的定义和初始化

- 一维数组的定义

```
int a[10];
```

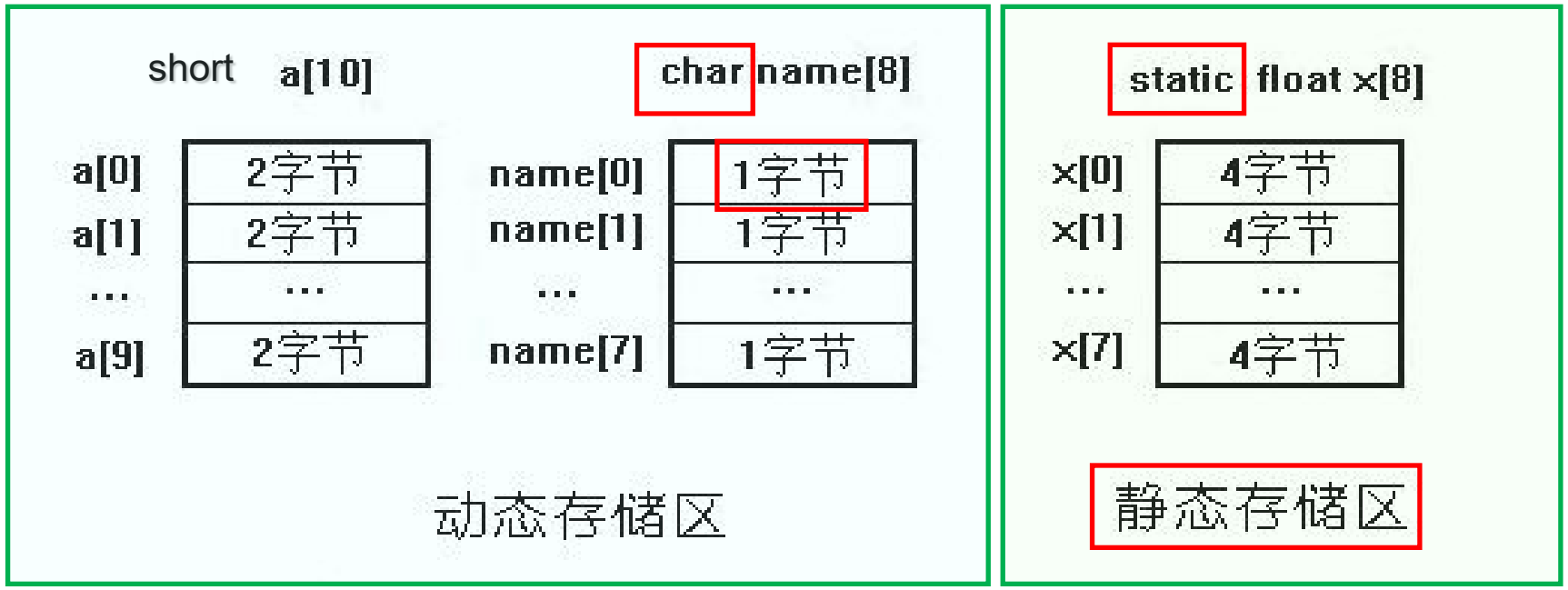
- 数组大小必须是值为正的常量，不能为变量，一旦定义，不能改变大小
- 大小最好用宏定义，以适应未来的变化

```
#define N 10

int a[N];
```

6.2 数组的定义和初始化——一维数组的定义和初始化

- 根据数组的数据类型，为每一元素安排相应字节数的存储单元
- 根据数组的存储类型，将其安排在内存的动态、静态存储区或寄存器区



6.2 数组的定义和初始化——一维数组的定义和初始化


```
int a[10];
```

引用时下标允许是
int型变量或表达式

- 一维数组的引用

  数组名[下标]

- 允许快速随机访问
  - 允许使用a[i]这样的形式访问每个元素
  - 可以像使用普通变量一样使用a[0],a[1],...,a[9]

6.2 数组的定义和初始化——一维数组的定义和初始化

- 未初始化的数组元素值是什么？
  - 静态数组和全局数组自动初始化为0值
  - 否则，是随机数
- 一维数组的初始化

```
int a[5] = { 12, 34, 56, 78, 9 };
int a[5] = { 0 };
int a[] = { 11, 22, 33, 44, 55 };
```

