



中国科学技术大学  
University of Science and Technology of China

# 计算机程序设计

## 第二章 程序设计入门

白雪飞

中国科学技术大学微电子学院

# 目录

---

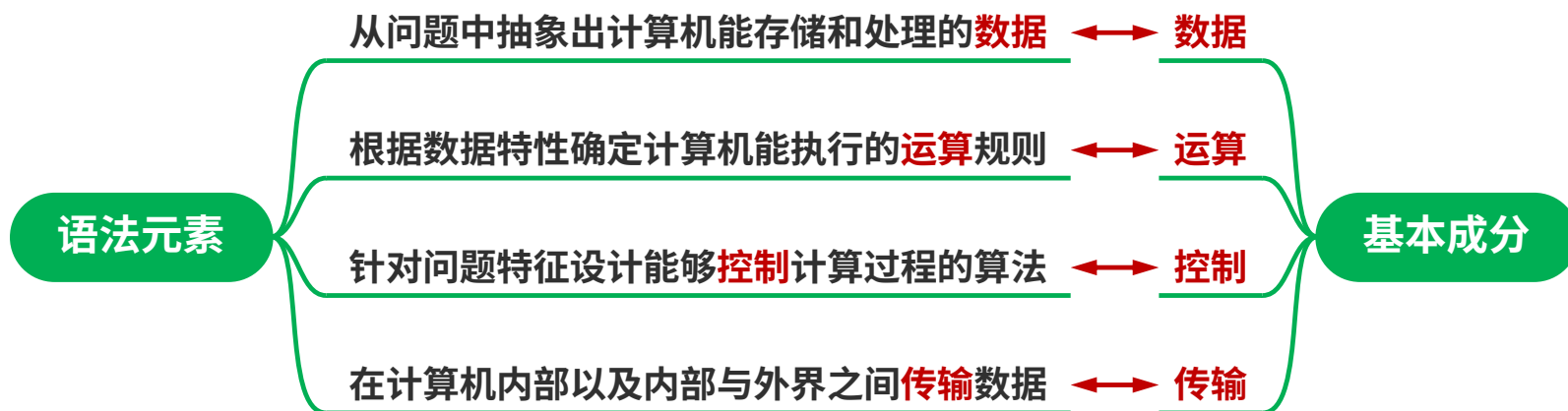


- 引言
- 数据与运算
- 输出与输入
- 条件判断与选择
- 循环与数组
- C语言程序规范
- 小结



# 引言

## 程序设计语言



## 程序设计语言的语法元素和基本成分



# 数据与运算



- 数据与数据类型
- 运算符与表达式

## ■ 常量 (Constant)

- 程序运行期间，其值不能或不允许改变的数据对象
- **字面常量** (Literal Constant)、**符号常量** (Symbolic Constant)

## ■ 变量 (Variable)

- 程序运行期间，其值可以改变的数据对象
- **变量名代表**存放变量数据对象的**内存空间**

## ■ 标识符 (Identifier)

- 标识某个实体的符号，用于为变量、函数等命名
- 标识符由**字母、数字、下划线**组成，且**数字不能作为首字符**
- C语言标识符是大小写敏感的
- 通常使用有意义的单词或缩写命名标识符
- 不能与C语言关键字重复

# 数据类型

## ■ 基本数据类型

- 字符型: `char` 占用1字节, 字符ASCII编码格式
- 整型: `int` 通常占用4字节, 整数编码格式
- 浮点型: `float` 占用4字节, 单精度浮点数编码格式
- `double` 占用8字节, 双精度浮点数编码格式

## ■ 变量的数据类型

- 在变量的声明中指定其数据类型

```
int    weight;    // 声明整型变量weight
float  bmi;        // 声明浮点型变量bmi
```

## ■ 常量的数据类型

- 编译器根据常量的值推定其数据类型



## ■ 注释 (Comment)

- 编译器忽略注释部分的内容
- 适当的注释有助于对程序的阅读理解
- 在调试过程中，注释部分程序段有助于定位错误

## ■ 注释的方式

- 单行注释：// 表示其后是注释，直至本行结束

```
float bmi;           // 声明存储BMI值的浮点型变量bmi，其中的值未知
```

- 多行注释：/\*和\*/ 组合范围内的一行或多行内容都视为注释

```
/*  
    声明存储BMI值的浮点型变量bmi  
    其中的值未知  
*/  
float bmi;
```



