第4章 数 组

（视频讲解：1小时）

为了存放鞋子，假设你把衣柜最下面的一层分成了10个连续的格子。此时，让他人帮你拿鞋子就会很方便，例如你可直接告诉他拿衣柜最下面一层第三个格子中的鞋子。同样假设现在我们有10个整数存储在内存中，为方便存取，我们可以借助C语言提供的数组，通过一个符号来访问多个元素。

学习目标：

- 一维数组的原理及使用方法。
- 字符数组的使用方法。
- str等系列函数的使用方法。

4.1 一维数组

4.1.1 数组的定义

某班学生的学习成绩、一行文字、一个矩阵等数据的特点如下：

（1）具有相同的数据类型。

（2）使用过程中需要保留原始数据。

C语言为了方便操作这些数据，提供了一种构造数据类型——数组。所谓数组，是指一组具有相同数据类型的数据的有序集合。

一维数组的定义格式为

```
类型说明符  数组名 [常量表达式];
```

例如，

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

定义一个整型数组，数组名为a，它有10个元素。

声明数组时要遵循以下规则：

（1）数组名的命名规则和变量名的相同，即遵循标识符命名规则。

（2）在定义数组时，需要指定数组中元素的个数，方括号中的常量表达式用来表示元素的个数，即数组长度。

（3）常量表达式中可以包含常量和符号常量，但不能包含变量。也就是说，C语言不允许对数组的大小做动态定义，即数组的大小不依赖于程序运行过程中变量的值。

以下是错误的声明示例：

```
int n;
```

```
scanf("%d", &n);  /* 在程序中临时输入数组的大小 */
int a[n];
```

数组声明的其他常见错误如下：

```
① float a[0];        /* 数组大小为 0 没有意义 */
② int b(2)(3);           /* 不能使用圆括号 */
③ int k=3, a[k];     /* 不能用变量说明数组大小*/
```

4.1.2 一维数组在内存中的存储

语句 int mark[100];定义的一维数组 mark 在内存中的存放情况如图 4.1.1 所示，每个元素都是整型元素，占用 4 字节，数组元素的引用方式是“数组名[下标]”，所以访问数组 mark 中的元素的方式是 mark[0],mark[1],...,mark[99]。注意，没有元素 mark[100]，因为数组元素是从 0 开始编号的。

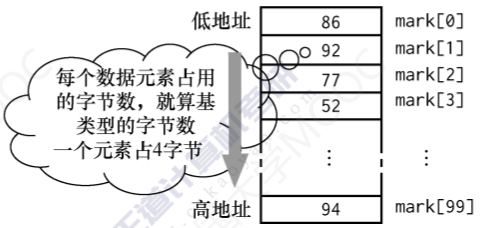

图 4.1.1 一维数组 mark 在内存中的存放

下面介绍一维数组的初始化方法。

（1）在定义数组时对数组元素赋初值。例如，

```
int a[10]={0,1,2,3,4,5,6,7,8,9};
```

不能写成

```
int a[10];a[10]={0,1,2,3,4,5,6,7,8,9}
```

（2）可以只给一部分元素赋值。例如，

```
int a[10]={0,1,2,3,4};
```

定义 a 数组有 10 个元素，但花括号内只提供 5 个初值，这表示只给前 5 个元素赋初值，后 5 个元素的值为 0。

（3）如果要使一个数组中全部元素的值为 0，那么可以写为

```
int  a[10]={0,0,0,0,0,0,0,0,0,0};
```

或

```
int a[10]={0};
```

（4）在对全部数组元素赋初值时，由于数据的个数已经确定，因此可以不指定数组的长度。例如，

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

下面借助一个数组的实例来掌握数组元素的赋值、访问、数组传递。例 4.1.1 中给出了该例

的全部代码。由于截图无法全部显示，因此这里首先给出全部代码，然后单独对每部分进行解析。

【例 4.1.1】一维数组的存储及函数传递。

```
#include <stdio.h>
#include <stdlib.h>

//一维数组的传递，数组长度无法传递给子函数
//C 语言的函数调用方式是值传递
void print(int b[],int len)
{
    int i;
    for(i=0;i<len;i++)
    {
        printf("%3d",b[i]);
    }
    b[4]=20;  //在子函数中修改数组元素
    printf("\n");
}
//数组越界
//一维数组的传递
#define N 5
int main()
{
    int j=10;
    int a[5]={1,2,3,4,5};  //定义数组时，数组长度必须固定
    int i=3;
    a[5]=20;  //越界访问
    a[6]=21;
    a[7]=22;  //越界访问会造成数据异常
    print(a,5);
    printf("a[4]=%d\n",a[4]);  //a[4]发生改变
    system("pause");
}
```