6.2 数组的定义和初始化——一维数组的定义和初始化

如何使两个数组的值相等？

```
int a[4] = {1,2,3,4};
int b[4];
b = a;
```

数组名表示数组的首地址，
不代表整个数组元素值！

解决方法

- 方法1：逐个元素赋值

```
b[0]=a[0];
b[1]=a[1];
b[2]=a[2];
b[3]=a[3];
```

- 方法2：通过循环语句赋值

```
int i;
for (i=0;i<4;i++)
{
    b[i] = a[i];
}
```

6.2 数组的定义和初始化——一维数组的定义和初始化

- 更高效的数组初始化方法

```
memset(a, 0, sizeof(a));
```

- 用sizeof(a)来获得数组a所占的内存字节数
- 更高效的数组赋值方法

```
memcpy(b, a, sizeof(a));
```

- 需要包含相应的头文件：

```
#include <string.h>
```

6.2 数组的定义和初始化——一维数组的定义和初始化

- 【例】编程实现显示用户输入的月份（不考虑闰年）拥有的天数

```
1   #include <stdio.h>
2   #define  MONTHS 12
3   int main()
4   {
5       int days[MONTHS] = {31,28,31,30,31,30,31,31,30,31,30,31};
6       int month;
7       do{
8          printf("Input a month:");
9          scanf("%d", &month);
10      }while(month < 1 || month > 12); /* 处理不合法数据的输入 */
11      printf("The number of days is %d\n", days[month-1]);
12      return 0;
13  }
```

为什么要处理非法的数据输入？

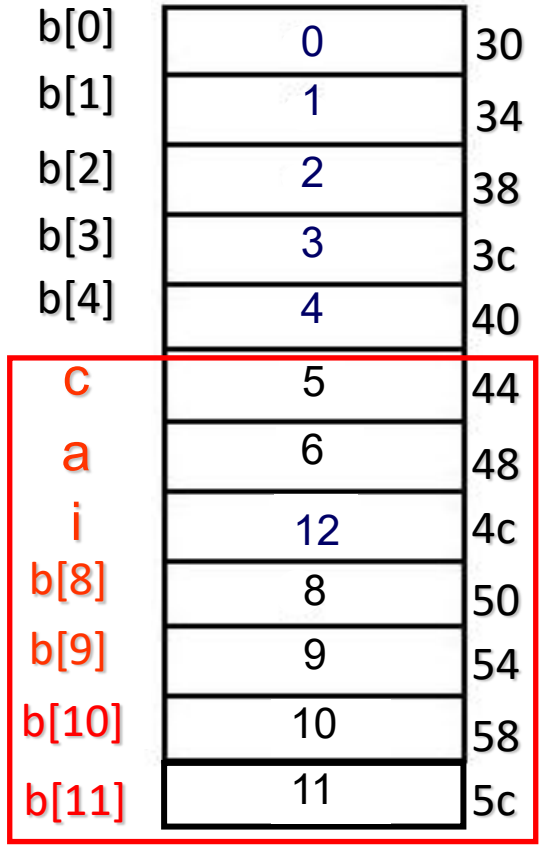
6.2 数组的定义和初始化——一维数组的定义和初始化

- 访问数组元素时，下标越界是大忌！
  - 编译程序不检查下标越界，导致运行时错误
  - 下标越界，将访问数组以外的空间
  - 那里的数据是未知的，不受我们掌控，可能带来严重后果
  - 后果有多严重呢？

6.2 数组的定义和初始化——一维数组的定义和初始化

【例】当下标值小于0或超过数组长度时会出现什么情况？

```
1   #include <stdio.h>
2   int main()
3   {
4       int a = 1, c = 2, b[5], i;
5       printf("%p, %p, %p\n", b, &a, &c);
6       for (i=0; i<=11; i++)
7       {
8           b[i] = i;
9           printf("%d  ", b[i]);
10      }
11      printf("\na=%d, c=%d\n", a, c);
12      return 0;
13  }
```

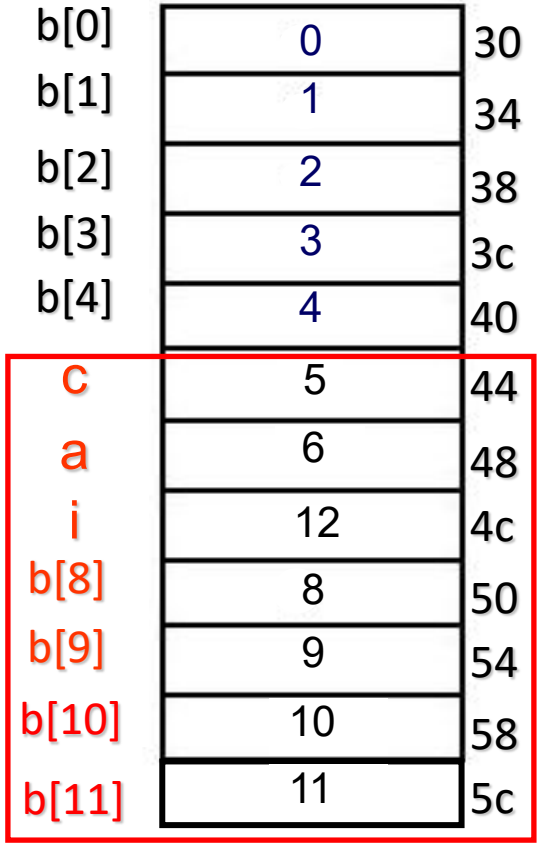


变量c和a的值，因数组下标越界而被悄悄破坏了

6.2 数组的定义和初始化——一维数组的定义和初始化

【例】当下标值小于0或超过数组长度时会出现什么情况？

```
1   #include <stdio.h>
2   int main()
3   {
4       int a = 1, c = 2, b[5], i;
```



