1. 程序设计与C语言

1.1. 计算机和编程语言

1.1.1 计算机与编程语言：计算机怎么做事情，编程语言是什么

（1）计算机语言

程序是用特殊的编程语言写出来表达如何解决问题的

不是用编程语言来和计算机交谈，而是描述要求它如何做事情的过程或方法

（2）算法

我们要让计算机做计算，就需要找出计算的步骤，然后用编程语言写出来

计算机做的所有的事情都叫做计算

计算的步骤就是算法

1.1.2 计算机的思维方式：重复是计算机最擅长的

（1）程序的执行

解释：借助一个程序，那个程序能试图理解你的程序，然后按照你的要求执行

编译：借助一个程序，就像一个翻译，把你的程序翻译成计算机真正能懂的语言——机器语言——写的程序，然后这个机器语言写的程序就能直接执行了

（2）解释语言vs编译语言

语言本无编译/解释之分

常用的执行方式而已

解释型语言有特殊的计算能力

编译型语言有确定的运算性能

1.2 C语言

1.2.1 为什么是C：C语言在工业界占有重要地位，在很多领域无可替代

（1）其他语言

现代的编程语言在语法上的差异很小——几乎都是C-like语言

语言的能力/适用领域主要是由库和传统所决定的

1.2.2 简单历史：关于C语言和版本的及其简单的历史

（1）C语言

C语言是从B语言发展而来的，B语言是从BCPL发展而来的，BCOL是从FORTRAN发展而来的

BCPL和B都支持指针间接方式，所以C也支持了

C语言还受到了PL/I的影响，还和PDP-Ⅱ的机器语言有很大的关系

1973年3月，第三版的Unix上出现了C语言的编译器

1973年11月，第四版的Unix(System Four)发布了，这个版本是完全用C语言重新写的

（2）C的发展与版本—K&R

经典C，又被叫做”K&R the C”

The C Programming Laguage, by Brian Kernighan and Dennis Ritchine, 2nd Edition, Prentice Hall

（3）C的发展与版本—标准

1989年ANSI发布了一个标准——ANSI C

1990年ISO接受了ANSI的标准——C89

C的标准在1995年和1999年两次更新——C95和C99

1.2.3 编程软件：C语言的编程软件选择太多，我们推荐Dev C++

（1）C语言用在哪里

操作系统

嵌入式系统

驱动程序

底层驱动

图形引擎、图像处理、声音效果

（2）C语言是一种工业语言

开发效率>>学习过程

开发效率>>开发乐趣

日常应用很少直接用C语言编写

学习C的过程主要是写练习代码，而非真实软件

（3）编译-->运行

C需要被编译才能运行，所以你需要：编辑器+编译器或者IDE（集成开发环境）

（4）推荐的编程软件

Dev C++

MS Visual Studio Express (Windows)

Xcode (Mac OS X)

Eclipse-CDT

Geany (+MinGW)

Sublime (+MinGW)

vim/emacs (+MinGW)

1.3 第一个程序

1.3.1 第一个程序：如何在Dev C++中编辑、编译和运行程序

1.3.2 详解第一个程序：程序框架、printf、出错怎么办

（1）程序框架

|  |
| --- |
| #include <stdio.h>  int main()  {    return 0;  } |

本课程中所有的程序都需要这一段，直到学函数之前，我们的代码都只是在这个框架之中

（2）输出

printf("Hello World!\n");

""里面的内容叫做“字符串”，printf会把其中的内容原封不动地输出

\n表示需要在输出的结果后面换一行

（3）程序中的错误

编译的时候发现的错误所在的地方会以红色的底表示出来

具体的错误原因列在下方的窗口里（是英文的）

C的编译器给出的错误提示往往不那么好“猜”

1.3.3 做点计算：如何让程序输出算术结果，顺便说说第一周的练习

（1）做计算

printf("%d\n", 23+43);

%d说明后面有一个整数要输出在这个位置上

printf("23+43=%d\n", 23+43);

（2）四则运算

|  |  |  |
| --- | --- | --- |
| 四则运算 | C符号 | 意义 |
| ＋ | + | 加 |
| － | - | 减 |
| × | \* | 乘 |
| ÷ | / | 除 |
|  | % | 取余 |
| () | () | 括号 |

（3）第一周的练习

在自己的计算机上安装好编程的软件

依次下载或拷贝课程网站上的四个程序

编译并运行：结果如何、有没有问题、尝试理解

1.3.4 Mac OS X如何在命令行编辑、编译和运行C程序

2. 计算

2.1 变量

2.1.1 变量定义：变量是做什么的，如何定义一个变量

（1）如何能在程序运行时输入那个数字，然后计算输出结果？

需要有地方放输入的数字

需要有办法输入数字

需要输入的数字能参与计算

（2）如何输入

输入也在终端窗口中

输入是以行为单位进行的，行的结束标志就是你按下了回车键，此前你的程序不会读到任何东西

（3）变量

int price = 0;

这一行定义了一个变量，变量的名字是price，类型是int，初始值是0

变量是一个保存数据的地方，当我们需要在程序里保存数据时，就需要一个变量来保存它

用一个变量保存了数据，它才能参加到后面的计算中

（4）变量定义

变量定义的一般形式是：<类型名称> <变量名称>;

int price;

int amount;

int price, amount;

（5）变量的名字

变量需要一个名字，变量的名字是一种“标识符”，意思是用它来识别这个和那个的不同的名字

标识符有标识符的构造规则。基本的原则是：标识符只能由字母、数字和下划线组成，数字不可以出现在第一个位置上，C语言的关键字（有的地方加它们保留字）不可以用做标识符。

（6）C语言的保留字

auto, break, case, char, const, continue, default, do, double, else, enum, extern, float, for, goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, undersigned, void, volatile, while, inline, restrict

2.1.3 变量赋值与初始化：a=b表示把b的值赋给a，而不是它们相等

（1）赋值与初始化

int price = 0;

这一行，定义了一个变量：变量的名字是price，类型是int，初始值是0

price=0是一个式子，这里的=是一个赋值运算符，表示将=右边的值赋给左边的变量

（2）赋值

和数学不同，a=b在数学中表示关系，即a和b的值一样；而在程序设计中，a=b表示要求计算机做一个动作：将b的值赋给a

关系是静态的，动作是动态的。在数学中，a=b和b=a是等价的，而在程序中，两者的意思完全相反。

（3）初始化

当赋值发生在定义变量的时候，就是变量的初始化

虽然C语言并没有强制要求所有的变量都在定义的地方做初始化，但是所有的变量在第一次被使用（出现在赋值运算符的右边）之前应该被赋值一次

如果没有初始化？——变量是一个很奇怪的数（变量在内存里，变量会获得所在内存原有的数据）

（4）变量初始化

<类型名称> <变量名称> = <初始值>;

int price = 0;

int amount = 100;

组合变量定义的时候，也可以在这个定义中单独给单个变量赋初值，如：int price = 0, amount = 100;

（5）表达式

“=”是赋值运算符，有运算符的式子就叫做表达式

price = 0;

change = 100 - price;

（6）变量类型

int price = 0;

这一行定义了一个变量，变量的名字是price，类型是int，初始值是0

C是一种有类型的语言，所有的变量在使用前必须定义或申明，所有的变量必须具有确定的数据类型

数据类型表示在变量中可以存放什么样的数据，变量中只能存放指定类型的数据，程序运行过程中也不能改变变量的类型

（7）第二个变量

int change = 100 – price; #C99的用法

定义了第二个变量change，并且做了计算

（8）ANSI C

只能在代码开头的地方定义变量

2.1.4 变量输入：如何让程序读入用户输入的数字，scanf

（1）读整数

scanf("%d", &price);

要求scanf这个函数读入下一个整数，读到的结果赋值给变量price

小心price前面的&

如果输入的不是整数会怎样？——给一个默认的0

2.1.5 常量vs变量：不变的量是常量，以及，我们要怎样输入两个变量的值

（1）常量

int change = 100 - price;

固定不变的数，是常数。直接写在程序里，我们称作直接量(literal)

更好的方式，是定义一个常量：const int AMOUNT = 100; #C99的用法

我们一般对一个const的变量名进行全大写（如AMOUNT），用来强调它是const

（2）const

const是一个修饰符，加在int的前面，用来给这个变量加上一个const（不变的）的属性

这个const的属性表示这个变量的值一旦初始化，就不能再修改了

int change = AMOUNT - price;

如果你试图对常量做修改，把它放在赋值运算符的左边，就会被编译器发现，指出为一个错误

（3）尝试

让用户输入变量AMOUNT的值，而不是使用固定的初始值

这个变量在哪里定义合适呢？

（4）plus.c

|  |
| --- |
| int a;  int b;  printf("请输入两个整数：");  scanf("%d %d", &a, &b);  printf("%d + %d = %d\n", a, b, a+b); |

2.1.6 浮点数：整数运算的结果只有整数部分，不然就要用浮点数

（1）身高5尺7寸？

美国人固执地使用英制计量单位，他们习惯用几尺几寸（英尺英寸）来报自己的身高

如果遇到一个美国人告诉你他是5英尺7，他的身高应该是一米几呢？——(5+7÷12)×0.3048=1.7018米

|  |
| --- |
| #include <stdio.h>  int main()  {  printf("请分别输入身高的英尺和英寸，"  "如输入\"5 7\"表示5英尺7英寸：");    int foot;  int inch;    scanf("%d %d", &foot, &inch);    printf("身高是%f米。\n",  ((foot + inch / 12) \* 0.3048));    return 0;  } |

10/3\*3=>?\_——9

（2）因为···

因为两个整数的运算的结果只能是整数

10和10.0在C中是完全不同的数

10.0是浮点数

|  |  |
| --- | --- |
| #include <stdio.h>  int main()  {  printf("%d\n", 10.0/3\*3);    return 0;  }  提示错误；应将%d改为%f | #include <stdio.h>  int main()  {  printf("%f\n", 10.0/3\*3);    return 0;  }  结果10.000000 |

（3）浮点数

带小数点的数值。

浮点这个词的本意就是指小数点是浮动的，是计算机内部表达非整数（包含分数和无理数）的一种方式。

另一种方式叫做定点数，不过在C语言中你不会遇到定点数。

|  |
| --- |
| #include <stdio.h>  int main()  {  printf("请分别输入身高的英尺和英寸，"  "如输入\"5 7\"表示5英尺7英寸：");    float foot;  float inch;    scanf("%f %f", &foot, &inch);    printf("身高是%f米。\n",  ((foot + inch / 12) \* 0.3048));    return 0;  } |

人们借用浮点数这个词来表达所有的带小数点的数。

（4）改进

(foot + inch / 12) \* 0.3048改为(foot + inch / 12.0) \* 0.3048

当浮点数和整数放到一起运算时，C会将整数转换成浮点数，然后进行浮点数的运算

（5）double

inch是定义为int类型的变量，如果把int换成double，我们就把它改为double类型的浮点数变量了

double的意思是“双”，它本来是“双精度浮点数”的第一个单词，人们用来表示浮点数类型。

除了double，还有float（意思就是浮点！）表示单精度浮点数

|  |
| --- |
| #include <stdio.h>  int main()  {  printf("请分别输入身高的英尺和英寸，"  "如输入\"5 7\"表示5英尺7英寸：");    double foot;  double inch;    scanf("%lf %lf", &foot, &inch);    printf("身高是%f米。\n",  ((foot + inch / 12) \* 0.3048));    return 0;  } |

（6）数据类型

整数

int

printf("%d", ...)

scanf("%d", ...)