图 4.1.2 显示了代码运行情况。如图 4.1.2 所示，在第 24 行左键打上断点，然后单击“运行”按钮，在监视窗口一次输入&j、&a、&i 来查看整型变量 j、整型数组 a、整型变量 i 的地址，左键拖动对应地址到内存窗口即可看到三个变量的地址，这里就像我们给衣柜的每个格子的编号，第一格、第二格……一直到柜子的最后一格。操作系统对内存中的每个位置也给予一个编号，对于 Windows 32 位控制台应用程序来说，这个编号的范围是从 0x00 00 00 00 到 0xFF FF FF FF，

总计为2的32次方，大小为4G。这些编号称为地址。

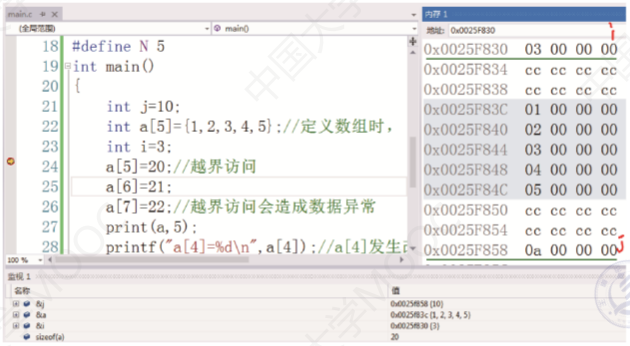

图4.1.2 代码运行情况1

我们看到，先定义的变量j的地址大于后定义的变量i的地址，所以先定义的变量放在高地址，后定义的变量放在低地址。其实每个函数开始执行时，系统会为其分配对应的函数栈空间，而变量j、变量a、变量i都在main函数的栈空间中，由于后定义的变量在上面，因此这种效果称为栈向上增长。

在监视窗口中输入sizeof(a)，可以看到数组a的大小为20字节，计算方法其实就是sizeof(int)*5：数组中有5个整型元素，每个元素的大小为4字节，所以共有20字节。访问元素的顺序是依次从a[0]到a[4]，a[5]=20、a[6]=21均为访问越界。图4.1.3也显示了代码运行情况，从中看出，执行到第26行时，数组a与变量j中间的8字节的保护空间已被赋值（微软公司的编译器在不同的变量间设置了保护空间），而执行到第27行时，变量j的值被修改了，这就是访问越界的危险性——未对变量j赋值，其值却发生了改变!

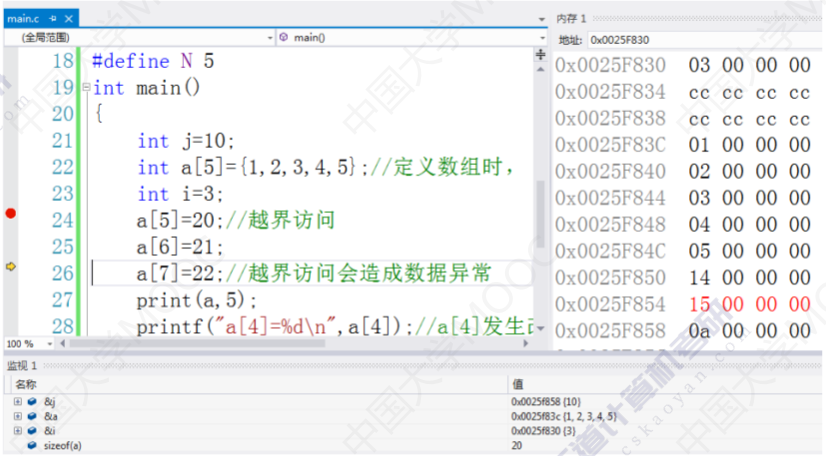

图4.1.3 代码运行情况2

数组另一个值得关注的地方是，编译器并不检查程序对数组下标的引用是否在数组的合法范围内。这种不加检查的行为有好处也有坏处，好处是不需要浪费时间对有些已知正确的数组下标

