一、初识C语言

（一）显示计算结果

1、计算整数的和并显示结果

- 计算整数15和37的**和**，并显示结果

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
int main(void)
{
    printf("%d", 15 + 37);

    return 0;
}
```

```
52
```

2、程序和编译

- 通过**字符**序列创建出的程序称为**源程序**（source program),用来保存源程序的文件称为**源文件**（source file)
  - source是"原始"的意思，因此源程序也叫做原始程序
- 习惯上将C语言源文件的扩展名定为`.c`
- 发生错误时，会显示出相应的**诊断消息**(dignostic message)

3、注释

- 源程序中`/*`和`*/`之间的部分，称为**注释**（comment）

4、固定代码

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
int main(void)
{
    printf("%d", 15 + 37);

    return 0;
```

- stdio是standard I/O (标准输入输出)的缩写，不要与studio混淆

5、printf函数：格式化输出函数

- printf函数可以在显示器上进行输出操作（末尾的f源自format(格式化)这个单词）
- 想要使用某个函数的功能，就必须通过**函数调用**（function call)
  - **函数调用**是申请进行处理的请求，而调用函数时的一些辅助指示则通过**实参**来发出

```
printf("%d", 15 + 37)
```

  - `printf`函数名，显示
  - `"%d"`实参，显示为十进制的形式 decimal（十进制数）
  - `15 + 37`实参，15+37的结果

6、语句

- C语言中需要在末尾加上分号来构成正确的**语句**（statement）

7、计算并显示整数的差

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
int main(void)
{
    printf("%d", 15 - 37);

    return 0;
```

专题 1-1 翻译阶段和编译

- 运行C语言前，理论上要经过8个**翻译阶段**（translation phase）
- 运行源代码需要安装必要的软件环境，也就是**编译器**
- 大多数C语言编译器都是通过**编译方式**把源代码翻译成计算机能够直接理解执行的形式，但也存在逐行解释然后执行的**解释方式**（执行速度比较缓慢）

8、格式化字符串和转换说明

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
int main(void)
{
    printf("15+37的和是%d \n", 15 + 37);

    return 0;
}
```

9、符号的称呼

- P8

10、.无格式化输出

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
int main(void)
{
    printf("你好");

    return 0;
}
```

练习 1-1

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
int main(void)
{
    printf("15减去37的结果是%d。\n",15-37);

    return 0;
}
```

11、.字符串常量

- 像“ABC”这样用双引号括起来的一连串连续排列的文字，称为**字符串常量**(string literal)

12、转义字符

- `\n`换行(new line)
- `\a`响铃（alert）

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
int main(void)
{
    printf("你好 \a\a\a\n");

    return 0;
}
```

- 在显示“你好”后响铃3次

练习 1-2

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
int main(void)
{
    printf("天\n地\n人\n");

    return 0;
}
```

练习 1-3

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
int main(void)
{
    printf("喂！\n\n您好！\n再见。\n");

    return 0;
}
```

（二）变量

1、变量和声明

- 变量是用来放置数字和字符等的”盒子“
- 在用来存放数值的盒子——变量中放入数值后，只要该盒子还在，值就会一直被保存，而且还可以自由地去除或替换数值
- `int n`
    - 通过声明准备出一个名为n的变量（盒子）。这个变量只能用来存放整数值

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
int main(void)
{
    int vx, vy;

    vx = 57;
    vy = vx + 10;

    printf("vx的值是%d。\n",vx);
    printf("vy的值是%d。\n", vy);

    return 0;
}
```

2、赋值

- “=”表示把右侧的值赋给左侧的变量

3、初始化

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
int main(void)
{
    int vx, vy;

    printf("vx的值是%d。\n",vx);
    printf("vy的值是%d。\n", vy);

    return 0;
}
```

- 变量 vx 和 vy 变成了奇怪的值。这是因为在生成变量的时候，变量会被放入一个不确定的值，即**垃圾值**

4、声明时初始化

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
int main(void)
{
    int vx, vy;

    vx = 57;
    vy = vx + 10;

    printf("vx的值是%d。\n",vx);
    printf("vy的值是%d。\n", vy);

    return 0;
}
```