输出的地址值是系统相关的

6.2 数组的定义和初始化—二维数组的定义和初始化

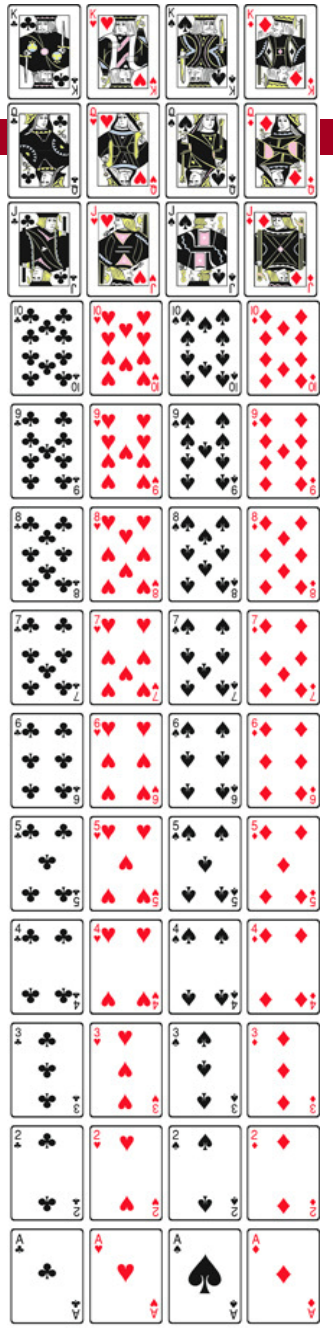
- 一维数组： `int a[5];`
  - 用一个下标确定各元素在数组中的顺序
  - 可用排列成一行的元素来表示

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

- 二维数组：`int b[2][3];`
  - 用两个下标确定各元素在数组中的顺序
  - 用排列成i行，j列的元素来表示
  - 逻辑结构，非物理结构
  - 内存中线性保存

b[0][0]	b[0][1]	b[0][2]
b[1][0]	b[1][1]	b[1][2]