进行检查，坏处是这样做将无法检测出无效的下标引用。一个良好的经验法则是：如果下标值是通过那些已知正确的值计算得来的，那么就无须检查；如果下标值是由用户输入的数据产生的，那么在使用它们之前就必须进行检查，以确保它们位于有效范围内（对于后端开发工程师来说，如果这个下标是从前端接收过来的，那么一定要进行判断）。

如图 4.1.4 所示，在第 27 行按 F11 键，进入 print 函数，这时会发现数组 b 的大小变为 4 字节，如图 4.1.5 所示，这是因为一维数组在传递时，其长度是传递不过去的，所以我们通过 len 来传递数组中的元素个数。实际数组名中存储的是数组的首地址，在调用函数传递时，是将数组的首地址给了变量 b（其实变量 b 是指针类型，具体原理会在下一章讲解），在 b[]的方括号中填写任何数字都是没有意义的。这时我们在 print 函数内修改元素 b[4]=20，可以看到数组 b 的起始地址和 main 函数中数组 a 的起始地址相同，即二者在内存中位于同一位置，当函数执行结束时，数组 a 中的元素 a[4]就得到了修改。

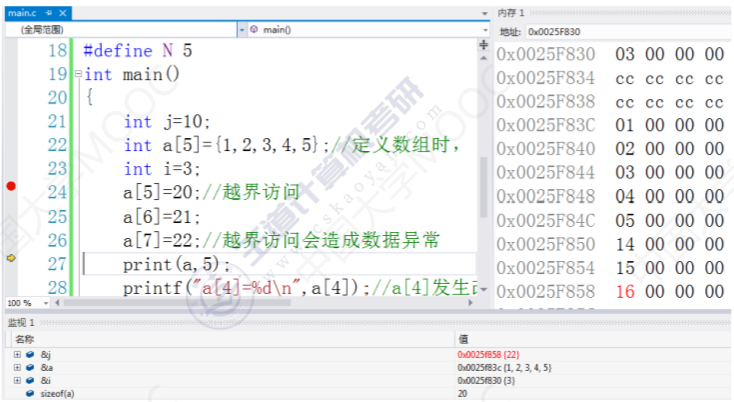

图 4.1.4　代码运行情况 3

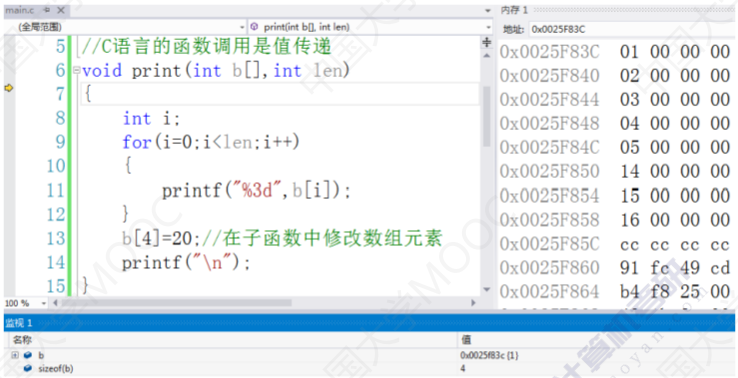

图 4.1.5　代码运行情况 4

4.2 字符数组

4.2.1 字符数组的定义及初始化

字符数组的定义方法与前面介绍的一维数组、二维数组的类似。例如，

```
char c[10];
```

字符数组的初始化可以采用以下方式。

（1）对每个字符单独赋值进行初始化。例如，

```
c[0]='l';c[1]=' ';c[2]='a';c[3]='m';c[4]='';c[5]='h';c[6]='a';c[7]='p';c[8]='p';c[9]='y';
```

（2）对整个数组进行初始化。例如，

```
char c[10]={'l','a','m','h','a','p','p','y'}
```

但工作中一般不用以上两种初始化方式，因为字符数组一般用来存取字符串。通常采用的初始化方式是 char c[10]= "hello"。因为 C 语言规定字符串的结束标志为'\0'，而系统会对字符串常量自动加一个'\0'，为了保证处理方法一致，一般会人为地在字符数组中添加'\0'，所以字符数组存储的字符串长度必须比字符数组少 1 字节。例如，char c[10]最长存储 9 个字符，剩余的 1 个字符用来存储'\0'。下面通过例 4.2.1 来看具体代码，代码执行过程如图 4.2.1 所示。