- 变量声明中等号右边的部分，用来指定变量生成时的值，称为**初始值**（initializer）

5、初始化和赋值

- 初始化：在生成变量的时候放入数值
- 赋值：在已生成的变量中放入数值

```c
int abc = 123       /*初始化（在生成变量的时候放入数值）*/

int xyz;        /*用不定制（垃圾值）初始化*/

xyz = 57        /*赋值（在已生成的1变量中放入数值）*/
```

练习 1-4

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
int main(void)
{
    int x=3.14

    printf("x的值是%d。\n",x);

    return 0;
}
```

> 报错

（三）输入和显示

1、通过键盘进行输入

- 读取一个整数值，并显示出来进行确认
- ```c
  #define _CRT_SECURE_NO_WARNINGS 1
  #include <stdio.h>

  int main(void)
  {
      int no;

      printf("请输入一个整数：");
      scanf("%d", &no);

      printf("您输入的是%d。\n", no);

      return 0;
  }

  ```

2、格式化输入函数scanf

- `scanf`函数可以从键盘读取输入的信息
- 通过转换说明“`%d`”来限制函数只能读取十进制数
- 从键盘读取输入的十进制数，并把它保存在`no`中
- `scanf`函数进行读取时，变量名前必须加上一个特殊符号`&`

3、乘法运算

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
```
```

```
{
    int no;

    printf(" 请输入一个整数");
    scanf("%d", &no);

    printf("它的5倍数是%d。\n", 5*no);

    return 0;
}
```

练习 1-5

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int no;

    printf(" 请输入一个整数");
    scanf("%d", &no);

    printf("该整数加上12的结果是%d。\n", no+12);

    return 0;
}
```

练习 1-6

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int no;

    printf(" 请输入一个整数");
    scanf("%d", &no);

    printf("该整数减去6的结果是%d。\n", no-6);

    return 0;
}
```

4、输出函数puts

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
```

```
{
    int n1, n2;

    puts(" 请输入两个整数");
    printf("整数1: "); scanf("%d", &n1);
    printf("整数2: "); scanf("%d", &n2);

    printf("它们的和是%d。\n", n1 + n2);

    return 0;
}
```

- C语言允许在同一行中书写多条语句（同一条语句也可以分成多行书写）
- `puts`函数（末尾的s取自string）
  - `puts`函数可以按照顺序输出作为实参的字符串，并在结尾**换行**。
    - `puts("...")`与`printf("...\n")`的功能基本相同

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int n1, n2;
    int wa;

    puts(" 请输入两个整数");
    printf("整数1: "); scanf("%d", &n1);
    printf("整数2: "); scanf("%d", &n2);

    wa = n1 + n2;

    printf("它们的和是%d。\n", wa);

    return 0;
}
```

练习 1-7

```
    #define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    puts("天");
    puts("地");
    puts("人");

    return 0;
}
```

练习 1-8

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
int main(void)
{
    int n1, n2;

    puts("请输入两个整数");
    printf("整数1："); scanf("%d", &n1);
    printf("整数2："); scanf("%d", &n2);

    printf("它们的乘积是%d。\n", n1* n2);

    return 0;

```

练习 1-9

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
int main(void)
{
    int n1, n2, n3;

    puts("请输入三个整数");
    printf("整数1："); scanf("%d", &n1);
    printf("整数2："); scanf("%d", &n2);
    printf("整数3："); scanf("%d", &n3);

    printf("它们的和是%d。\n", n1+ n2+ n3);

    return 0;

}
```

总结

- 求长方形的面积

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
int main(void)
{
    int width;
    int height;

    puts("求长方形的面积");

    printf("长：");
    scanf("%d", &width);

    printf("宽：");
```

```
    scanf("%d", &height);

    printf("面积是%d。\a\n", width * height);

    return 0;
}
```

二、运算和数据类型

（一）运算

1、运算符和操作符

- 像`+`、`*`这样可以进行运算的符号称为**运算符**(operator)， 作为运算对象的变量或常量称为**操作数**(operand)
- `vx + vy`
  - `+`就是运算符，`vx`和`vy`就是操作数
  - 运算符左侧的操作数称为**第一操作数**或**左操作数**，运算符右侧的操作数称为**第二操作数**或**右操作数**

2、乘除运算符和加减运算符

3、除法运算的商和余数

- 除法运算`/`只取商的整数部分，也就是说会**舍弃小数点以后的部分**
  - 5 / 3 = 1
  - 3 / 5 =0
- 求余`%`
  - 5 % 3 =2
  - 3 % 5=3