- n维数组：`int c[3][2][4];`
  - 用n个下标来确定各元素在数组中的顺序

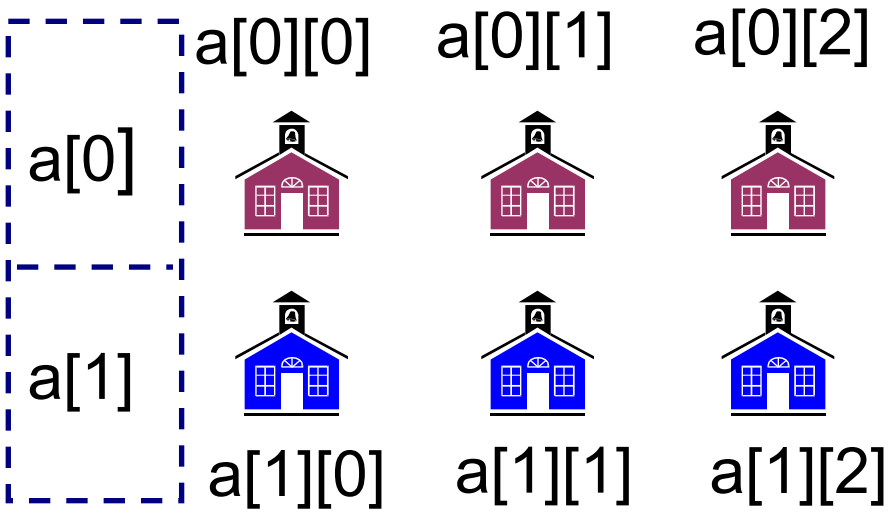


6.2 数组的定义和初始化—二维数组的定义和初始化

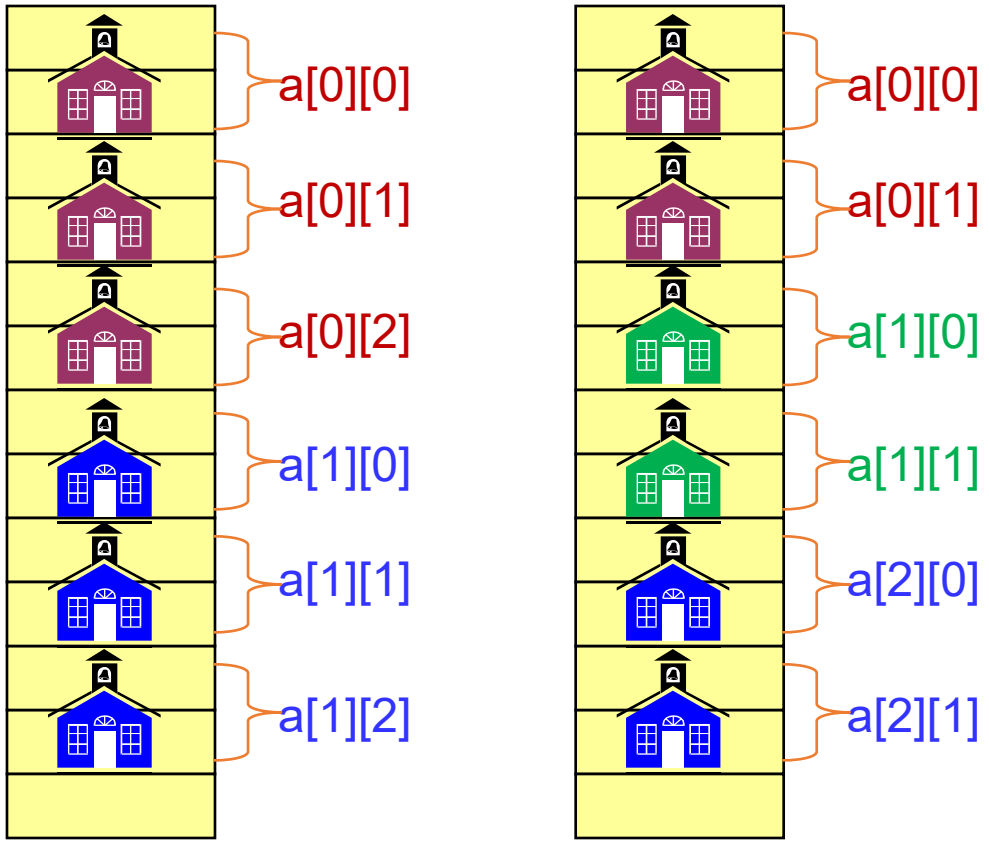
存放顺序：按行存放，线性存储

先顺序存放第0行，再存放第1行

```
short a[2][3];
```



若short a[3][2];则…



已知每行列数才能正确读出数组元素

所以初始化时，第二维长度不能省略

6.2 数组的定义和初始化—二维数组的定义和初始化

【例】以下程序的运行结果是什么？

```
int main()
{
    int a[][3]={{1,2,3},{4,5},{6},{0}};
    printf("%d,%d,%d\n",a[1][1],a[2][1],a[3][1]);
    return 0;
}
```