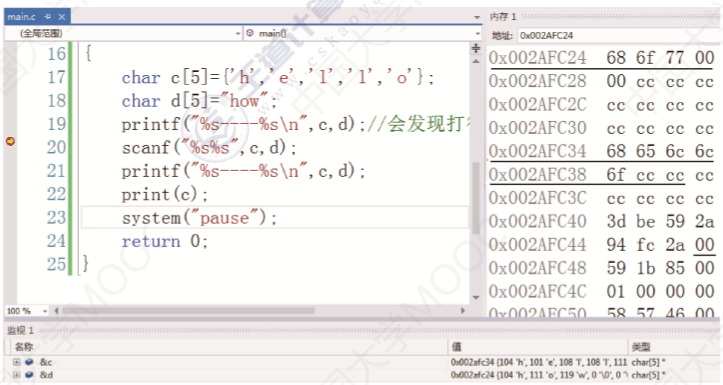

图 4.2.1 例 4.2.1 中代码的执行过程

【例 4.2.1】字符数组初始化及传递。

```
#include <stdio.h>
#include <stdlib.h>
void print(char c[])
{
    int i=0;
    while(c[i])
    {
        printf("%c",c[i]);
        i++;
```

```
        }
        printf("\n");
    }
    //字符数组存储字符串，必须存储结束符'\0'
    //scanf 读取字符串时使用%s
    int main()
    {
        char c[5]={'h','e','l','l','o'};
        char d[5]="how";
        printf("%s----%s\n",c,d); //会发现打很多“烫”字
        scanf("%s%s",c,d);
        printf("%s----%s\n",c,d);
        print(c);
        system("pause");
        return 0;
    }
```

例 4.2.1 中代码的执行结果如图 4.2.2 所示。为什么对数组赋值"hello"却打印出很多“烫”字？这是因为 printf 通过%s 打印字符串时，原理是依次输出每个字符，当读到结束符'\0'时，结束打印；scanf 通过%s 读取字符串，对 c 和 d 分别输入"are"和"you"（中间加一个空格），scanf 在使用%s 读取字符串时，会忽略空格和回车。

我们通过 print 函数模拟实现 printf 的%s 打印效果，如图 4.2.3 所示，当 c[i]为'\0'时，其值是 0，循环结束，也可以写为 c[i]!='\0'。

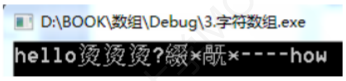

图 4.2.2　例 4.2.1 中代码的执行结果

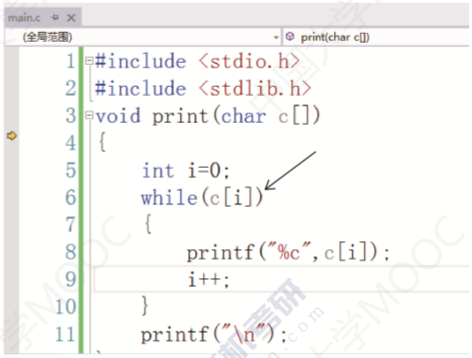

图 4.2.3　print 函数模拟实现 printf 的%s 打印效果

4.2.2　gets 函数与 puts 函数

gets 函数类似于 scanf 函数，用于读取标准输入。前面我们已经知道 scanf 函数在读取字

符串时遇到空格就认为读取结束，所以当输入的字符串存在空格时，我们需要使用 gets 函数进行读取。

gets 函数的格式如下：

```
char *gets(char *str);
```

gets 函数从 STDIN（标准输入）读取字符并把它们加载到 str（字符串）中，直到遇到换行符（\n）或到达 EOF。如图 4.2.4 所示，在第 8 行打断点，执行后，我们输入"how are you"，共 11 个字符，可以看到 gets 会读取空格，同时可以看到我们并未给数组进行初始化赋值，但是最后有'\0'，这是因为 gets 遇到\n 后，不会存储\n，而是将其翻译为空字符'\0'。

puts 函数类似于 printf 函数，用于输出标准输出。puts 函数的格式如下：

```
int puts(char *str);
```

函数 puts 把 str（字符串）写入 STDOU（标准输出）。puts 执行成功时返回非负值，执行失败时返回 EOF。如图 4.2.4 所示，puts 会将数组 c 中存储的"how are you"字符串打印到屏幕上，同时打印换行，相对于 printf 函数，puts 只能用于输出字符串，同时多打印一个换行符。