- 数据与数据类型
- 运算符与表达式

## 操作数

- 也称操作对象、运算对象
- 常量和变量都是操作数
- 常量: 80, 1.85
- 变量: bmi

## 运算符

- 除法运算符 /
  - 浮点数参与的除法运算, 其规则和数学运算一致
- 赋值运算符 =
  - 将右侧的值赋给左侧变量

## 表达式

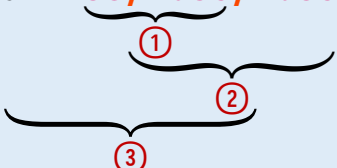
- 运算符和操作数的组合
- 常量和变量都是最简单的表达式
- $bmi = 80 / 1.85 / 1.85$ ,  
 $80 / 1.85 / 1.85$ ,  $80 / 1.85$

$bmi = 80 / 1.85 / 1.85;$

## 运算次序

- 根据式中运算符的优先级和结合性决定

■  $bmi = 80 / 1.85 / 1.85$



## 类型转换

- 运算符两侧操作数类型不同时, 自动进行类型转换
- 算术运算向表示范围更大的类型转换
- 赋值运算向左侧类型转换

## 语句

- 表达式后添加; 可以构成一条语句
- 并非所有语句都以; 结尾



# 输出与输入



- 用printf()函数输出
- C语言程序与函数
- 用scanf()函数输入



# 格式化输出函数

## ■ 函数一般形式

```
printf("输出格式字符串", 表达式);
```

- 将表达式的值转换成可显示的形式并输出到屏幕上

## ■ 常用输出格式

- %c 输出为单个字符
- %d 输出为十进制整数形式
- %f 输出为十进制小数形式，小数点后六位数字

```
printf("%d", 80);           // 以十进制整数形式输出常量80
printf("%f", 1.85);         // 以十进制小数形式输出常量1.85
printf("%f", bmi);          // 以十进制小数形式输出变量bmi的值
printf("%f", 80/1.85/1.85); // 以十进制小数形式输出表达式的值
```



- 用printf()函数输出
- C语言程序与函数
- 用scanf()函数输入

## ■ 例2.2-1：计算并显示BMI值。

```
#include <stdio.h>           // 文件包含，stdio.h包含标准输入/输出库函数的说明

int main()                   // C语言程序有且只有一个main函数，程序从main函数开始执行
{                             // 大括号中包含的是main函数的函数体
    float bmi;               // 声明存储BMI值的浮点型变量bmi，其中的值未定

    bmi = 80 / 1.85 / 1.85;  // 计算与赋值语句，结果存储在变量bmi中
    printf("%f", bmi);       // 输出语句，用十进制小数格式输出变量bmi的值

    return 0;                // 程序执行完毕，返回到操作系统
}
```

### 运行结果：

23.374725

### 结果分析：

程序输出结果的小数位数只与输出格式有关，而与有效数字及精度无关。

$80/1.85^2$ 精确结果为：23.374726077428...

程序输出结果虽然有八位数字，但是只有七位是准确的。





- 用printf()函数输出
- C语言程序与函数
- 用scanf()函数输入

## ■ 函数一般形式

```
scanf("输入格式字符串", 内存地址);
```

- 将输入的字符串转换成指定类型数据，存放到从指定地址开始的内存中

## ■ 常用输入格式

- `%c` 接收任意单个字符，并转换为ASCII编码
- `%d` 接收符合整数格式的一串字符，并转换为整数
- `%f` 接收符合浮点数格式的一串字符，并转换为单精度浮点数
- `%lf` 接收符合浮点数格式的一串字符，并转换为双精度浮点数

```
int    weight;  
float  height;  
scanf("%d", &weight);    // 将输入的整型数据存入变量weight所在内存空间  
scanf("%f", &height);    // 将输入的浮点型数据存入变量height所在内存空间
```

# 数据输入、处理、输出举例



## ■ 例2.2-2：输入体重和身高，计算并显示BMI值。

```
#include <stdio.h>                                // 包含标准输入/输出库函数调用所需信息

int main()
{
    float bmi, height;                             // 同类型变量可以一起声明
    int weight;

    scanf("%d", &weight);                          // 输入整数格式的体重值
    scanf("%f", &height);                          // 输入浮点数格式的身高值
    bmi = weight / height / height;                // 计算BMI
    printf("%f", bmi);                             // 输出BMI的值

    return 0;
}
```