带小数点的数

double

printf("%f", ...)

scanf("%lf", ...)

（7）整数

整数类型不能表达小数部分的数，整数和整数的运算结果还是整数

计算机里会有纯粹的整数这种奇怪的东西，是因为整数的运算比较快，而且占地方也小。

其实人们日常生活中大量做的还是纯粹整数的计算，所以整数的用处还是很大的。

2.1.n 小测验

（1）以下哪些是有效的变量名？

A.main

B.4ever

C.monkey-king

D.int

正确答案：A、D你错选为C

（2）给定：

int a,b;

scanf("%d %d", &a, &b);

则以下哪些输入方式是正确的？

A.1 2

B.1,2

C.1(回车)2

D.1、2

正确答案：A、C你选对了

（3）给定：

int a,b;

以下哪些scanf的使用是正确的？

A.scanf("%d", &a);

scanf("%d", &b);

B.scanf("%d %d", &a, &b);

C.scanf("%d, %d", &a, &b);

D.scanf("%d %d", a, b);

正确答案：A、B、C你错选为A、B

（4）给定以下代码段：

int a,b=0;

则a的初始值是0

A.√

B.×

正确答案：B你选对了

（5）写出下式的运算结果：

10/3.0\*3

正确答案：10.0

（6）写出下式的运算结果：

10/3\*3.0

正确答案：9.0

2.2 表达式

2.2.1 表达式：运算符和算子，取余计算，程序就是数据加计算

（1）表达式

一个表达式是一系列运算符和算子的组合，用来计算一个值

|  |
| --- |
| amount = x \* (1 + 0.033) \* (1 + 0.033) \* (1 + 0.033);  total = 57;  count = count + 1;  value = (min + 2) \* lastValue; |

（2）运算符

运算符(operator)是指进行运算的动作，比如加法运算符“+”，减法运算符“-”

算子(operand)是指参与运算的值，这个值可能是常数，也可能是变量，还可能是一个方法的返回值

（3）计算时间差

输入两个时间，每个时间分别输入小时和分钟的值，然后输出两个时间之间的差，也以几小时几分表示

|  |
| --- |
| int hour1, minute1;  int hour1, minute2;  scanf("%d %d", &hour1, &minute1);  scanf("%d %d", &hour2, &minute2); |

如果直接分别减，会出现分钟借位的情况：1点40分和2点10分的差？

|  |
| --- |
| int hour1, minute1;  int hour1, minute2;  scanf("%d %d", &hour1, &minute1);  scanf("%d %d", &hour2, &minute2);  int t1 = hour1 \* 60 + minute1;  int t2 = hour2 \* 60 + minute2;  int t = t2 - t1;  printf("时间差是%d小时%d分。", t/60, t%60); |

2.2.2 运算符优先级：优先级、结合关系、赋值运算符

（1）运算符优先级

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 优先级 | 运算符 | 运算 | 结合关系 | 举例 |
| 1 | + | 单目不变 | 自右向左 | a\*+b |
| 1 | - | 单目取负 | 自右向左 | a\*-b |
| 2 | \* | 乘 | 自左向右 | a\*b |
| 2 | / | 除 | 自左向右 | a/b |
| 2 | % | 取余 | 自左向右 | a%b |
| 3 | + | 加 | 自左向右 | a+b |
| 3 | - | 减 | 自左向右 | a-b |
| 4 | = | 赋值 | 自右向左 | a=b |

（2）赋值运算符

赋值也是运算，也有结果

a=6的结果是a被赋予的值，也就是6

a=b=6 -> a=(b=6)

（3）“嵌入式赋值”

|  |
| --- |
| int a = 6;  int b;  int c = 1 + (b = a) |

* 不利于阅读
* 容易产生错误

（4）结合关系

一般自左向右

单目+-和赋值=自右向左

|  |
| --- |
| result = a = b = 3 + c;  result = 2;  result = (result = result \* 2) \* 6 \* (result = 3 + result); |

* 这样的表达式太复杂，不容易阅读和理解，容易造成读程序时的误解。所以，要避免写出这样的复杂表达式来的。这个表达式应该被拆成若干个表达式，然后以明显的正确的顺序来进行计算。

（5）计算复利

在银行存定期的时候，可以选择到期后自动转存，并将到期的利息计入本金合并转存。

如果1年期的定期利率是3.3%，那么连续自动转存3年后，最初存入的x元定期会得到多少本息余额？

——本息合计=x(1+3.3%)3

|  |
| --- |
| int x;  scanf("%d", &x);  double amount = x \* (1 + 0.033) \* (1 + 0.033) \* (1 + 0.033);  printf("%lf", amount); |

2.2.3 交换变量：如何交换两个变量的值，顺便看下Dev的调试功能

（1）交换两个变量

如果已经有

int a = 6;

int b = 5;

如何交换a、b两个变量的值？

|  |
| --- |
| int a = 5;  int b = 6;  int t;  t = a;  a = b;  b = t;  printf("a = %d, b = %d\n", a, b); |

2.2.4 复合赋值和递增递减：这是两类有历史也有争议的运算符

（1）复合赋值

5个算术运算符，+-\*、%，可以与赋值运算符=结合起来，形成复合赋值运算符：+=, -=, \*=, /=, %=

total += 5;即total = total + 5;

注意两个运算符中间不要有空格

total += (sum + 100) / 2;即total = total + (sum + 100) / 2;

total \*= sum + 12;即total = total \* (sum + 12);

total /= 12 + 6;即total = total / (12+6);

（2）递增递减运算符

++和—是两个很特殊的运算符，他们是单目运算符，这个算子还必须是变量

这两个运算符分别叫做递增和递减运算符，他们的作用就是给这个变量+1或者-1

count ++;即count += 1;或count = count + 1;

（3）前缀后缀

++和--可以放在变量的前面，叫做前缀形式，也可以放在变量的后面，叫做后缀形式

a++的值是a加1以前的值，而++a的值是加了1以后的值，无论哪个，a自己的值都加了1了

|  |  |  |
| --- | --- | --- |
| 表达式 | 运算 | 表达式的值 |
| count++ | 给count加1 | count原来的值 |
| ++count | 给count加1 | count+1以后的值 |
| count-- | 给count减1 | count原来的值 |
| --count | 给count减1 | count-1以后的值 |

举例

|  |
| --- |
| #include <stdio.h>  int main()  {  int a = 10;  printf("a++ = %d\n", a++);  printf("a = %d\n", a);    printf("++a = %d\n", ++a);  printf("a = %d\n", a);    return 0;  }  a++ = 10  a = 11  ++a = 12  a = 12 |

（4）++--

这两个运算符有其历史来源

可以单独使用，但是不要组合进表达式

++i-- -->?

i++++ -->?

a=b+=c++-d+--e/-f

2.2.5 如何使用PAT系统练习来做编程练习题

2.2.n 小测验

（1）写出以下代码执行后，t1和t2的值，以空格隔开：

int a=14;

int t1 = a++;

int t2 = ++a;

14 15

正确答案：

14 16

|  |
| --- |
| #include <stdio.h>  int main()  {    int a=14;  int t1 = a++;  int t2 = ++a;  printf("a=%d, t1=%d, t2=%d", a, t1, t2);    return 0;  }  a=16, t1=14, t2=16 |

（2）写出以下表达式的结果，一个结果一行：

6 + 5 / 4 - 2

2 + 2 \* (2 \* 2 - 2) % 2 / 3

10 + 9 \* ((8 + 7) % 6) + 5 \* 4 % 3 \* 2 + 3

1 + 2 + (3 + 4) \* ((5 \* 6 % 7 / 8) - 9) \* 10

5.2500

2

44

-609.5

正确答案：

5

2

44

-627

|  |
| --- |
| #include <stdio.h>  int main()  {  double a, b, c, d;    a = 6 + 5 / 4 - 2;  b = 2 + 2 \* (2 \* 2 - 2) % 2 / 3;  c = 10 + 9 \* ((8 + 7) % 6) + 5 \* 4 % 3 \* 2 + 3 ;  d = 1 + 2 + (3 + 4) \* ((5 \* 6 % 7 / 8) - 9) \* 10;  printf("a=%lf, b=%lf, c=%lf, d=%lf", a, b, c, d);    return 0;  }  a=5.000000, b=2.000000, c=44.000000, d=-627.000000 |

3. 判断

3.0 编程练习解析

3.1 判断

3.1.1 做判断：if语句根据条件决定做还是不做

（1）如果

if (条件成立){

…

}

3.1.2 判断的条件：关系运算，作比较的运算符

（1）条件

计算两个值之间的关系，所以叫做关系运算

|  |  |
| --- | --- |
| 运算符 | 意义 |
| == | 相等 |
| != | 不相等 |
| > | 大于 |
| >= | 大于或等于 |
| < | 小于 |
| <= | 小于或等于 |

（2）关系运算的结果

当两个值的关系符合关系运算符的预期时，关系运算的结果为整数1，否则为整数0

printf("%d\n", 5==3);

printf("%d\n", 5>3);

printf("%d\n", 5<=3);

（3）优先级

所有的关系运算符的优先级比运算的低，但是比赋值运算的高

7 >= 3 + 4

int r = a > 0;

判断是否相等的==和!=的优先级比其它的低，而连续的关系运算是从左到右进行的

5 > 3 == 6 > 4

6 > 5 > 4

a == b == 6

a == b > 0

3.1.3 找零计算器：判断，注释，流程图

（1）找零计算器

找零计算器需要用户做两个操作：输入购买的金额，输入支付的票面，而找零计算器则根据用户的输入做出相应的动作：计算并打印找零，或告知用户余额不足以购买

从计算机程序的角度看，这就意味着程序需要读用户的两个输入，然后进行一些计算和判断，最后输出结果

（2）注释

以两个斜杠//开头的语句把程序分成了三个部分：初始化；读入金额和票面；计算并打印找零

注释(comment)插入在程序代码中，用来向读者提供解释信息。它们对于程序的功能没有任何影响，但是往往能使程序更容易被人类读者理解。

（2）/\*\*/注释

延续数行的注释，要用多行注释的格式来写。多行注释由一对字符序列/\*开始，而以\*/结束。

也可以用于一行内的注释：int ak = 47 /\* 36 \*/, y = 9;