1	2	3
4	5	0
6	0	0
0	0	0

结果：5，0，0

【例】若`int a[ ][3]={1, 2, 3, 4, 5, 6, 7};`则a数组的第一维大小是多少？

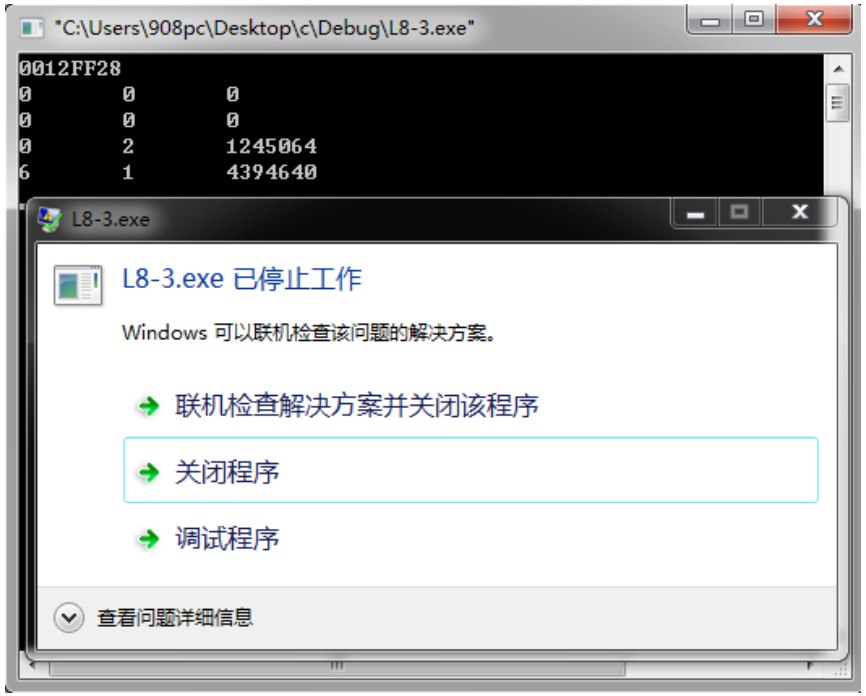
1	2	3
4	5	6
7	0	0

6.2 数组的定义和初始化—二维数组的定义和初始化

二维数组下标越界

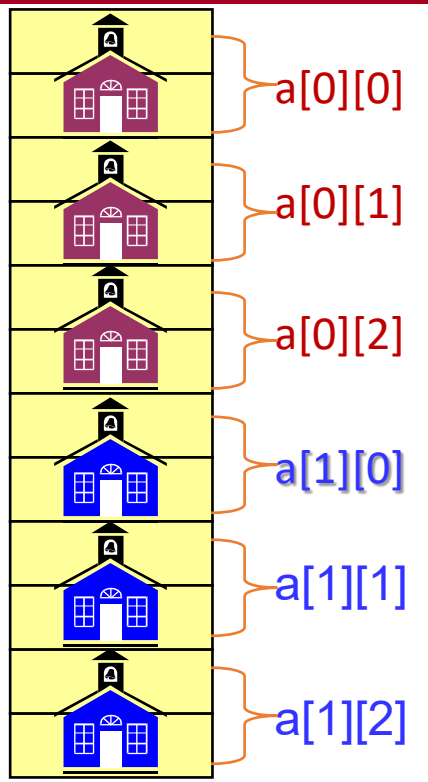
```
#include <stdio.h>
int main()
{
    int i, j;
    int a[2][3] = {0};
    printf("%p\n", a);
    a[3][0] = 6;
    for (i=0; i<4; i++)
    {
        for (j=0; j<3; j++)
        {
            printf("%d\t", a[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

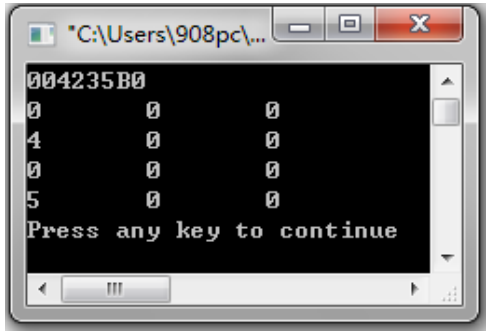
a[0][0]	a[0][1]	a[0][2]
a[1][0]	a[1][1]	a[1][2]
a[2][0]	a[1][1]	a[1][2]
a[3][0]	a[1][1]	a[1][2]



6.2 数组的定义和初始化—二维数组的定义和初始化


```
#include <stdio.h>
int main()
{
    int i, j;
    int a[2][3] = {0};
    printf("%p\n", a);
    a[1][0] = 4;
    for (i=0; i<2; i++)
    {
        for (j=0; j<3; j++)
        {
            printf("%d\t", a[i][j]);
        }
        printf("\n");
    }
    a[0][3] = 5;
    for (i=0; i<2; i++)
    {
        for (j=0; j<3; j++)
        {
            printf("%d\t", a[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```





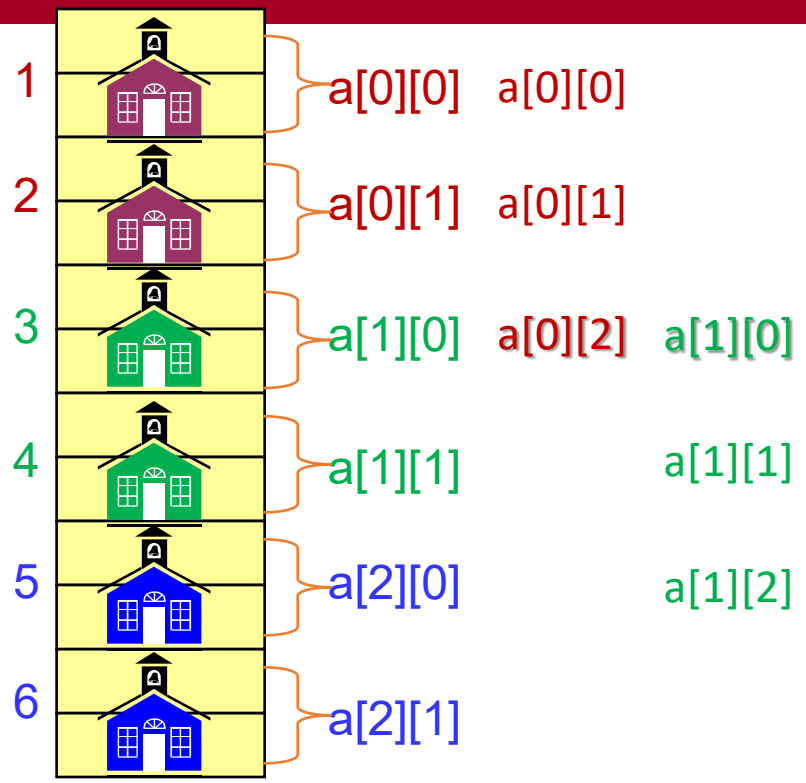
a[0][3]和a[1][0]指的是同一元素，不检查下标越界，a[0][3]的写法也合法，但隐患严重