### 运行结果：

```
80↵
1.85↵
23.374725
```

### 变量监测：

```
weight    ??? → 80
height    ??? → 1.85
bmi       ??? → 23.374725...
```



# 条件判断与选择



- 关系运算
- if-else语句与流程图
- 逻辑运算

# 关系运算

## ■ 关系运算符

- 小于  $<$ ，小于等于  $<=$ ，大于  $>$ ，大于等于  $>=$ ，等于  $==$ ，不等于  $!=$
- 二元运算符：有两个操作数

## ■ 关系表达式

- 关系运算符将两个操作数连接起来
- 关系表达式的值只能为1或0，且为int类型
- 若关系成立，则关系表达式的值为1，表示“真”
- 若关系不成立，则关系表达式的值为0，表示“假”

## ■ 关系表达式求值举例

- 表达式  $-1 > 1$  的值为0
- 表达式  $1.8 <= 3.5$  的值为1
- 表达式  $3 != 5$  的值为1
- 若变量bmi的值为23.37，则表达式  $bmi > 25$  的值为0

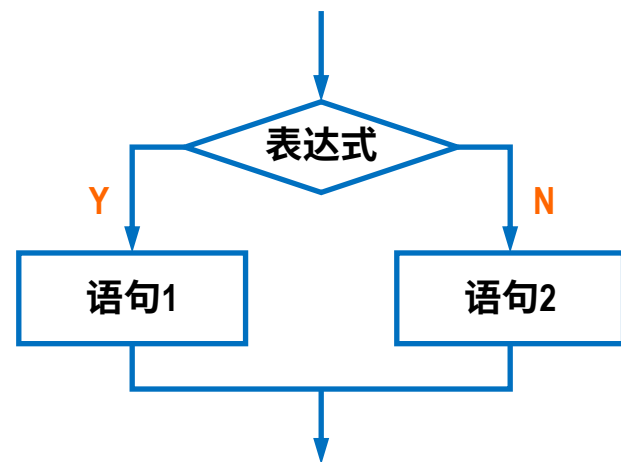


- 关系运算
- if-else语句与流程图
- 逻辑运算

# if-else语句

## ■ 基本形式

```
if (表达式)
    语句1
else
    语句2
```



## ■ 说明

- if-else语句是分支结构语句
- if后括号中的表达式为判断条件
- 若表达式的值为“非零”，判断为“真”，执行语句1
- 若表达式的值为“零”，判断为“假”，执行语句2
- 若语句1或语句2包含多条语句，应使用{}括起来
- else部分是可选的
- 若没有else部分，等价于语句2为空语句，该分支没有任何操作



# 分支语句举例



## ■ 例2.3-1：输入体重和身高，显示是否超重。

```
#include <stdio.h>

int main()
{
    float bmi, height;
    int weight;

    printf("输入体重 (千克) : "); // 输出提示信息
    scanf("%d", &weight); // 输入整数格式的体重值
    printf("输入身高 (米) : "); // 输出提示信息
    scanf("%f", &height); // 输入浮点数值的身高值
    bmi = weight / height / height; // 计算BMI

    if (bmi > 25)
        printf("你超重了哦!"); // 若BMI大于25, 则提示超重
    else
        printf("你没有超重。"); // 否则显示未超重

    return 0;
}
```