4、使用printf函数打印%

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
int main(void)
{
    int vx, vy;

    puts("请输入两个整数：");
    printf("整数vx：");    scanf("%d", &vx);
    printf("整数vy：");    scanf("%d", &vy);

    printf("vx %% vy = %d\n", vx % vy);

    return 0;
}
```

- `%%`

- 这里的格式化字符串中的`%`符号聚友转换说明的功能。因此，当不需要进行转换说明，而只想输出%的时候，就必修写成`%%`
- 当使用不具有转化说明功能的`puts`函数来进行输出的时候，就不能写成`%%`（这样会输出`%%`的）

5、获取整数的最后一位数字

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
int main(void)
{
    int no;

    printf("请输入一个整数：");
    scanf("%d", &no);

    printf("最后一位是%d。\n", no % 10);

    return 0;
}
```

- 通过求余运算符，巧妙地获取整数的最后一位数字

专题 2-1 除法运算的结果

- 两个操作数都是正时
  - 不管是那种编译器，商和余都是正数
- 两个操作数中至少有一个为负时
  - 至于/预算符的结果是“小于代数商的最大整数”还是“大于代数商的最小整数”， 要取决于编译器

6、多个转换说明

- 读取两个整数，并显示它们的商和余数。
  - ```c
    #define _CRT_SECURE_NO_WARNINGS 1
    #include <stdio.h>
    int main(void)
    {
        int a, b;

        puts("请输入两个整数：");
        printf("整数a："); scanf("%d", &a);
        printf("整数b："); scanf("%d", &b);

        printf("a除以b得%d余%d。\n", a / b, a % b);

        return 0;
    }
    ```
  - ```c
    printf("a除以b得%d余%d。\n", a / b, a % b);
    ```

    - 两个转换说明%，表示分别对应从左边数第二个和第三个参数
- 使用scanf函数为变量输入数值时，也可以指定两个以上的转换说明。
    - ```
      scanf("%d%d", &a, &b);
      ```

练习 2-1

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
int main(void)
{
    int x, y;

    puts("请输入两个整数");

    printf("整数x: "); scanf("%d", &x);
    printf("整数y: "); scanf("%d", &y);

    printf("x的值是y的%d%%\n", x / y * 100);

    return 0;
}
```

```
printf("x的值是y的%d%%\n", x / y * 100);
```

- 在整数除法中，`x / y` 可能会导致**精度丢失**。如果 `x` 小于 `y`，整数除法会将结果舍入到 0，从而计算不出正确的百分比。为了解决这个问题，可以将 `x` 转换为浮点数进行计算，或者在乘以 100 之前确保使用浮点数运算。
- 可以使用如下方式修正：
    1. 使用浮点运算来计算百分比，以避免精度丢失。
    2. 调整格式化占位符为 `%f`，用来输出浮点数结果。

修改后的代码如下：

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int x, y;

    puts("请输入两个整数");

    printf("整数x: "); scanf("%d", &x);
    printf("整数y: "); scanf("%d", &y);

    // 使用浮点运算避免整数除法的舍入误差
    printf("x的值是y的%.0f%%\n", (double)x / y * 100);

    return 0;
```
```

```
}
```

其中，`%.0f` 用于保留两位小数。这样可以正确计算并显示 `x` 相对于 `y` 的百分比。

练习 2-2

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int a, b;

    puts("请输入两个整数");

    printf("整数a: "); scanf("%d", &a);
    printf("整数b: "); scanf("%d", &b);

    printf("它们的和是%d，积是%d", a + b, a * b);

    return 0;
}
```

7、单目运算符

- 单目运算符
  - 只需要一个操作数
- 双目运算符
  - 需要两个操作数
- 三目运算符号
  - 需要三个操作数
- `+`操作符、 `-`操作符、 `!`操作符、 `~`操作符统称**单目运算符**

8、赋值运算符

`=`

9、表达式和赋值表达式

- 表达式有变量和常量，以及连接它们的运算符组成
- 使用赋值运算符的表达式称为赋值表达式

10、表达式语句

```c
vc = vx + 32;
```

- 这种由表达式和分号组成的语句称为**表达式语句**

(二)数据类型

1、求平均值

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int a, b;

    puts("请输入两个整数");
    printf("整数a: "); scanf("%d", &a);
    printf("整数b: "); scanf("%d", &b);

    printf("它们的平均值是%d。 \n", (a + b) / 2);

    return 0;
}
```

- 将表达式a + b括起来的（），是优先计算的标记

2、数据类型

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int n;  /* 整数 */
    double x;   /* 浮点数 */;

    n = 9.99;
    x = 9.99;

    printf("int型变量n的值：%d\n", n);
    printf("n/2:%d\n", n / 2);

    printf("double型变量x的值：%f\n", x);
    printf("x/2.0:%f\n", x / 2.0);

    return 0;
}
```