3.1.4 否则的话：如果条件不成立呢？

3.1.5 if语句再探：if和else后面也可以没有{}而是一条语句

一个基本的if语句由一个关键字if开头，跟上在括号里的一个表示条件的逻辑表达式，然后是一对大括号{}之间的若干语句。如果表示条件的逻辑表达式的结果不是零，那么执行后面跟着的这些大括号中的语句，否则跳过这些语句不执行，而继续下面的其他语句。

if语句这一行结束的时候并没有表示语句结束的;，而后面的赋值语句写在if的下一行，并且缩进了，在这一行结束的时候有一个表示语句结束的;。这表明这条赋值语句是if语句的一部分，if语句拥有和控制这条赋值赋值语句，决定它是否要被执行。

if (total > amount)

total += amount + 10；

3.2 分支

3.2.1 嵌套的if-else：在if或else后面要执行的还是if语句，就成了嵌套

（1）嵌套的判断

当if的条件满足或者不满足的时候要执行的语句也可以是一条if或if-else语句，这就是嵌套的if语句

|  |
| --- |
| if (code == READY)  if (count < 20)  print("一切正常\n");  else  printf("继续等待\n"); |

（2）else的匹配

else总是与最近的那个if匹配

|  |
| --- |
| if (code == READY) {  if (count < 20)  print("一切正常\n");  } else  printf("继续等待\n"); |

（3）缩进

缩进格式不能暗示else的匹配

|  |
| --- |
| if (code == READY)  if (count < 20)  print("一切正常\n");  else  printf("继续等待\n"); |

（4）嵌套的if

3.2.2 级联的if-else if：像分段函数这样的程序就可以写成级联的if

|  |
| --- |
| if (x < 0) {  f = -1;  } else if (x == 0) {  f = 0;  } else {  f = 2 \* x;  } |

（1）级联的if-else if

|  |
| --- |
| if (exp1)  st1;  else if (exp2)  st2;  else  st3; |

3.2.3 if-else的常见错误：没有{}、多了;、=和==，这些都是常见的错误

3.2.4 多路分支：switch-case语句

（1）switch-case

|  |  |
| --- | --- |
| if (type == 1)  printf("你好");  else if (type == 2)  print("早上好");  else if (type == 3)  print("晚上好");  else if (type == 4)  print("再见");  else  print("啊？什么啊？"); | switch (type) {  case 1:  printf("你好");  break;  case 2:  printf("早上好");  break;  case 3:  printf("晚上好");  break;  case 4:  printf("再见");  break;  default:  printf("啊？什么啊？");  } |

即

|  |
| --- |
| switch (控制表达式) {  case 常量：  语句  ......  case 常量：  语句  ......  default：  语句  ......  } |

控制表达式只能是整数型的结果

（2）break

switch语句可以看作是一种基于计算的跳转，计算控制表达式的值后，程序会跳转到相匹配的case（分支标号）处。分支标号知识说明switch内部位置的路标，在执行完分支中的最后一条语句后，如果后面没有break，就会顺序执行到下面的case里去，直到遇到一个break，或者switch结束为止。

4. 循环

4.1 循环

4.1.1 循环：有些事情就得用循环才能解决

（1）数数几位数

人的方式：眼睛一看就知道了——325->3位！

计算机的方式：判断数的范围来决定它的位数——352∈[100,999]->3位

（2）程序实现

|  |
| --- |
| int x;  int n = 1;  scanf("%d", &x);  if (x > 999) {  n = 4;  } else if (x > 99) {  n = 3;  } else if (x > 9) {  n = 2;  }  printf("%d\n", n) |

因为题目明确了4位数及以下的正整数，所以可以简化一些判断

因为从高处往下判断，所以不需要判断上限了

反过来不行

问题：任意范围的正整数怎么办？

（3）换个方式想

352->3很快

123812843267518273618273612675317是几位数？

（4）数数

人怎么数？从左往右数，一次划掉一个数字

计算机怎么划掉那个数字？

（5）三位数逆序的题

352

352%100->52

那么

123812843267518273618273612675317%100000000000000000000000000000000->23812843267518273618273612675317怎么得到那个100000000000000000000000000000000？

（6）人vs计算机

如果换一下，从右边开始划

123812843267518273618273612675317/10->12381284326751827361827361267531

去掉最右边的数。然后？——不断的划，直到没数可以划

在这个过程中计数

（7）试试代码

|  |
| --- |
| int x;  int n = 0;  scanf("%d", &x);  n++;  x /= 10;  while (x > 0) {  n++;  x /= 10;  }  printf("%d\n", n); |

4.1.2 while循环：就像if一样，条件满足就不断地做后面大括号里的句子

（1）while循环

如果我们把while翻译为“当”，那么一个while循环的意思就是：当条件满足时，不断地重复循环体内的语句

循环执行之前判断是否继续循环，所以有可能循环一次也没有被执行

条件成立是循环成立的条件

循环体内要有改变条件的机会，否则循环会出不来——死循环

（2）再想想

如果没有外面的运算？还对不对？

|  |
| --- |
| int x;  int n = 0;  scanf("%d", &x);  while (x > 0) {  n++;  x /= 10;  }  printf("%d\n", n); |

（3）看程序运行结果

人脑模拟计算机的运行，在纸上列出所有的变量，随着程序的发展不断重新计算变量的值。

当程序运行结束时，留在表格最下面的就是程序的最终结果。

（4）验证

测试程序使用边界数据，如有效范围两端的数据、特属于的倍数等——个位数；10；0；负数

在程序适当的地方插入printf来输出变量的内容

4.1.3 do-while循环：不管三七二十一，先做循环内的句子再判断条件

（1）do-while循环

在进入循环的时候不做检查，而是在执行完一轮循环体的代码之后，再来检查循环的条件是否满足，如果满足则继续下一轮的循环，不满足则结束循环

do

{

<循环体语句>

}

while (<循环条件>)

{

<循环体语句>

}

（2）两种循环

do-while循环和while循环很想，区别是在循环体执行结束的时候才来判断条件。也就是说，无论如何，循环都会执行至少一遍，条件满足时执行循环，条件不满足时结束循环。

|  |
| --- |
| int x;  scanf("%d", &x);  int n = 0;  do  {  x /= 10;  n++;  } while (x > 0);  printf("%d", n); |

4.2 循环应用

4.2.1 循环计算

（1）log2x

|  |
| --- |
| int x;  int ret = 0;  scanf("%d", &x);  while(x > 1) {  x /= 2;  ret++;  }  printf("log2 of %d is %d.", x, ret); |

printf()中的x改变了，怎么办？

（2）小套路

|  |
| --- |
| int x;  int ret = 0;  scanf("%d", &x);  int t = x;  while(x > 1) {  x /= 2;  ret++;  }  printf("log2 of %d is %d.", t, ret); |

（3）计数循环

|  |
| --- |
| int count = 100;  while (count >= 0) {  count --;  printf("%d\n", count);  }  printf("发射！\n"); |

这个循环需要执行多少次？

循环停下来的时候，有没有输出最后的0？

循环结束后，count的值是多少？

4.2.2 猜数游戏

（1）猜数游戏

让计算机来来想一个数，然后让用户来猜，用户每输入一个数，就告诉它是大了还是小了，直到用户猜中为止，最后还要告诉用户猜了几次。

因为需要不断重复让用户猜，所以需要用到循环

在实际写出程序之前，我们可以先用文字描述程序的思路

核心的重点是循环的条件，人们往往会考虑循环终止的条件

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  int main()  {  srand(time(0));  int number = rand()%100+1;  int count = 0;  int a = 0;  printf("我已经想好了一个1到100之间的数。");  do {  printf("猜猜这个1到100之间数：");  scanf("%d", &a);  count ++;  if (a > number) {  printf("你猜的数大了。");  } else if (a < number) {  printf("你猜的数小了。");  }  } while (a != number);  printf("太好了，你用了%d次就猜到了答案。\n", count);  return 0;  } |

（2）随机数

每次召唤rand()就得到一个随机的整数

（3）%100

x%n的结果是[0,n-1]的一个整数

4.2.3 算平均数

（1）算平均数

让用户输入一系列的正整数，最后输入-1表示输入结束，然后程序计算出这些数字的平均数，输出输入的数字的个数和平均数

变量->算法->流程图->程序

（2）变量

一个记录读到的整数的变量

平均数要怎么算？

只需要每读到一个数，就把它加到一个累加的变量里，到全部数据读完，再拿它去除读到的数的个数就可以了。

一个变量记录累加的结果，一个变量记录读到的数的个数

|  |  |
| --- | --- |
| int number;  int sum = 0;  int count = 0;  scanf("%d", &number);  while (number != -1) {  sum += number;  count ++;  scanf("%d", &number);  }  printf("%f\n", 1.0\*sum/count); | int number;  int sum = 0;  int count = 0;  do {  scanf("%d", &number);  if (number != -1) {  sum += number;  count ++;  }  } while (number != -1);  printf("%f\n", 1.0\*sum/count); |

4.2.4 整数求逆

（1）整数的分解

一个整数是由一至多位数字组成的，如何分解出整数的各个位上的数字，然后加以计算

对一个整数做%10的操作，就得到了它的个位数；

对一个整数做/10的操作，就去掉了它的个位数；

然后再对2的结果做%10，就得到原来数的十位数了；

依此类推

（2）数的逆序

输入一个正整数，输出逆序的数

结尾的0的处理

|  |
| --- |
| #include <stdio.h>  int main()  {  int x;  scanf("%d", &x);  int digit;  int ret = 0;    while (x > 0) {  digit = x % 10;  ret = ret \* 10 + digit;  printf("x=%d, digit=%d, ret=%d\n", x, digit, ret);  x /= 10;  }  printf("%d", ret);    return 0;  } |

5. 循环控制

5.1 第三种循环

5.1.1 for循环：这是最古老的循环，确实样子看上去有点古怪

（1）阶乘

写一个程序，让用户输入n，然后计算输出n!