缺少对输入数据的合法性检查

### 运行结果一：

输入体重 (千克) : 80↵  
输入身高 (米) : 1.85↵  
你没有超重。

### 运行结果二：

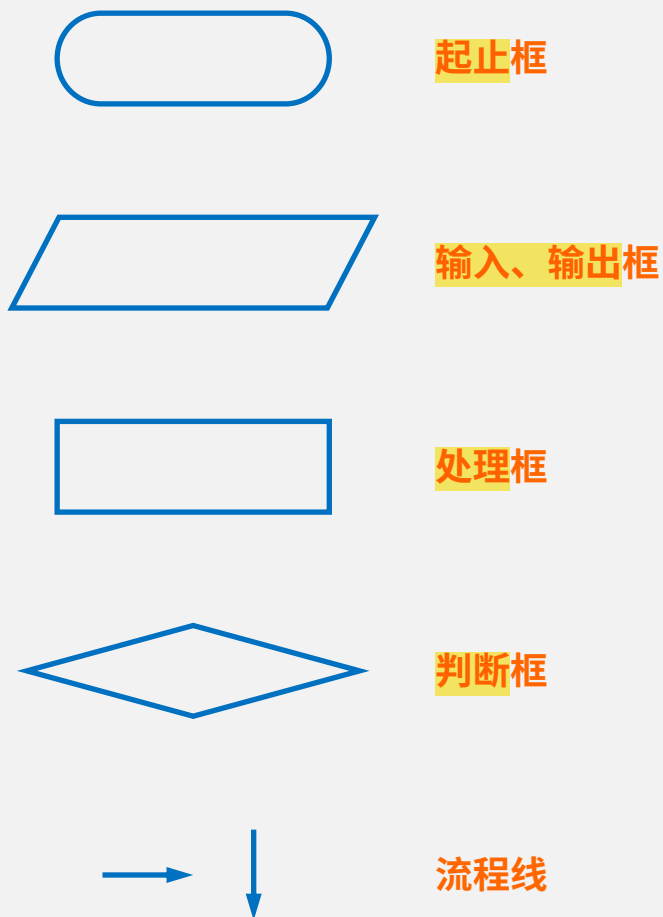
输入体重 (千克) : 78↵  
输入身高 (米) : 1.72↵  
你超重了哦!

### 运行结果三：

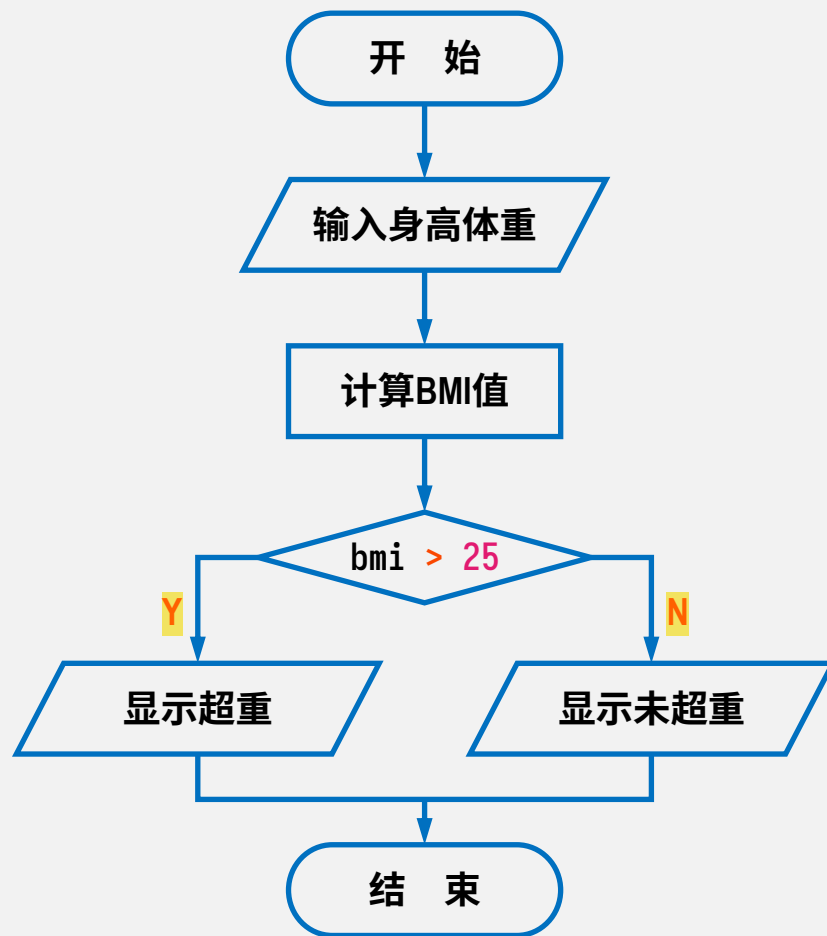
输入体重 (千克) : -60↵  
输入身高 (米) : 1.65↵  
你没有超重。

### 运行结果四：

输入体重 (千克) : 0↵  
输入身高 (米) : 0↵  
你没有超重。



常用流程图符号



流程图示例



- 关系运算
- if-else语句与流程图
- 逻辑运算

## ■ 逻辑运算符

- **一元**运算符：逻辑非 **!**
- **二元**运算符：逻辑与 **&&**，逻辑或 **||**

```
if (bmi >= 20 && bmi <= 25)
    printf("你的体重很标准。");
else
    printf("你的体重超标了。");
```