- int
  - 小数点以后的部分会被舍弃
- double
  - 保留小数点以后的部分
- `%f`中的**f**就是浮点数floating-point的首字母，默认显示小数点后6位数字

3、数据类型和对象

- 每种数据类型可存储的值都是有范围的，如：`int`类型的取值范围是-32767~32767
- 数据类型都有一些固定的属性，继承了这些属性而创建出来的实体变量称为**对象**（object）

4、整型常量和浮点数常量

- 像5和37这样的常量，它们都是整数类型的，所以称为**整数常量**(interger constant)
- 像3.14这样包含小数的常量，称为**浮点数常量**(floating constant)

5、double类型的运算

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    double vx, vy;

    puts("请输入两个数。");
    printf("实数vx:");    scanf("%lf", &vx);  // 使用 %lf 来读取 double 类型
    printf("实数vy:");    scanf("%lf", &vy);  // 使用 %lf 来读取 double 类型

    printf("vx + vy = %f\n", vx + vy);
    printf("vx - vy = %f\n", vx - vy);
    printf("vx * vy = %f\n", vx * vy);
    printf("vx / vy = %f\n", vx / vy);

    return 0;
}
```

- `int`类型
  - ```c
    printf("%d", no)
    ```
  - ```c
    scanf("%d", &no)
    ```
- `doube`类型
  - ```c
    printf("%f", no)
    ```
  - ```c
    scanf("%lf", no)
    ```

练习 2-3

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    double x;
```

```
    printf("请输入一个实数：");
    scanf("%lf", &x);

    printf("你输入的是：%f\n", x);

    return 0;
}
```

6、数据类型和运算

- 像 `int/int` 和 `double/double` 这样两个类型相同的操作数之间的运算，所得结果的数据类型和运算对象的数据类型是一致的
    - 像 `int/double` 和 `double/int` 这样一个操作数是 `int` 类型，另一个操作数是 `double` 类型的情况，`int` 类型的操作数会进行**隐式类型转换**，自动向上转型为 `double` 类型，运算演变为 `double` 类型之间的运算。因此，运算的结果也就变成了 `double` 类型

练习 2-4

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    // 定义常量
    const int int_const = 10;          // 整数常量
    const double double_const = 3.14; // 浮点型常量

    // 定义变量
    int int_var;
    double double_var;

    // 输入变量的值
    printf("请输入一个整数：");
    scanf("%d", &int_var);
    printf("请输入一个浮点数：");
    scanf("%lf", &double_var);

    // 进行各种运算
    printf("\n--- 整数常量与 int 型变量运算 ---\n");
    printf("int_const + int_var = %d\n", int_const + int_var);
    printf("int_const - int_var = %d\n", int_const - int_var);
    printf("int_const * int_var = %d\n", int_const * int_var);
    printf("int_const / int_var = %d\n", int_const / int_var);

    printf("\n--- 浮点常量与 double 型变量运算 ---\n");
    printf("double_const + double_var = %f\n", double_const + double_var);
    printf("double_const - double_var = %f\n", double_const - double_var);
    printf("double_const * double_var = %f\n", double_const * double_var);
    printf("double_const / double_var = %f\n", double_const / double_var);

    printf("\n--- int 型变量与 double 型变量混合运算 ---\n");
```

```
    printf("int_var + double_var = %f\n", int_var + double_var);
    printf("int_var - double_var = %f\n", int_var - double_var);
    printf("int_var * double_var = %f\n", int_var * double_var);
    printf("int_var / double_var = %f\n", int_var / double_var);

    return 0;
}

```

```
请输入一个整数：5
请输入一个浮点数：2.5

--- 整数常量与 int 型变量运算 ---
int_const + int_var = 15
int_const - int_var = 5
int_const * int_var = 50
int_const / int_var = 2