变量：显然读用户的输入需要一个int的n，然后计算的结果需要用一个变量保存，可以是int的factor，在计算中需要有一个变量不断地从1递增到n，那可以是int的i

|  |  |
| --- | --- |
| int n;  scanf("%d", &n);  int fact = 1;  int i = 1;  while (i <= n) {  fact \*= i;  i ++;  }  printf("%d! = %d\n", n, fact); | int n;  scanf("%d", &n);  int fact = 1;  int i = 1;  for (i=1; i<=n; i++) {  fact \*= i;  }  printf("%d! = %d\n", n, fact); |

（2）for循环

for循环像一个计数循环：设定一个计数器，初始化它，然后在计算器到达某值之前，重复执行循环体，而每执行一轮循环，计数器值以一定步进进行调整，比如加1或者减1

（3）for=对于

for (count=10; count>0; count--)就读成：“对于一开始的count=10，当count>0时，重复做循环体，每一轮循环在做完循环体内语句后，使得count--。”

（4）小套路

做求和的程序时，记录结果的变量应该初始化为0，而做求积的变量时，记录结果的变量应该初始化为1

循环控制变量i只在循环里被使用了，在循环外面它没有任何用处。因此我们可以把变量i的定义写到for语句里面去

|  |
| --- |
| int n;  scanf("%d", &n);  int fact = 1;  for (int i=1; i<=n; i++) {  fact \*= i;  }  printf("%d! = %d\n", n, fact); |

这种方法只有C99能用，而且即使宣称支持C99的编译器也需要加特殊选项才能使用

（5）尝试

还是1，所以程序的循环不需要从1开始，那么改成从多少合适呢？这样修改后，程序对所有的n都正确吗？这样的改动有价值吗？

除了可以从1乘到n来计算n!，还可以从n乘到1来计算吧？试试把循环换个方向来计算n。到时候，还需要循环控制变量i吗？

for循环的三个表达式可以省掉那么一个的，但是它的分号不能删

|  |  |  |
| --- | --- | --- |
| for (i=n; i>1; i--)  fact \*= n; | for (n=n; n>1; n--)  fact \*= n; | for (; n>1; n--)  fact \*= n; |

5.1.2 循环的计算和选择：如何计算循环的次数，如何选择不同的循环

（1）循环次数

for (i=0; i<n; i++)

则循环的次数是n，而循环结束以后，i的值是n。循环的控制变量i，是选择从0开始还是从1开始，是判断i<n还是判断i<=n，对循环的次数，循环结束后的变量的值都有影响。

（2）for == while

|  |  |
| --- | --- |
| for (int i=1; i<=n; i++) {  fact \*= i; | int i = 1;  while (i<=n) {  fact \*= i;  i ++;  } |

（3）for循环

for (初始动作;条件;每轮的动作) {

}

for中的每一个表达式都是可以忽略的，

for (; 条件; ) == while (条件)

（4）tips for loops

如果有固定次数，用for

如果必须执行一次，用do-while

其他情况用while

5.2 循环控制

5.2.1 循环控制：如何用break和continue来控制循环

（1）素数

只能被1和自己整除的数，不包括1：2，3，5，7，11，13，17，19······

（注意：以下程序都未排除输入x=1时的问题）

|  |
| --- |
| #include <stdio.h>  int main()  {  int x;    scanf("%d", &x);    int i;  int isPrime = 1;  for (i=2; i<x; i++) {  if (x % i == 0) {  isPrime = 0;  }  }  if (isPrime == 1) {  printf("是素数\n");  } else {  printf("不是素数\n");  }    return 0;  } |

（2）break vs continue

break：跳出循环

continue：跳过循环这一轮剩下的语句进入进入下一轮

|  |  |
| --- | --- |
| #include <stdio.h>  int main()  {  int x;    scanf("%d", &x);    int i;  int isPrime = 1;  for (i=2; i<x; i++) {  if (x % i == 0) {  isPrime = 0;  break;  }  }  if (isPrime == 1) {  printf("是素数\n");  } else {  printf("不是素数\n");  }    return 0;  } | #include <stdio.h>  int main()  {  int x;    scanf("%d", &x);    int i;  int isPrime = 1;  for (i=2; i<x; i++) {  if (x % i == 0) {  isPrime = 0;  continue;  }  printf("%d\n", i);  }  if (isPrime == 1) {  printf("是素数\n");  } else {  printf("不是素数\n");  }    return 0;  } |

5.2.2 嵌套的循环：在循环里面还是循环

（1）100以内的素数

如何写程序输出100以内的素数？

|  |
| --- |
| #include <stdio.h>  int main()  {  int x;    for (x=1; x<=100; x++) {  int i;  int isPrime = 1;  for (i=2; i<x; i++) {  if (x % i == 0) {  isPrime = 0;  break;  }  }  if (isPrime == 1) {  printf("%d ", x);  }  }    return 0;  } |

（2）嵌套的循环

循环里面还是循环

（3）前50个素数，每5的一行，对齐排列

|  |
| --- |
| #include <stdio.h>  int main()  {  int x;  int cnt = 0;    for (x=1; cnt<50; x++) {  int i;  int isPrime = 1;  for (i=2; i<x; i++) {  if (x % i == 0) {  isPrime = 0;  break;  }  }  if (isPrime == 1) {  cnt ++;  printf("%d\t", x);  if (cnt % 5 == 0) {  printf("\n");  }  }  }    return 0;  } |

5.2.3 从嵌套的循环中跳出：break只能跳出其所在的循环

（1）凑硬币

如何用1角、2角和5和角的硬币凑出10元以下的金额呢？

|  |
| --- |
| #include <stdio.h>  int main()  {  int x;  int one, two, five;  scanf("%d", &x);  for (one=1; one<x\*10; one++) {  for (two=1; two<x\*10/2; two++) {  for (five=1; five<x\*10/5; five++) {  if (one + two\*2 + five\*5 == x\*10) {  printf("可以用%d个1角加%d个2角加%d个5角得到%d元\n", one, two, five, x);  }  }  }    }  return 0;  } |

如果只需要一种硬币组合呢？——用break？

（2）break和continue

只能对它所在的那层循环做

（3）接力break

|  |
| --- |
| #include <stdio.h>  int main()  {  int x;  int one, two, five;  int exit = 0;  scanf("%d", &x);  for (one=1; one<x\*10; one++) {  for (two=1; two<x\*10/2; two++) {  for (five=1; five<x\*10/5; five++) {  if (one + two\*2 + five\*5 == x\*10) {  printf("可以用%d个1角加%d个2角加%d个5角得到%d元\n", one, two, five, x);  exit = 1;  break;  }  }  if (exit == 1) break;  }  if (exit == 1) break;    }  return 0;  } |

（4）goto

|  |
| --- |
| #include <stdio.h>  int main()  {  int x;  int one, two, five;  scanf("%d", &x);  for (one=1; one<x\*10; one++) {  for (two=1; two<x\*10/2; two++) {  for (five=1; five<x\*10/5; five++) {  if (one + two\*2 + five\*5 == x\*10) {  printf("可以用%d个1角加%d个2角加%d个5角得到%d元\n", one, two, five, x);  goto out;  }  }  }    }  out:  return 0;  } |

5.3 循环应用

5.3.1 前n项求和

（1）

|  |
| --- |
| #include <stdio.h>  int main()  {  int n;  int i;  double sum = 0.0;    scanf("%d", &n);  for (i=1; i<=n; i++) {  sum += 1.0/i;  }  printf("f(%d)=%f\n", n, sum);    return 0;  } |

（2）

|  |
| --- |
| #include <stdio.h>  int main()  {  int n;  int i;  double sum = 0.0;  double sign = 1.0;    scanf("%d", &n);  for (i=1; i<=n; i++) {  sum += sign/i;  sign = -sign;  }  printf("f(%d)=%f\n", n, sum);    return 0;  } |

5.3.2 整数分解

（1）正序分解分解整数

输入输入一个非负整数，正序输出它的每一位数字

输入：13425

输出：1 3 4 2 5

|  |
| --- |
| #include <stdio.h>  int main()  {  int x;  scanf("%d", &x);    int mask = 1;  int t = x;  while (t>9) {  t /= 10;  mask \*= 10;  }  printf("x=%d, mask=%d\n", x, mask);  do {  int d = x / mask;  printf("%d", d);  if (mask > 9) {  printf(" ");  }  x %= mask;  mask /= 10;  } while (mask > 0);  printf("\n");  } |

5.3.3 求最大公约数

（1）枚举

step1. 设t为2

step 2. 如果u和v都能被t整除，则记下这个t

step 3. t加1后重复第2步，直到t等于u或v

step 4. 那么，曾经记下的最大的可以同时整除u和v的t就是gcd

|  |
| --- |
| #include <stdio.h>  int main()  {  int a, b;  int min;    scanf("%d %d", &a, &b);  if (a<b) {  min = a;  } else {  min = b;  }  int ret = 0;  int i;  for (i=1; i<min; i++) {  if (a%i == 0) {  if (b%i == 0) {  ret = i;  }  }  }  printf("%d和%d的最大公约数是%d。\n", a, b, ret);    return 0;  } |

（2）辗转相除法

如果b等于0，计算结束，a就是最大公约数

否则，计算a除以b的余数，让a等于b，而b等于那个余数

回到第一步

|  |
| --- |
| #include <stdio.h>  int main()  {  int a, b;  int t;    scanf("%d %d", &a, &b);  while (b != 0) {  t = a%b;  a = b;  b = t;  }  printf("gcd=%d\n", a);    return 0;  } |

6. 数据类型

6.1 数据类型

6.1.1 数据类型：C语言有哪些基础数据类型，sizeof可以做什么

（1）C是有类型的语言

C语言的变量，必须：在使用前定义，并且确定类型

C以后的语言法发展方向：

C++、Java更强调类型，对类型的检查更严格

JavaScript、Python、PHP不看重类型，甚至不需要事先定义

（2）类型安全

支持强类型的观点认为明确的类型有助于尽早发现程序中的简单错误

反对强类型的观点认为过于强调类型迫使程序员面对底层、实现而非事务逻辑

总的来说，早期语言强调类型，面向底层的语言强调类型

C语言需要类型，但是对类型的安全检查并不足够

（3）C语言的类型（下划线的是C99的类型）

整数：char、short、int、long、long long

浮点数：float、double、long double

逻辑：bool

指针

自定义类型

（4）类型有何不同

类型名称：int、long、double

输入输出时的格式化：%d、%ld、%lf

所表达的数的范围：char < short < int < float < double

内存中所占据的大小：1个字节到16个字节

内存中的表达形式：二进制数（补码）、编码

（5）sizeof

是一个运算符，可以给出某个类型或变量在内存中所占据的字节数：sizeof(int)、sizeof(i)

是静态运算符，它的结果在编译时刻就决定了

不要在sizeof的括号里做运算，这些运算是不会做的

6.1.2 整数类型：除了int，还有多少整数类型

|  |
| --- |
| #include <stdio.h>  int main()  {  printf("sizeof(char)=%ld\n", sizeof(char));  printf("sizeof(short)=%ld\n", sizeof(short));  printf("sizeof(int)=%ld\n", sizeof(int));  printf("sizeof(long)=%ld\n", sizeof(long));  printf("sizeof(long long)=%ld\n", sizeof(long long));    return 0;  } |

（1）整数

char：1字节（8比特）

short：2字节

int：取决于编译器（CPU），通常的意义是“1个字”

long：取决于编译器（CPU），通常的意义是“1个字”

long long：8字节

6.1.3 整数的内部表达：整数是如何表达的，尤其是负数如何表达

（1）整数的内部表达

计算机内部一切都是二进制

18 -> 00010010

0 -> 00000000

-18 -> ?