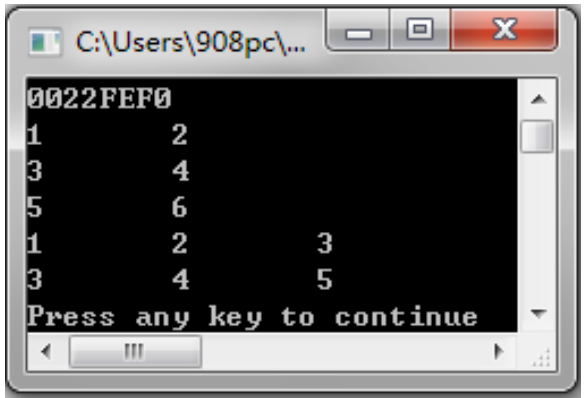
6.2 数组的定义和初始化—二维数组的定义和初始化


```
#include <stdio.h>
int main()
{
    int i, j;
    int a[3][2] = {1,2,3,4,5,6};
    printf("%p\n", a);

    for (i=0; i<3; i++)
    {
        for (j=0; j<2; j++)
        {
            printf("%d\t", a[i][j]);
        }
        printf("\n");
    }

    for (i=0; i<2; i++)
    {
        for (j=0; j<3; j++)
        {
            printf("%d\t", a[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```



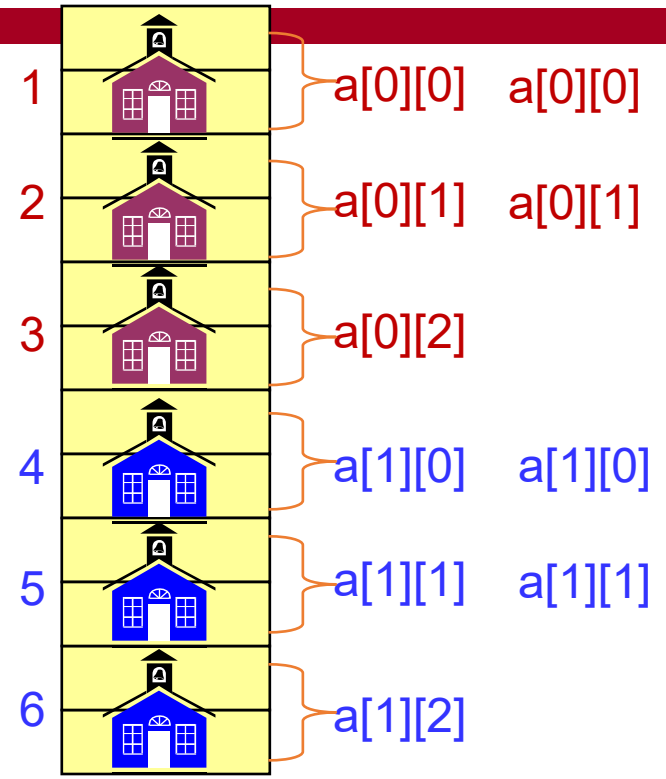


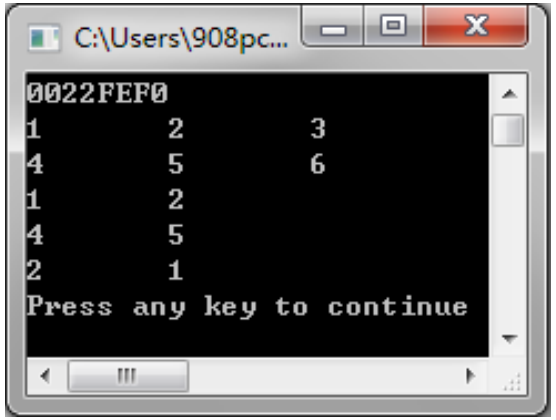
6.2 数组的定义和初始化—二维数组的定义和初始化


```
#include <stdio.h>
int main()
{
    int i, j;
    int a[2][3] = {1,2,3,4,5,6};
    printf("%p\n", a);

    for (i=0; i<2; i++)
    {
        for (j=0; j<3; j++)
        {
            printf("%d\t", a[i][j]);
        }
        printf("\n");
    }

    for (i=0; i<3; i++)
    {
        for (j=0; j<2; j++)
        {
            printf("%d\t", a[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```