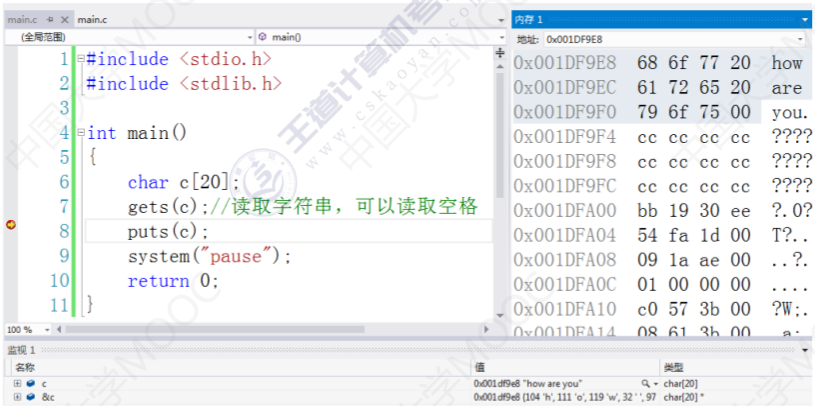

图 4.2.4 gets 函数和 puts 函数

4.2.3 str 系列字符串操作函数

str 系列字符串操作函数主要包括 strlen、strcpy、strcmp、strcat 等（读者可以加入前言中的 QQ 群，群内提供 C/C++函数大全）。strlen 函数用于统计字符串长度，strcpy 函数用于将某个字符串复制到字符数组中，strcmp 函数用于比较两个字符串的大小，strcat 函数用于将两个字符串连接到一起。各个函数的具体格式如下所示：

```
#include <string.h>
  size_t strlen(char *str);
  char *strcpy(char *to, const char *from);
  int strcmp(const char *str1, const char *str2);
```

```
char *strcat(char *str1, const char *str2);
```

对于传参类型 char*，直接放入字符数组的数组名即可。

接下来我们通过例 4.2.2 来具体学习 str 系列字符串操作函数，掌握每个函数的内部实现。

【例 4.2.2】str 系列字符串操作函数的使用。

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

int mystrlen(char c[])
{
    int i=0;
    while(c[i++]);
    return i-1;
}
//strlen 统计字符串长度
int main()
{
    int len;  //用于存储字符串长度
    char c[20];
    char d[100]="world";
    while(gets(c)!=NULL)
    {
        puts(c);
        len=strlen(c);
        printf("len=%d\n",len);
        len=mystrlen(c);
        printf("mystrlen len=%d\n",len);
        strcat(c,d);
        strcpy(d,c);  //c 中的字符串复制给 d
puts(d);
        printf("c?d %d\n",strcmp(c,d));
        puts(c);
    }
    system("pause");
    return 0;
}
```

通过 gets 函数循环读取字符串的目的是，方便大家不断地输入不同的字符串并查看程序的执行效果，并在修改程序中的某部分后能够以多种输入进行测试。如果要结束循环，那么可以按组合键 Ctrl+Z。图 4.2.5 所示为我们输入"hello"后的执行结果，通过 strlen 函数计算的字符串长度为 5，我们自己写的函数就是 strlen 函数的计算原理，即通过判断结束符来确定字符串的长度。

strcpy 函数用来将字符串中的字符逐个地赋值给目标字符数组。例中我们将 c 复制给 d，就是将 c 中的每个字符依次赋值给 d，也会将结束符赋值给 d。注意，目标数组一定要大于字符串大小，即 sizeof(d)>strlen(c)，否则会造成访问越界。

图 4.2.5 输入"hello"后程序的执行结果

strcmp 函数用来比较两个字符串的大小，由于字符数组 c 中的字符串与 d 相等，所以这里的返回值为 0。如果 c 中的字符串大于 d，那么返回值为 1；如果 c 中的字符串小于 d，那么返回值为−1。如何比较两个字符串的大小呢？具体操作是从头开始，比较相同位置字符的 ASCII 码值，若发现不相等则直接返回，否则接着往后比较。例如，strcmp("hello","how")的返回值是−1，即"hello"小于"how"，因为第一个字符 h 相等，接着比较第二个位置的字符，e 的 ASCII 码值小于 o 的，然后返回−1。

strcat 函数用来将一个字符串接到另外一个字符串的末尾。例中字符数组 c 中存储的是"hello"，我们将 d 中的"world"与 c 拼接，最终结果为"helloworld"。注意，目标数组必须大于拼接后的字符串大小，即 sizeof(c)>strlen(“helloworld”)。