（2）二进制负数

1个字节可以表达的数：00000000——11111111（0——255）

三种方案：

①仿照十进制，有一个特殊的标志表示负数

②取中间的数为0，如10000000表示0，比它小的是负数，比它大的是整数

③补码

（3）补码

考虑-1，我们希望-1+1=0，如何能做到？

0 ——> 00000000

1 ——> 00000001

11111111+00000001 ——> 100000000

由0 - 1 ——> -1，则-1 = (1)00000000 – 00000001 ——> 11111111

11111111被当做纯二进制看待时，是255，被当做补码看待时是-1

同理，对于-a，其补码就是0-a，实际是2n-a，n是这种类型的位数

Tips：tips：补码的意义就是拿补码和原码可以加出一个溢出的“零”

6.1.4 整数的范围：如何推算整数类型所能表达的数的范围，越界了会怎样

（1）数的范围

对于一个字节（8位），可以表达的是：00000000 – 11111111

其中：00000000 ——> 0 ；11111111~10000000 ——> -1~-128；00000001~01111111 ——> 1~127

|  |
| --- |
| #include <stdio.h>  int main()  {  char c = 255;  int i = 255;  printf("c=%d, i=%d\n", c, i);    return 0;  }  c=-1, i=255 |

char：1字节-128~127

short：2字节-32768~32767

int：取决于编译器(CPU)，通常的意义是“1个字”

long：4字节

long long：8字节

（2）补码

考虑-1，我们希望-1+1->0。如何能做到？——0->00000000；1->00000001；11111111+00000001->100000000

因为0-1->-1，所以-1=(1)00000000-00000001->11111111

11111111被当做纯二进制看待时，是255，被当做补码看待时是-1

同理，对于-a，其补码就是0-a，实际是2n-a，n是这种类型的位数

（3）unsigned

如果一个字面量常数想要表达自己是unsigned，可以在后面加u或U：255U

用l或L表示long(long)

unsigned的初衷并非扩展数能表达的范围，而是为了做纯二进制运算，主要是为了移位

（4）整数越界

整数是以纯二进制方式进行计算的，所以：

11111111+1->100000000->0

01111111+1->10000000->-128

10000000-1->01111111->127

6.1.5 整数的格式化：如何格式化输入输出整数，如何处理8进制和16进制

（1）整数的输入输出

只有两种形式：int或long long

%d: int

%u: undersigned

%ld: long long

%lu: undersigned long long

（2）8进制和16进制

一个以0开始的数字字面量是8进制

一个以0x开始的数字字面量是16进制

%o用于8进制，%x用于16进制

8进制和16进制只是如何把数字表达为字符串，与内部如何表达数字无关

16进制很适合表达二进制数据，因为4位二进制正好是一个16进制位

8进制的移位数字正好表达3位二进制，因为早期计算机的字长是12的倍数，而非8

6.1.6 选择整数类型：没什么特殊需要就只用int就好了

（1）选择整数类型

为什么整数要有那么多种？——为了准确表达内存，做底层内存的需要

没有特殊需要，就选择int——①现在的CPU的字长普遍是32位或64位，一次内存读写就是一个int，一次计算也是一个int，选择更短的类型不会更快，甚至可能更慢；②现代编译器一般会设计内存对齐，所以更短的类型实际在内存中有可能也占据一个int的大小（虽然sizeof告诉你更小）

unsigned与否只是输出的不同，内部计算是一样的

6.1.7 浮点类型：double和float，它们是什么，如何输入输出

（1）浮点类型

|  |  |  |  |
| --- | --- | --- | --- |
| 类型 | 字长 | 范围 | 有效数字 |
| float | 32 | ±(1.20×10-38~3.40×1038), 0, ±inf,nan | 7 |
| double | 64 | ±(2.2×10-308~1.79×10308), 0, ±inf,nan | 15 |

（2）浮点的输入输出

|  |  |  |
| --- | --- | --- |
| 类型 | scanf | printf |
| float | %f | %f, %e |
| double | %lf | %f, %e |

（3）科学计数法

-5.67E+16

“-”——可选的+或-符号

“.”——小数点也是可选的

“E”——可以用e或E

“+”——符号可以是-或+也可以省略（表示+）

“-5.67E+16”——整个词不能有空格

（3）输出精度

在%和f之间加上.n可以指定输出小数点后几位，这样的输出是做四舍五入的

printf("%.3f\n", -0.0049);

printf("%.30f\n", -0.0049);

printf("%.3f\n", -0.0049);

6.1.8 浮点的范围与精度：浮点数到底能表达那些数

（1）超过范围的浮点数

printf输出inf表示超过范围的浮点数：±∞

printf输出nan表示不存在的浮点数

（2）浮点运算的精度

|  |
| --- |
| #include <stdio.h>  int main()  {  float a, b, c;  a = 1.345f;  b = 1.123f;  c = a + b;  if (c == 2.468)  printf("相等\n");  else  printf("不相等！c=%.10f, 或%f\n", c, c);    return 0;  } |

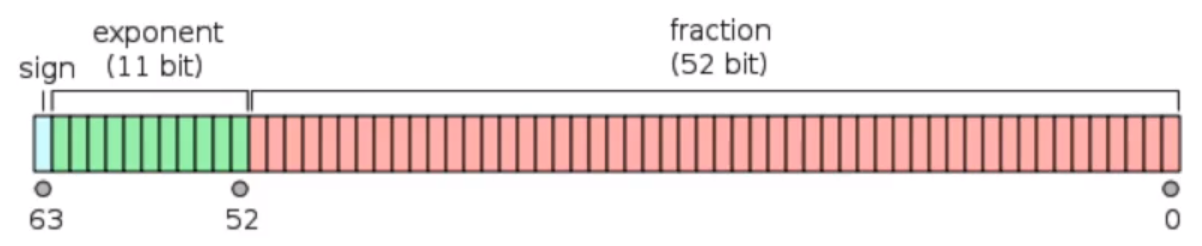
带小数点的字面量是double而非float

float需要用f或F后缀来表明身份

f1 == f2可能失败

fabs(f1-f2) < le-l2

（3）浮点数的内部表达



浮点数在计算时是由专用的硬件部件实现的

计算double和float所用的部件是一样的

（4）选择浮点类型

如果没有特殊需要，只使用double

现代CPU能直接对double做硬件运算，性能不会比float差，在64位的机器上，数据存储的速度也不比float慢

6.1.9 字符类型：char是整数也是字符

（1）字符类型

char是一种整数，也是一种特殊的类型——字符。这是因为

用单引号表示的字符字面量：'a','1'

''也是一个字符

printf和scanf里用%c来输入输出字符

（2）字符的输入输出

如何输入'1'这个字符给char c？

scanf("%c", &c); -> 1

scanf("%d", &i); c=i; -> 49

'1'的ASCII编码是49，所以当c==49时，它代表'1'

printf("%i %c\n", c, c);

一个49各自表述！

（3）混合输入

scanf("%d %c", &i, &c);

scanf("%d%c", &i, &c);

有何不同？

6.1.10 逃逸字符：反斜杠开头的字符是特殊的控制字符

（1）逃逸字符

用来表达无法印出来的控制字符或特殊字符，它由一个反斜杠“\”开头，后面跟上另一个字符，这两个字符合起来，组成了一个字符

printf("请分别输入身高的英尺和英寸，"

"如输入\"5 7\"表示5英尺7英寸：") ;

|  |  |
| --- | --- |
| 字符 | 意义 |
| \b | 回退一格 |
| \t | 到下一个表格位 |
| \n | 换行 |
| \r | 回车 |
| \" | 双引号 |
| \' | 单引号 |
| \\ | 反斜杠本身 |

（2）回车换行

源自打字机的操作

6.1.11 类型转换：如何在不同类型之间做转换

（1）自动类型转换

当运算符的两边出现不一致的类型时，会自动转换成较大的类型

大的意思是能表达的数的范围更大

char->short->int->long->long long

int->float->double

对于printf，任何小于int的类型都会被转换成int；float会被转换成double

但是scanf不会，要输入short，需要%hd

（2）强制类型转换

要把一个量强制转换成另一个类型（通常是较小的类型），需要：

(类型)值，如：(int)10.2, (short)32

注意这时候的安全性，小的变量不总能表达打的量：(short)32768

只是从那个变量计算出了一个新的类型的值，它并不改变那个变量，无论是值还是类型都不改变

6.2 其他运算：逻辑、条件、逗号

6.2.1 逻辑类型：表示关系运算和逻辑运算结果的量

（1）bool

#include <stdbool.h>

之后就可以使用bool和true、false

6.2.2 逻辑运算：对逻辑量进行与、或、非运算

（1）逻辑运算

逻辑运算是对逻辑量进行的运算，结果只有0或1

逻辑量是关系运算运算或逻辑运算的结果

|  |  |  |  |
| --- | --- | --- | --- |
| 运算符 | 描述 | 示例 | 结果 |
| ! | 逻辑非 | !a | 若a是true结果就是false；若a是false结果就是true |
| && | 逻辑与 | a && b | 若a和b都是true，结果就是true；否则就是false |
| || | 逻辑或 | a || b | 若a和b有一个是true，结果为true；两个都是false，结果为false |

（2）尝试

如果要表达数学中的区间，如：或，应该如何写C的表达式？——

像4<x<6这样的式子，不是C能正确计算的式子，因为4<x的结果是一个逻辑值（0或1）

如何判断一个字符c是否是大写字母？——

（3）理解一下

age>20 && age<30

index<0 || index>99

!age<20

（4）优先级

! > && > ||

!done && (count>MAX)

|  |  |  |
| --- | --- | --- |
| 优先级 | 运算符 | 结合性 |
| 1 | () | 从左到右 |
| 2 | ! + - ++ -- | 从右到左（单目的+和-） |
| 3 | \* / % | 从左到右 |
| 4 | + - | 从左到右 |
| 5 | < <= > >= | 从左到右 |
| 6 | == != | 从左到右 |
| 7 | && | 从左到右 |
| 8 | || | 从左到右 |
| 9 | = += -= \*= /= %= | 从右到左 |

代码换写

|  |  |
| --- | --- |
| if (gameover == 0) {  if (player2move == 2) {  printf("Your turn\n");  }  } | if (gameover == 0 && player2move == 2) {  printf("Your turn\n");  } |

（5）短路

逻辑运算是自左向右运行的，如果左边的结果已经已经能决定结果了，就不会做右边的计算

a==6 && b==1

a==6 && b+=1

对于&&，左边是false时就不做右边了

对于||，左边是true时就不做右边了

不要把赋值，包括复合赋值组合进表达式！

6.2.3 条件运算与逗号运算

（1）条件运算符

count = (count > 20) ? count - 10: count + 10;

条件、条件满足时的值和条件不满足时的值

|  |
| --- |
| if (count>20)  count = count - 10;  else  count = count + 10; |

（2）优先级

条件运算符的优先级高于赋值运算符，但是低于其他运算符

m<n ? x : a+5

a++ >= 1 && b-- > 2 ? a : b

x=3 \* a > 5 ? 5 ； 20

（3）嵌套条件表达式

count = (count > 20) ? (count < 50) ? count - 10 : count - 5 : (count < 10) ? count + 10 : count + 5

条件运算符是自右向左结合的

w < x ? x + w : x < y ? x : y

我们不希望你使用嵌套的条件表达式

（4）逗号运算

逗号用来连接两个表达式，并以其右边的表达式的值作为它的结果。逗号的优先级是所有的运算符中最低的，所以它两边的表达式会先计算；逗号的组合关系是自左向右，所以左边的表达式会先计算，而右边的表达式的值就留下来作为逗号运算的结果。如：i = (3+4, 5+6);

（5）在for中使用逗号运算：for (i=0, j=10; i<j; i++, j--) ...