## ■ 逻辑表达式

- 操作数逻辑值：**“非零”**为**“真”**，**“零”**为**“假”**
- 表达式的取值：**“真”**为**1**，**“假”**为**0**，且为**int**类型
- 逻辑**非**：若操作数为**“真”**，则表达式的值为**“假”**  
若操作数为**“假”**，则表达式的值为**“真”**
- 逻辑**与**：若左操作数为**“真”**，则表达式的值取决于**右操作数**逻辑值  
若左操作数为**“假”**，则表达式的值为**“假”**，**不再计算**右操作数
- 逻辑**或**：若左操作数为**“假”**，则表达式的值取决于**右操作数**逻辑值  
若左操作数为**“真”**，则表达式的值为**“真”**，**不再计算**右操作数



# 循环与数组

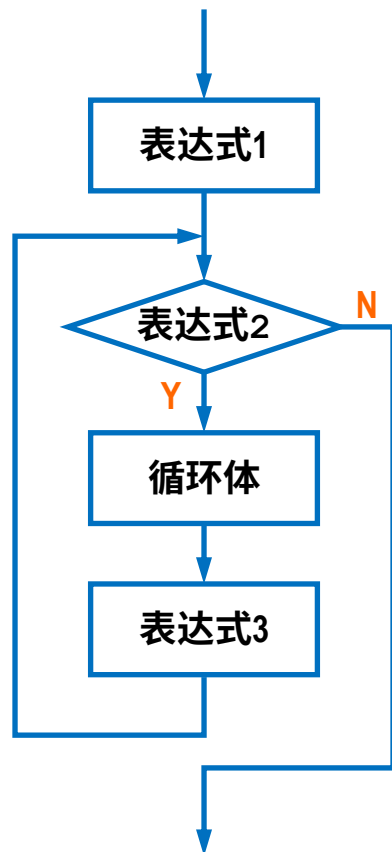
- for循环与伪代码
- 数组

## ■ 一般形式

```
for (表达式1; 表达式2; 表达式3)  
    循环体
```

## ■ 说明

- `for`语句是循环结构语句
- `for`后括号中有三个表达式，并使用两个`;`分隔开
- 表达式1只求值一次，通常为循环控制变量初始化
- 表达式2为循环条件，其值为“假”时循环终止
- 若循环体包含多条语句，应使用`{}`括起来
- 表达式3在循环体执行之后求值，通常用于修改循环控制变量的值



# for循环举例



## ■ 例2.4-1：对1~100之间的整数累加求和。

```
#include <stdio.h>

int main()
{
    int i, sum=0;           // sum用于累加求和，初始值为0

    for (i=1; i<=100; i=i+1)
        sum = sum + i;      // 将i累加求和并保存到sum中

    printf("sum=%d\n", sum);

    return 0;
}
```

### 运行结果：

sum=5050

### 程序解析：

- 1** 循环控制变量用于循环条件的判断，一般情况下，随着循环的进行，循环控制变量的值将趋于使循环结束，例如本例中的变量*i*；
- 2** for语句中的表达式1最先执行且只执行一次，通常用于循环控制变量的初始化，例如*i=1*；
- 3** for语句中的表达式2为循环条件，其值为“非零”时执行循环体，其值为“零”时终止循环；
- 4** for语句中的表达式3通常用于修改循环控制变量的值，使其趋向于使循环结束，例如*i=i+1*。



# 循环与分支语句举例

## ■ 例2.4-2：输入GPA并输出提示信息，重复三次。

```
#include <stdio.h>

int main()
{
    float GPA;
    int i;

    for (i=0; i<3; i++) {
        scanf("%f", &GPA);
        if (GPA > 4.3)
            printf("你作弊啦! \n");
        else
            printf("还要加油哦\n");
    }

    return 0;
}
```