6.2 数组的定义和初始化—二维数组的定义和初始化

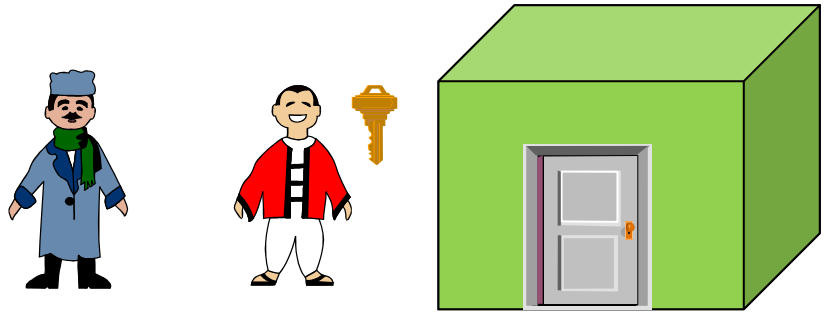
【例8】从键盘输入某年某月（包括闰年），编程输出该年的该月拥有的天数

```
1   #include <stdio.h>
2   #define  MONTHS 12
3   int main()
4   {
5       int days[2][MONTHS] = {{31,28,31,30,31,30,31,31,30,31,30,31},
6                              {31,29,31,30,31,30,31,31,30,31,30,31}};
7       int year, month;
8       do{
9           printf("Input year,month:");
10          scanf("%d,%d", &year, &month);
11      } while(month < 1 || month > 12);  /* 处理不合法数据的输入 */
12      if (((year%4 == 0) && (year%100 != 0)) || (year%400 == 0))/*闰年*/
13          printf("The number of days is %d\n", days[1][month-1]);
14      else  /*非闰年*/
15          printf("The number of days is %d\n", days[0][month-1]);
16      return 0;
17  }
```

第六讲 数组——学习内容

- 为什么使用数组(Array)?
- 数组的定义和初始化
- 向函数传递一维数组
- 向函数传递二维数组
- 数组综合应用案例

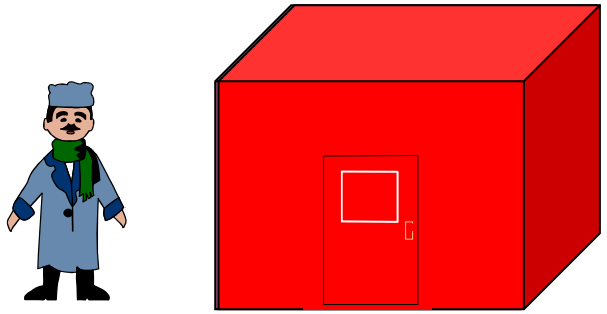
6.3 向函数传递一维数组



数组作函数参数

——按地址调用

传递数组的首地址，实参与形参数组占同一段内存单元



普通变量作函数参数

——按值调用

传递变量值的副本，实参与形参变量占不同的内存单元

6.3 向函数传递一维数组

【例】计算平均分：计数控制—键盘输入学生人数

```
1   #include <stdio.h>
2   #define N 40
3   int Average(int score[], int n);
4   void ReadScore(int score[], int n);
5   int main()
6   {
7       int score[N], aver, n;
8       printf("Input n:");
9       scanf("%d", &n);
10      ReadScore(score, n);
11      aver = Average(score, n);
12      printf("Average score is %d\n",aver);
13
14  }
```

用不带下标的数组名做函数实参

```
int Average(int score[], int n)
{
    int i, sum = 0;
    for (i=0; i<n; i++)
    {
        sum += score[i];
    }
    return  sum / n;
}
```

```
void ReadScore(int score[], int n)
{
    int i;
    printf("Input score:");
    for (i=0; i<n; i++)
    {
        scanf("%d", &score[i]);
    }
}
```

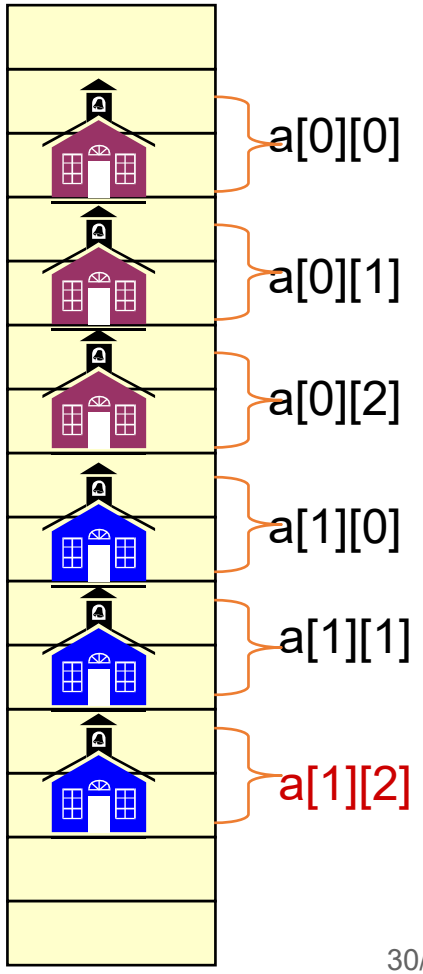
第六讲 数组——学习内容

- 为什么使用数组(Array)?
- 数组的定义和初始化
- 向函数传递一维数组
- 向函数传递二维数组
- 数组综合应用案例

6.4 向函数传递二维数组


```
short a[2][3];
```

向函数传递二维数组的首地址



- 在声明函数的二维数组形参时，为什么不能省略数组第二维的长度（列数）呢？
- 元素`a[i][j]`在数组中相对于第一个元素的位置：

```
i * 3 + j
```