7. 函数

7.1 函数的定义与使用

7.1.1 初见函数

（1）素数求和

|  |  |
| --- | --- |
| #include <stdio.h>  int main()  {  int m, n;  int sum = 0;  int cnt = 0;  int i;    scanf("%d %d", &m, &n);  if (m==1) m=2;  for (i=m; i<=n; i++) {  int isPrime = 1;  int k;  for (k=2; k<i-1; k++) {  if (i%k == 0) {  isPrime = 0;  break  }  }  if (isPrime) {  sum += i;  cnt++;  }  }  printf("%d %d\n", cnt, sum);    return 0;  } | #include <stdio.h>  int isPrime(int i)  {  int ret = 1;  int k;  for (k=2; k<i-1; k++) {  if (i%k == 0) {  ret = 0;  break;  }  }  return ret;  }    int main()  {  int m, n;  int sum = 0;  int cnt = 0;  int i;    scanf("%d %d", &m, &n);  if (m==1) m=2;  for (i=m; i<=n; i++) {  if (isPrime(i)) {  sum += i;  cnt++;  }  }  printf("%d %d\n", cnt, sum);    return 0;  } |

（2）求和

求出1到10、20到30和35到45的三个和

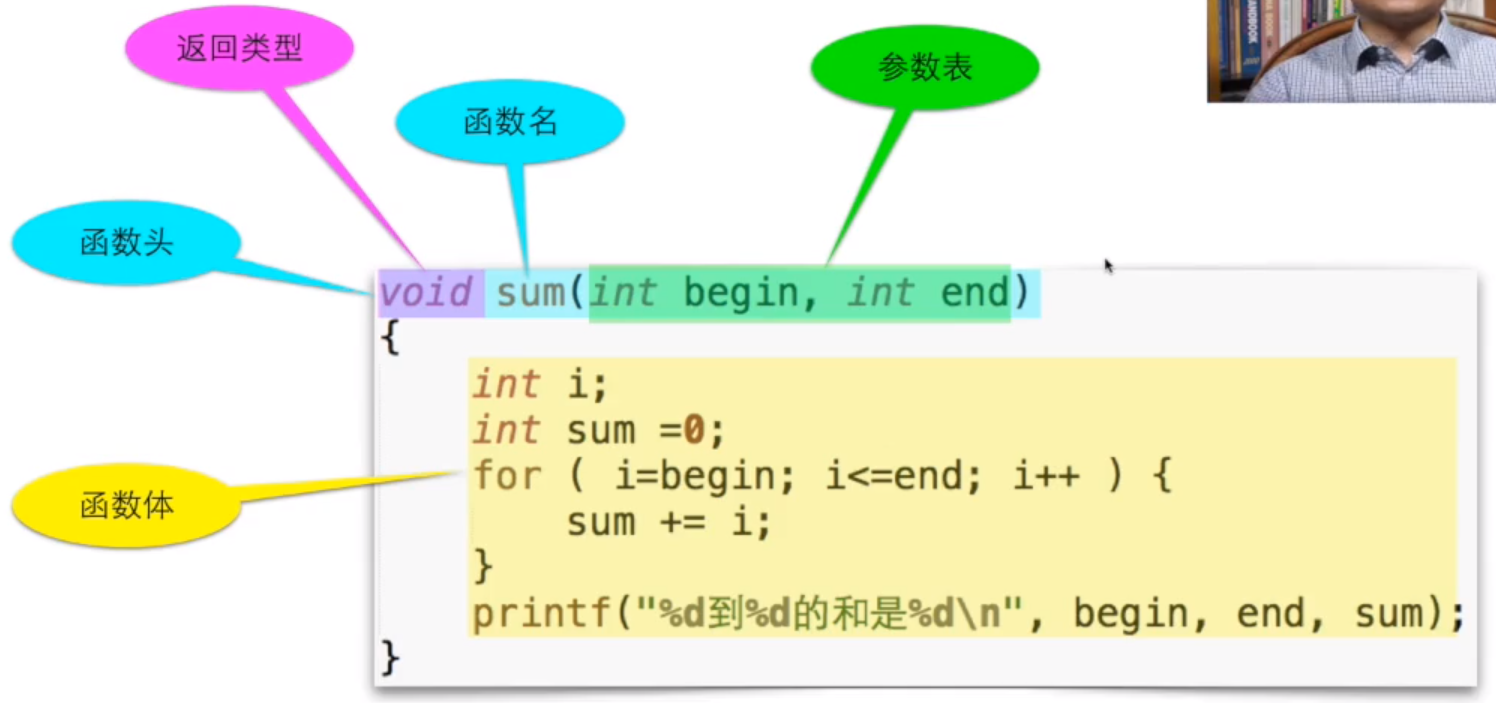
|  |  |
| --- | --- |
| #include <stdio.h>  int main()  {  int i;  int sum;    for (i=1, sum=0; i<=10; i++) {  sum += i;  }  printf("%d到%d的和是%d\n", 1, 10, sum);    for (i=20, sum=0; i<=30; i++) {  sum += i;  }  printf("%d到%d的和是%d\n", 20, 30, sum);    for (i=35, sum=0; i<=45; i++) {  sum += i;  }  printf("%d到%d的和是%d\n", 35, 45, sum);    return 0;  } | #include <stdio.h>  void sum(int begin, int end)  {  int i;  int sum;  for (i=begin; i<=end; i++) {  sum += i;  }  printf("%d到%d的和是%d\n", begin, end, sum);  }  int main()  {  sum(1, 10);  sum(20, 30);  sum(35, 45);    return 0;  } |

Tips：“代码复制”是程序不良的表现

7.1.2 函数的定义与调用

（1）什么是函数？

函数是一块代码，接收零个或多个参数，做一件事，并返回零个或一个值



（2）调用函数

函数名(参数值)

()起到了表示函数调用的重要作用，即使没有参数也需要()

7.1.3 从函数中返回

（1）从函数中返回值

return停止函数的执行，并送回一个值

return;

return 表达式;

（2）没有返回值返回值额的函数

void函数名(参数表)

不能使用带值的return，可以没有return

调用的时候不能做返回值的赋值

Tips：如果函数有返回值，则必须使用带值的return

7.2 函数的参数和变量

7.2.1 函数原型：用来告诉编译器这个函数长什么样儿

（1）函数先后关系

|  |
| --- |
| #include <stdio.h>  void sum(int begin, int end)  {  int i;  int sum;  for (i=begin; i<=end; i++) {  sum += i;  }  printf("%d到%d的和是%d\n", begin, end, sum);  }  int main()  {  sum(1, 10);  sum(20, 30);  sum(35, 45);    return 0;  } |

像这样把sum()写在上面，是因为：

C的编译器自上而下顺序分析你的代码

在看到sum(1, 10)的时候，它需要知道sum()的样子，也就是sum()要几个参数，每个参数的类型如何，返回什么类型

（2）如果不知道

也就是把要调用的函数放到下面了

|  |
| --- |
| #include <stdio.h>  void sum(int begin, int end); // 声明 函数原型  int main()  {  sum(1, 10); // 根据原型判读判断  sum(20, 30);  sum(35, 45);    return 0;  }  void sum(int begin, int end) // 定义 实际的函数头  {  int i;  int sum;  for (i=begin; i<=end; i++) {  sum += i;  }  printf("%d到%d的和是%d\n", begin, end, sum);  } |

（3）函数原型

函数头，以分号结尾，就构成了函数的原型

函数原型的目的是告诉编译器这个函数长什么样

名称

参数（数量及类型）

返回类型

旧标准习惯把函数原型写在调用它的函数里面

|  |
| --- |
| #include <stdio.h>  int main()  {  void sum(int begin, int end); // 声明 函数原型  sum(1, 10); // 根据原型判读判断  sum(20, 30);  sum(35, 45);    return 0;  }  void sum(int begin, int end) // 定义 实际的函数头  {  int i;  int sum;  for (i=begin; i<=end; i++) {  sum += i;  }  printf("%d到%d的和是%d\n", begin, end, sum);  } |

现在一般写在调用它的函数前面

函数原型里可以不写参数的名字，但是一般仍然写上（对机器没意义，但对阅读者有意义）

|  |
| --- |
| #include <stdio.h>  void sum(int, int); // 声明 函数原型  int main()  {  sum(1, 10); // 根据原型判读判断  sum(20, 30);  sum(35, 45);    return 0;  }  void sum(int begin, int end) // 定义 实际的函数头  {  int i;  int sum;  for (i=begin; i<=end; i++) {  sum += i;  }  printf("%d到%d的和是%d\n", begin, end, sum);  } |

7.2.2 参数传递：调用函数的时候，是用表达式的值来初始化函数的参数

（1）调用函数

如果函数有参数，调用参数时必须传递给它数量、类型正确的值

可以传递给函数的值是表达式的结果，这包括：

字面量

变量

函数的返回值

计算的结果

（2）类型不匹配？

调用函数时给的值与参数的类型不匹配是C语言传统上最大的漏洞

编译器总是悄悄替你把类型转换好，但是这很可能不是你所期望的

|  |  |
| --- | --- |
| #include <stdio.h>  void cheer(int i)  {  printf("cheer %d\n", i);  }  int main()  {  cheer(2);  return 0;  }  cheer 2 | #include <stdio.h>  void cheer(int i)  {  printf("cheer %d\n", i);  }  int main()  {  cheer(2.4);  return 0;  }  cheer 2 |

后续的语言，C++/Java在这方面很严格

（3）传过去的是什么？

这样的代码能交换a和b的值吗？

|  |
| --- |
| #include <stdio.h>  void swap(int a, int b);  int main()  {  int a = 5;  int b = 5;    swap(a, b);    printf("a=%d b=%d\n", a, b);    return 0;  }  void swap(int a, int b)  {  int t = a;  a = b;  b = t;  }  a=5 b=6 |