// 循环控制变量

// 循环三次

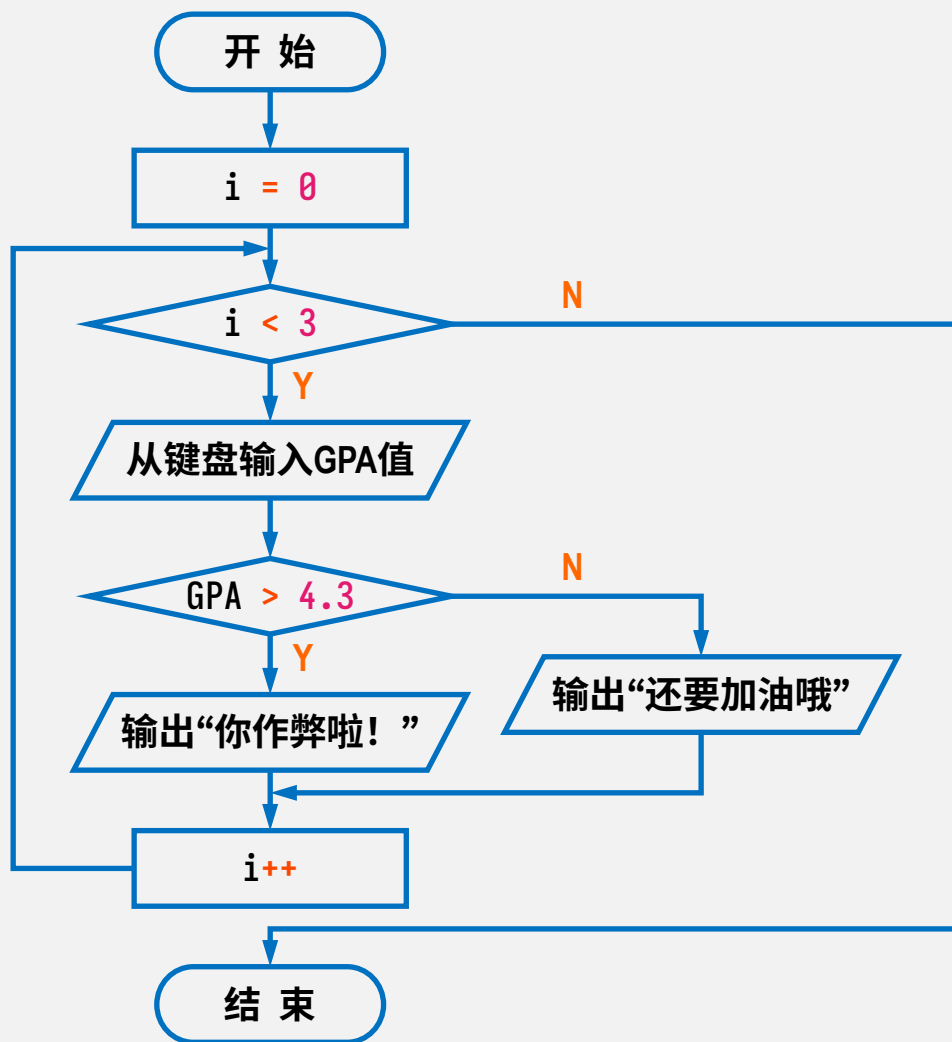
// for语句结束位置

### 程序解析：

- 1** C语言习惯从0开始计数，这将为计算带来诸多方便；
- 2** 表达式*i++*中，*++*为自增运算符，将使操作数*i*的值增加1；
- 3** *for*语句循环体包括两条语句，应使用*{}*把它们括起来。

### 运行结果：

```
3.8↵
还要加油哦
4.0↵
还要加油哦
5.2↵
你作弊啦!
```



```
1 For i = 0 to i < 3
2     从键盘输入GPA值
3     If GPA > 4.3 输出"你作弊啦！"
4     Else 输出"还要加油哦"
5     End If
6     i = i + 1
7 End For
```

## ▲ 伪代码

介于自然语言和高级语言之间  
便于撰写与修改，易于阅读与理解  
难以检查逻辑错误或语法问题

## ◀ 流程图

直观且规范  
有助于提前排除设计中的部分错误  
不易于绘制和修改

- for循环与伪代码
- 数组

# 数组 (Array)

## ■ 数组的声明

- 数组用于存储多个**同一类型**的数据
- **定义**数组时，指定数组元素的数据**类型**、数组**长度**

```
float data[8];
```

- 定义数组时，可以同时为数组元素赋初值，即**初始化**

```
// 定义数组的同时，将已知数据作为初值赋给该数组各个元素
```

```
float data[8] = {9.812, 9.806, 9.901, 9.788, 9.853, 9.790, 9.819, 9.787};
```

## ■ 数组的存储

- 数组元素在内存中按照下标顺序**依次连续**存放

data[0]	data[1]	data[2]	data[3]	data[4]	data[5]	data[6]	data[7]
9.812	9.806	9.901	9.788	9.853	9.790	9.819	9.787