偏移1*3+2个元素

元素地址：首地址+偏移量

6.4 向函数传递二维数组

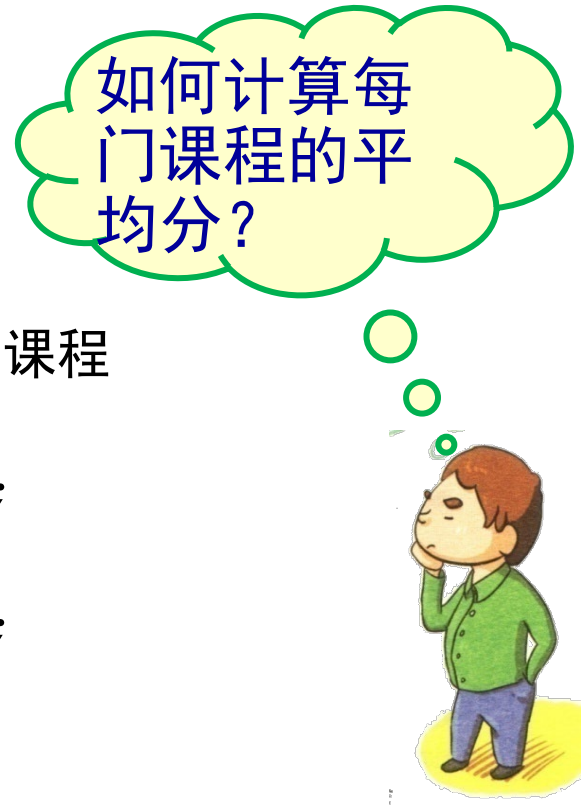
- 保存n个学生一门课程的成绩
  - 用一维数组
  - `int Average(int score[], int n);`
  - 通常不指定数组的长度，用另一个形参来指定数组的大小
- 保存n个学生的m门课程的成绩
  - 用二维数组
  - `void Average(int score[][COURSE_N], float aver[], int n);`
  - 可省略数组第一维的长度，不能省略第二维的长度
  - 数组**aver**可保存每个学生的平均分，或每门课程的平均分

6.4 向函数传递二维数组

计算每个学生的总分和平均分

```
void AverforStud(int score[][COURSE_N], int sum[],
                 float aver[], int n)
{
  int  i, j;

  for (i=0; i<n; i++)   //先遍历每个学生
  {
       sum[i] = 0;
       for (j=0; j<COURSE_N; j++) //遍历每门课程
       {
            sum[i] = sum[i] + score[i][j];
       }
       aver[i] = (float) sum[i] / COURSE_N;
  }
}
```

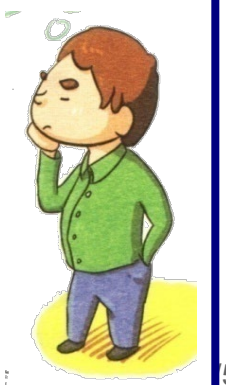


6.4 向函数传递二维数组

例 计算每门课程的总分和平均分

```
void AverforCourse(int score[][COURSE_N], int sum[],
                   float aver[], int n)
{
   int  i, j;

   for (j=0; j<COURSE_N; j++) //先遍历每门课程
   {
       sum[j] = 0;
       for (i=0; i<n; i++)     //遍历每个学生
       {
             sum[j] = sum[j] + score[i][j];
       }
       aver[j] = (float) sum[j] / n;
   }
}
```



6.4 向函数传递二维数组-要点总结

- C语言里面对二维数组的存储是按照一维数组来处理的
- 二维数组按照行展开的方式按顺序存储
- 参数传递二维数组，**必须指定二维数组的列数**
- 否则，函数无法勾画出二维数组的组织形式
- 有了列长度，通过下标a[i][j]时才能得到正确的下标地址：

a[i][j] = a[i * COLNUM + j]

本讲小结

1. **数组基础**
   数组是存储在连续内存空间的同类型数据集合。
   一维数组和二维数组是常见的数组形式。
   数组可以通过下标访问，下标从0开始。
2. **数组定义与初始化**
   使用宏定义数组长度，便于维护。
   数组初始化可以指定初始值或使用循环。
3. **数组使用注意事项**
   防止数组越界是程序员的责任。
4. **sizeof操作符**
   使用sizeof操作符可以获得数组的大小。
5. **数组作为函数参数**
   数组作为参数传递时实际上是传递数组的首地址。
   二维数组作为参数传递需要指定列数。