C语言在调用函数时，永远只能传值给函数

（4）传值

每个函数有自己的变量空间，参数也位于这个独立的空间中，和其他函数没有关系

过去，对于函数函数参数表中的参数，叫做“形式参数”，调用函数时给的值，叫做实际参数

|  |
| --- |
| #include <stdio.h>  void swap(int a, int b); // 形参  int main()  {  int a = 5;  int b = 5;    swap(a, b); // 实参    printf("a=%d b=%d\n", a, b);    return 0;  }  void swap(int a, int b) // 形参  {  int t = a;  a = b;  b = t;  } |

由于容易让初学者误会实际参数就是实际在函数中进行计算的参数，误会调用函数的时候把变量而不是值传进去了，所以我们不建议继续用这种古来的方式来称呼它们

7.2.3 本地变量：定义在函数内部的变量是本地变量，参数也是本地变量

（1）本地变量

函数的每次运行，就产生一个独立的变量空间，在这个空间中的变量，是函数的这次运行所独有的，称作本地变量

定义在函数内部的变量就是本地变量

参数也是本地变量

（2）变量的生存期和作用域

生存期：什么时候这个变量开始出现了，到什么时候它消亡了

作用域：在（代码的）什么范围内可以可以访问这个变量（这个变量可以起作用）

对于本地变量，这两个问题的答案是统一的：大括号内——块

（3）本地变量的规则

本地变量是定义在块内的：它可以是定义在函数的块内，也可以是定义在语句的块内，甚至可以随便拉一对大括号来定义变量

程序运行进入这个块之前，其中的变量不存在，离开这个块，其中的变量就消失了

块外面定义的变量在里面仍然有效

块里面定义了和外面同名的变量则掩盖了外面的

不能在一个块内定义同名的变量

本地变量不会被默认初始化

参数在进入函数的时候被初始化了

7.2.4 函数庶事：一些有关函数的细节问题，main()是什么

（1）没有参数时

void f(void);√

还是

void f();——在传统C中，它表示f函数的参数表未知，并不表示没有参数

（2）逗号运算符？

调用函数时的逗号和逗号运算符怎么区分？

调用函数时的圆括号里的逗号是标点符号，不是不是运算符

f(a, b)

f((a, b))

（3）函数里的函数？

C语言不允许函数嵌套定义

（4）这是什么？

int i, j, sum(int a, int b);

return (i);

（5）关于main

int main()也是一个函数

要不要写成int main(void)?

return的0有人看吗？

Windows：if error level 1 …

Unix Bash：echo $?

Csh：echo $status

8. 数组

8.1 数组

8.1.1 初试数组

（0）如何写一个程序计算用户输入的数字的平均数？

|  |
| --- |
| #include <stdio.h>  int main()  {  int x;  double sum = 0;  int cnt = 0;  scanf("%d", &x);  while (x != -1) {  sum += x;  cnt ++;  scanf("%d", &x);  }  if (cnt > 0) {  printf("%f\n", sum/cnt);  }    return 0;  } |

如何让记录很多数？——int num1, num2, num3 … ?

（1）数组

int number[100];

scanf("%d", &number[i]);

|  |
| --- |
| #include <stdio.h>  int main()  {  int x;  double sum = 0;  int cnt = 0;  int number[100]; // 定义数组  scanf("%d", &x);  while (x != -1) {  number[cnt] = x; // 对数组中的元素赋值  //  {  int i;  printf("%d\t", cnt);  for (i=0; i<=cnt; i++) {  printf("%d\t", number[i]);  }  printf("\n");  }  sum += x;  cnt ++;  scanf("%d", &x);  }  if (cnt > 0) {  printf("%f\n", sum/cnt);  int i;  for (i=0; i<cnt; i++) {  if (number[i] > sum/cnt) { // 使用数组中的元素  printf("%d\n", number[i]); // 遍历数组  }  }  }    return 0;  }  *1 2 3 4 5 6 7 8 9 10 11 12*  0 1  1 1 2  2 1 2 3  3 1 2 3 4  4 1 2 3 4 5  5 1 2 3 4 5 6  6 1 2 3 4 5 6 7  7 1 2 3 4 5 6 7 8  8 1 2 3 4 5 6 7 8 9  9 1 2 3 4 5 6 7 8 9 10  10 1 2 3 4 5 6 7 8 9 10 11  11 1 2 3 4 5 6 7 8 9 10 11 12  *-1*  6.500000  7  8  9  10  11  12 |

这个程序存在安全隐患，是什么？——这个程序是危险的，因为输入的数据可能超过100个

8.1.2 数组的使用：如何定义和使用数组，数组的下标和下标的范围

（1）定义数组

<类型> 变量名称[元素数量]

int grades[100];

double weight[20];

元素数量必须是整数

C99之前：元素数量必须是编译时刻确定的字面量

（2）数组

是一种容器（放东西的东西），特点是：

其中的所有元素具有相同的数据类型

一旦创建，不能改变大小

数组中的元素在内存中是连续依次排列的

（3）int a[10]

一个int的数组

10个单元：a[0], a[1], a[2], … , a[9]

每个单元就是一个int类型的变量

可以出现在赋值的左边或右边：a[2] = a[1] + 6;（在赋值的左边叫做左值，在右边的叫右值）

（4）数组的单元

数组的每个单元就是数组类型的一个变量

使用数组时放在[]中的数字叫做下标或索引，下标从0开始计数：grades[0], grades[99], average[5]

（5）有效的下标范围

编译器和运行环境都不会检查数组下标是否越界，无论是对数组单元做读还是写

一旦程序运行，越界的数组访问可能造成问题，导致程序崩溃——segmentation fault

但是也可能运气好，没造成严重的后果

所以这是程序员的责任来保证程序只使用有效的下标值：[0, 数组的大小-1]

（6）长度为0的数组？

int a[0];

可以存在，但无用处

8.1.3 数组的例子：统计个数

写一个程序，输入数量不确定的[0, 9]范围内的整数，统计每一种数字出现的次数，输入-1表示结束

|  |
| --- |
| #include <stdio.h>  int main()  {  const int number = 10; // 数组的大小  int x;  int count[number]; // 定义数组  int i;  for (i=0; i< number; i++) { // 初始化数组  count[i] = 0;  }  scanf("%d", &x);  while (x != -1) {  if (x>=0 && x<=9) {  count[x] ++; // 数组参与运算  }  scanf("%d", &x);  }  for (i=0; i< number; i++) { // 遍历数组输出  printf("%d:%d\n", i, count[i]);  }    return 0;  } |

8.2 数组运算

8.2.1 数组运算

在一组给定的数据中，如何找出某个数据是否存在？

|  |
| --- |
| #include <stdio.h>  int search(int key, int a[], int length);  int main(void)  {  int a[] = {2,4,6,7,1,3,5,9,11,13,23,14,32};  int x;  int loc;  printf("请输入一个数字：");  scanf("%d", &x);  loc = search(x, a, sizeof(a)/sizeof(a[0]));  if (loc != -1) {  printf("%d在第%d个位置上\n", x, loc);  } else {  printf("%d不存在\n", x);  }  return 0;  }  int search(int key, int a[], int length)  {  int ret = -1;  int i;  for (i=0; i<length; i++) {  if (a[i] == key) {  ret = i;  break;  }  }  return ret;  } |

（1）数组的集成初始化

int a[] = {2,4,6,7,1,3,5,9,11,13,23,14,32};

直接用大括号给出数组的所有元素的初始值

不需要给出数组的大小，编译器替你数

Tips：int a[13] = {2}; —— {2 0 0 0 0 0 0 0 0 0 0 0 0}

（2）集成初始化时的定位

int a[10] = {[0]=2. [2]=3, 6} // C99 only!

用[n]在初始化数据中给出定位

没有定位的数据接在前面前面额的位置后面

其他位置的值补零

也可以给出数组大小，让编译器算

特别适合初始数据稀疏的数组

（3）数组的大小

seizeof给出整个数组所占据内容的大小，单位是字节：sizeof(a)/sizeof(a[0])

sizeof(a[0])给出数组中单个元素的大小，于是相除就得到了数组的单元个数

这样的代码，一旦修改数组中初始的数据，不需要修改遍历的代码

（4）数组的赋值

int a[] = {2,4,6,7,1,3,5,9,11,13,23,14,32};

int b[] = a; 这样不可以！

（5）遍历数组

通常都是使用for循环，让循环变量i从0到<数组的长度，这样的循环体内最大的i正好是数组最大的有效下标

常见错误是：

循环结束条件是<=数组长度，或离开循环体后继续用i的值来做数组元素的下标

Tips：数组作为函数参数时，往往必须再用另一个参数来传入数组的大小

数组作为函数的参数时：不能在[]中给出数组的大小；不能再利用sizeof来计算数组的元素个数

8.2.2 数组例子：素数

（1）判断素数

|  |
| --- |
| int isPrime(int x);  int main(void)  {  int x;  scanf("%d", &x);  if (isPrime(x)) {  printf("%d是素数\n", x);  } else {  printf("%d不是素数\n", x);  }  return 0;  } |

（2）从2到x-1测试是否可以整除

|  |
| --- |
| int isPrime(int x);  int main(void)  {  int ret = 1;  int i;  if (x == 1) ret = 0;  for (i=2; i<x; i++) {  if (x % i == 0) {  ret = 0;  break;  }  }  return 0;  } |

对于n要循环n-1遍，当n很大时就是n遍

（3）去掉偶数后，从3到x-1，每次加2

|  |
| --- |
| int isPrime(int x);  {  int ret = 1;  int i;  if (x == 1 || (x%2 == 0 && x!=2))  ret = 0;  for (i=3; i<x; i+=2) {  if (x%i == 0) {  ret = 0;  break;  }  }  return ret;  } |

如果x是偶数，立刻

否则要循环(n-3)/2+1遍，当n很大时就是n/2遍

（4）无须到x-1，到sqrt(x)就够了

|  |
| --- |
| int isPrime(int x);  {  int ret = 1;  int i;  if (x == 1 || (x%2 == 0 && x!=2))  ret = 0;  for (i=3; i<sqrt(x); i+=2) {  if (x%i == 0) {  ret = 0;  break;  }  }  return ret;  } |

（5）判断是否能被已知的且<x的素数整除

|  |
| --- |
| int main(void)  {  const int number = 100;  int prime(number) = {2};  int count = 1;  int i = 3;  while (count < number) {  if (isPrime(i, prime, count)) {  prime[count++] = i;  }  i++;  }  for (i=0; i<number; i++) {  printf("%d", prime[i]);  if ((i+1)%5) printf("\t");  else printf("\n");  }  return 0;  }  int isPrime(int x, int knownPrimes[], int numberofKnownPrimes)  {  int ret = 1;  int i;  for (i=0; i<numberofKnownPrimes; i++) {  if (x % knownPrimes[i] == 0) {  ret = 0;  break;  }  }  return ret;  } |