数组在内存中的存储示意图

# 数组与循环举例



## ■ 例2.4-3：求数组中数据的平均值。

```
#include <stdio.h>

int main()
{
    float data[8] = {9.812, 9.806, 9.901, 9.788,
                    9.853, 9.790, 9.819, 9.787};

    float sum = 0;           // 用于求和的变量，初值为0
    float ave;               // 保存平均值的变量
    int i;                   // 循环控制变量

    for (i=0; i<8; i++) {    // 计数循环适合使用for语句
        sum = sum + data[i]; // 累加求和，数组下标从0开始
    }

    ave = sum / 8;           // 计算平均值
    printf("平均值是%f", ave); // 输出平均值

    return 0;
}
```

### 程序解析：

- 1** 变量声明放在函数体的最前面，包括函数中用到的所有变量；
- 2** 当循环体只包括一条语句时，可以不加{}，有些编程风格在这种情况下也加上{}，使逻辑结构清晰且便于调试；
- 3** 数组通常不能以整体参与运算，而是通过下标运算符[]逐个访问数组元素，下标可以是整数类型常量、变量或表达式；
- 4** 数组下标从0开始计数，最大至“数组长度-1”，数组下标超出数组范围，称为数组越界，编译器可能不会对此提示错误，当遇到与数组有关的错误时，应当考虑数组越界的可能性，特别是在数组下标为变量或表达式的情况下。

### 运行结果：

平均值是9.819500

# 数组与循环举例



## ■ 例2.4-4：从键盘输入数组元素并求平均值。

```
#include <stdio.h>

int main()
{
    float data[4];           // 数组元素值来自外部输入
    float sum=0, ave;
    int i;

    for (i=0; i<4; i++) {    // 循环四次
        scanf("%f", &data[i]);
        sum = sum + data[i]; // 数组下标从0开始
    }

    ave = sum / 4;
    printf("平均值是%.2f", ave); // 输出保留两位小数

    return 0;
}
```

### 程序解析：

- 1** 调试程序过程中，为了避免每次运行程序都要输入大量数据，可以先将**数组长度**设置成比较小的值，待调试基本完成后再改成实际长度做最终测试；
- 2** **数组元素**的用法与同类型普通变量完全一样，包括运算、输入、输出等；
- 3** 下标运算符**[]**具有最高优先级，可以将数组元素**data[i]**看作整体直接使用，无需在两边添加**()**；
- 4** 格式化输出函数**printf()**支持的格式很丰富，**"%.2f"**表示浮点数输出时保留两位小数；
- 5** 数组只能在定义时整体赋初值，不能在程序语句中整体赋值。

### 运行结果：

```
3.2 2.8 3.3 2.9↵
平均值是3.05
```



# C语言程序规范

## ■ 编程规范的作用

- 代码在大部分时间用于被阅读：新编代码、缺陷修补、特性扩展、新人学习
- 优秀的代码：可读、可维护、安全、可移植、可测试等

## ■ 部分基本编程规范

- 使用统一的排版风格，`{}`位置应使代码更紧凑，阅读节奏感更连续
- 缩进使用空格而不是制表符，每次缩进四个空格
- 标识符遵守统一的命名风格，有明确含义，符合阅读习惯，容易理解
- 按需注释，内容简洁明了，无二义性，信息全面且不冗余
- 不使用难以理解、无注释说明的“幻数”
- 用`()`明确表达式的操作顺序，避免过分依赖默认优先级
- 必须包含声明库函数的头文件
- 分支语句和循环体即使只有一条语句，也使用`{}`括起来
- `main`函数前面必须有类型`int`，结尾必须有`return`语句



# 小 结

## ■ 主要知识点

- 常量、变量、数据类型、运算符、表达式、数组的基本概念
- 格式化输入、输出函数的基本用法和常用格式
- 关系、逻辑运算符以及表达式求值规则
- 分支结构适用于在不同条件下进行差异化的数据处理
- 循环结构适用于批量处理数据，经常与数组配合使用
- 描述算法和程序流程的工具：流程图、伪代码等

## ■ 能力要求

- 了解常用的C语言语法
- 能够复现本章所有示例
- 能够对条件判断、输出语句进行局部修改，并得到预期的结果
- 能够绘制流程图、书写伪代码
- 了解C语言基本编程规范，并逐步养成良好的编程习惯



# 本章结束