（6）构造素数表

欲构造n以内的素数表

A. 令x为2

B. 将2x、3x、4x直至ax<n的数标记为非素数

C. 令x为下一个没有被标记为非素数的数，重复B；知道所有的数都已经尝试完毕

1. 开辟prime[n]，初始化其所有元素为1，prime[x]为1表示x是素数

2. 令行x=2

3. 如果x是素数，则对于(i=2; x\*i<n; i++)令prime[i\*x]=0

4. 令x++，如果x<n，重复3，否则结束

|  |
| --- |
| #include <stdio.h>  int main()  {  const int maxNumber = 25;  int isPrime[maxNumber];  int i;  int x;  for (i=0; i<maxNumber; i++) {  isPrime[i] = 1;  }  for (x=2; x<maxNumber; x++) {  if (isPrime[x]) {  for (i=2; i\*x<maxNumber; i++) {  isPrime[i\*x] = 0;  }  }  }  for (i=2; i<maxNumber; i++) {  if (isPrime[i]) {  printf("%d\t", i);  }  }  printf("\n")  return 0;  } |

算法不一定和人的思考方式相同

8.2.3 二维数组

（1）二维数组

int a[3][5];

通常理解为a是一个3行5列的矩阵

（2）二维数组的遍历

for (i=0; i<3; i++) {

for (j=0; j<5; j++) {

a[i][j] = i\*j;

}

}

a[i][j]是一个int

表示第i行第j列上的单元

a[i,j]是什么？

（3）二维数组的初始化

int a[][5] = {

{0,1,2,3,4},

{2,3,4,5,6},

};

列数必须给出，行数可以由编译器来数

每行一个{}，逗号分隔

最后的逗号可以存在，有古老的传统

如果省略，表示补零

也可以用定位（C99 only！）

（4）tic-tac-toe游戏

读入一个3\*3的矩阵，矩阵中的数字为1表示该位置上有一个X，为0表示为O

程序判断这个矩阵中是否有获胜的一方，输出表示获胜的一方的字符X或O，或输出无人获胜

（5）读入矩阵

|  |
| --- |
| const int size = 3;  int board[size][size];  int i, j;  int numofX;  int numofO;  int result = -1; // -1:没人赢，1:X赢， 0:O赢  // 读入矩阵  for (i=0; i<size; i++) {  for (j=0; j<size; j++) {  scanf("%d", &board[i][j]);  }  } |

（6）检查行

|  |
| --- |
| // 检查行  for (i=0; i<size && result == -1; i++) {  numofO = numofX = 0;  for (j=0; j<size; j++) {  if (board[i][j] == 1) {  numofX ++;  } else {  numofO ++;  }  }  if (numofO == size) {  result = 0;  } else if (numofX == size) {  result = 1;  }  } |

（7）行和列？

|  |  |
| --- | --- |
| / 检查行  for (i=0; i<size && result == -1; i++) {  numofO = numofX = 0;  for (j=0; j<size; j++) {  if (board[i][j] == 1) {  numofX ++;  } else {  numofO ++;  }  }  if (numofO == size) {  result = 0;  } else if (numofX == size) {  result = 1;  }  } | if (result == -1) {  for (j=0; j<size && result == -1; j++) {  numofO = numofX = 0;  for (i=0; i<size; i++) {  if (board[i][j] == 1) {  numofX ++;  } else {  numofO ++;  }  }  if (numofO == size) {  result = 0;  } else if (numofX == size) {  result = 1;  }  }  } |

能否用一个两重循环来检查行和列？

（8）检查对角线

|  |  |
| --- | --- |
| numofO = numofX = 0;  for (i=0; i<size; i++) {  if (board[i][j] == 1) {  numofX ++;  } else {  numofO ++;  }  }  if (numofO == size) {  result = 0;  } else if (numofX == size) {  result = 1;  } | numofO = numofX = 0;  for (i=0; i<size; i++) {  if (board[i][size-i-1] == 1) {  numofX ++;  } else {  numofO ++;  }  } |

9. 指针

9.1 指针

9.1.1 取地址运算：&运算符取得变量的地址

（1）运算符&

scanf("%d", &i);里的&

获得变量的地址，它的操作数必须是变量：int I; printf(“%x”, &i);

地址的大小是否与int相同取决于编译器：int i; printf(“%p”, &i);

（2）&不能取的地址

&不能对没有地址的东西取地址：&(a+b)? &(a++)? &(++a)?

（3）试试这些&

变量的地址

相邻的变量的地址

&的结果的sizeof

数组的地址

数组单元的地址

相邻的数组单元的地址

9.1.2 指针：指针变量就是记录地址的变量

（1）scanf

如果能够将取得的变量的地址传递给一个变量，能否通过这个地址在那个函数内访问这个变量？

scanf(“%d”, &i);

scanf()的原型应该是怎样的？我们需要一个参数能保存别的变量的地址，如何表达能够保存地址的变量？

（2）指针

就是保存地址的变量

int i;

int\* p = &i;

int\* p,q;

int \*p,q;

（3）指针变量

变量的值是内存的地址：普通变量的值是实际的值；指针变量的值是具有实际值的变量的地址

（4）作为参数的指针

void f(int \*p);

在被调用的时候得到了某个变量的地址：int i=0; f(&i);

在函数里面可以通过访问这个指针访问外面的这个i

（5）访问那个地址上的变量\*

\*是一个单目运算符，用来访问指针的值所表示的地址上的变量

可以做右值也可以做左值：int k = \*p; \*p = k + 1;

（6）\*左值之所以叫左值

是因为出现在赋值号左边的不是变量，而是值，是表达式计算的结果：a[0]=2; \*p=3; 是特殊的值，所以叫做左值

9.1.3 指针的使用：指针有什么用呢？

10. 字符串

10.1 字符串

10.1.1 字符串

（1）字符数组

char word[] = {'H', 'e', 'l', 'l', 'o', '!'}

word[0] -> H

word[1] -> e

word[2] -> l

word[3] -> l

word[4] -> o

word[5] -> !

这不是C语言的字符串，因为不能用字符串的方式做计算

（2）字符串

char word[] = {'H', 'e', 'l', 'l', 'o', '!', '\0'}

word[0] -> H

word[1] -> e

word[2] -> l

word[3] -> l

word[4] -> o

word[5] -> !

word[6] -> \0

以0（整数0）结尾的一串字符：0或'\0'是一样的的，但是和'0'不同

0标志字符串的结束，但它不是字符串的一部分：计算字符串长度的时候不包括这个0

字符串以数组的形式存在，以数组或指针的形式访问：更多的是以指针的形式

string.h里有很多处理字符串的函数

（3）字符串变量

char \*str = 'Hello';

char word[] = 'Hello';

char line[10] = 'Hello';

（4）字符串常量

"Hello"

"Hello"会被编译器变成一个字符数组放在某处，这个数组的长度是6，结尾还有表示结束的0

两个相邻的字符串常量会被自动连接起来

（5）字符串

C语言的字符串是以字符数组的形态存在的：不能用运算符对字符串做运算；通过数组的方式可以遍历字符串

唯一特殊的地方是字符串字面量可以用来初始化字符数组

10.1.2 字符串变量

（1）字符串常量

char\* s = "Hello, world!";

s是一个指针，初始化为指向一个字符串常量：由于这个常量所在的地方，所以实际上s是const char\* s，但是由于历史的元音，编译器接受不带const的写法；但是试图对s所指的字符串做写入会导致严重的后果

如果需要修改字符串，应该用数组：char s[] = "Hello, world!";

（2）指针还是数组？

char\* str = "Hello";

char word[] = "Hello";

数组：这个字符串在这里（作为本地变量空间自动被回收）

指针：这个字符串不知道在哪里（处理参数；动态分配空间）

如果要构造一个字符串——数组

如果要处理一个字符串——指针

（3）char\*是字符串？

字符串可以表达为char\*的形式

char\*不一定是字符串：本意是指向字符的指针，可能指向的是字符的数组（就像int\*一样）；只有它所指的字符数组有结尾的0，才能说它所指的是字符串

10.1.3 字符串输入输出

（1）字符串赋值？

char \*t = 'title';

char \*s;

s = t;

并没有产生新的字符串，只是让指针s指向了t所指的字符串，对s的任何操作就是对t做的

（2）字符串输入输出

char string[8];

scanf("%s", string);

printf("%s", string);

scanf读入一个单词（到空格、tab或回车为止）

scanf是不安全的，因为不知道要读入的内容的长度

（3）安全的输入

char string[8];

scanf("%7s", string);

在%和s之间的数字表示最多允许读入的字符的数量，这个数字比数组的大小小一

下一次scanf从哪里开始？

（4）常见错误

char \*string;

scanf("%s", string);

以为char\*是字符串类型，定义了一个字符串类型的变量string就可以直接使用了

由于没有对string初始化为0，所以不一定每次运行都出错

（5）空字符串

char buffer[100]="";——这是一个空的字符串，buffer[0]== '\0'

char buffer[]="";——这个数组的长度只有1

10.1.4 字符串数组，以及程序参数

（1）字符串数组

char \*\*a——a是一个指针，指向另一个指针，那个指针指向一个字符（串）

char a[][]——

（2）程序参数

int main(int argc, char const\*argv[])

argv[0]是命令本身：当使用Unix的符号链接时，反映符号链接的名字

10.2 字符串函数

10.2.1 单字符输入输出，用putchar和getchar

（1）putchar

int putchar(int c);

向标准输出写一个字符

返回写了几个字符，EOF(-1)表示写失败

（2）getchar

int getchar(void);

从标准输入读入一个字符

返回类型是int是为了返回EOF(-1)：Windows->Ctrl-Z，Unix->Ctrl-D

10.2.2 字符串函数strlen

（1）string.h

#include <string.h>

strlen、strcmp、strcpy、strcat、strstr

（2）strlen

size\_t strlen(const char \*s);——返回字符串长度（不包括结尾的0）

|  |
| --- |
| #include <stdio.h>  #include <string.h>  size\_t mylen(const char\* s)  {  int cnt = 0;  int idx = 0;  while (s[idx != '\0']) {  idx++;  cnt++;  }  return cnt;  }  int main(int argc, char const \*argv[])  {  char line[] = "Hello";  printf("strlen=%lu\n", mylen(line));  printf("sizeof=%lu\n", sizeof(line));    return 0;  } |

10.2.3 字符串函数strcmp

int strcmp(const char \*s1, const char \*2);

比较两个字符串，返回：0:s1==s2；1:s1>s2；-1:s1<s2

|  |
| --- |
| #include <stdio.h>  #include <string.h>  int mycmp(const char\* s1, const char\* s2)  {  while (\*s1 == \*s2 && \*s1 != '\0') {  s1++;  s2++;  }  return \*s1 - \*s2;  }  int main(int argc, char const \*argv[])  {  char s1[] = "abc";  char s2[] = "abc";  printf("%d\n", mycmp(s1, s2));  printf("%d\n", 'a'-'A');    return 0;  } |

10.2.4 字符串函数strcpy

（1）strcpy