--- 浮点常量与 double 型变量运算 ---
double_const + double_var = 5.640000
double_const - double_var = 0.640000
double_const * double_var = 7.850000
double_const / double_var = 1.256000

--- int 型变量与 double 型变量混合运算 ---
int_var + double_var = 7.500000
int_var - double_var = 2.500000
int_var * double_var = 12.500000
int_var / double_var = 2.000000
```

7、类型转换

- (类型名)a　把a的值转换为指定数据类型对应的值
  - `( )`称为**类型转换运算符**

练习 2-5

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int a, b;
    puts("请输入两个整数");

    printf("整数a: "); scanf("%d", &a);
    printf("整数b: "); scanf("%d", &b);

    printf("a是b的%f%%。\n", (double)a / b * 100);
    /* 使用了 (double)a / b 来确保浮点运算 */

    return 0;
```

```
}
```

8、转换说明

- 转换说明，包括`%`和`.`在内，总共由六部分构成，如`%09.9f`
  - 0标志
    - 设定了0标志之后，如果数值的前面有空余位，则用0补齐位数（如果省略了0标志，则会用空白补齐位数）
  - 最小字段宽度
    - 也就是至少要显示出的字符位数。不设定该位数或者显示数值的实际位数超过它的时候，会根据数值显示出必要的位数
  - 精度
    - 指定显示的最小位数，如果不指定，则整数的时候默认为1，浮点数的时候默认为6
  - 转换说明符
    - `d`，显示十进制的`int`型整数
    - `f`，显示十进制的`double`型浮点数
  - 如果设定了“`-`”，数据会左对齐显示，未设定则会右对齐显示

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    printf("[%d]\n", 123);
    printf("[%.4d]\n", 123);
    printf("[%4d]\n", 123);
    printf("[%04d]\n", 123);
    printf("[%-4d]\n\n", 123);

    printf("[%d]\n", 12345);
    printf("[%.3d]\n", 12345);
    printf("[%3d]\n", 12345);
    printf("[%03d]\n", 12345);
    printf("[%-3d]\n\n", 12345);

    printf("[%f]\n", 123.13);
    printf("[%.1f]\n", 123.13);
    printf("[%6.1f]\n\n", 123.13);

    printf("[%f]\n", 123.13);
    printf("[%.1f]\n", 123.13);
    printf("[%4.1f]\n", 123.13);

}
```

```
[123]
[0123]
```

```
[ 123]
[0123]
[123 ]

[12345]
[12345]
[12345]
[12345]
[12345]

[123.130000]
[123.1]
[ 123.1]

[123.130000]
[123.1]
[123.1]
```

练习 2-6

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int x;

    printf("请输入您的身高:");
    scanf("%d", &x);

    printf("您的标准体重是%.1f公斤\n", (x - 100) * 0.9);

    return 0;
}
```

总结

三、分支结构程序

（一）if语句

1、if语句·其1

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int no;

    printf("请输入一个整数：");
```

```
    scanf("%d", &no);

    if (no % 5)
        puts("输入的整数不能被5整除");

    return 0;
}
```

- `if`语句会判断`表达式`的值，如果结果不为0，则执行相应的`语句`
- 括号内对条件进行判断的表达式称为`控制表达式`

2、奇数的判断

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int no;

    printf("请输入一个整数：");
    scanf("%d", &no);

    if (no % 2)
        puts("输入的整数是奇数");

    return 0;
}
```

3、if语句·其2

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int no;

    printf("请输入一个整数：");
    scanf("%d", &no);

    if (no % 5)
        puts("该整数不能被5整除");  /*语句1*/
    else
        puts("该整数能被5整除");   /*语句2*/

    return 0;
}
```

- 当`表达式`的值不为0的时候执行语句1，当`表达式`的值为0的时候执行语句2

4、奇数·偶数的判断

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int no;

    printf("请输入一个整数：");
    scanf("%d", &no);

    if (no % 2)
        puts("该整数是奇数");
    else
        puts("该整数是偶数");

    return 0;
}
```

练习 3-1

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int A, B;

    printf("请输入两个整数。\n");

    printf("整数A: ");    scanf("%d", &A);
    printf("整数B: ");    scanf("%d", &B);

    if (A % B)
        puts("B不是A的约数");
    else
        puts("B是A的约数");

    return 0;
}
```

5、非0的判断

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int num;

    printf("请输入一个整数。\n");
    scanf("%d", &num);
```

```
10
11      if (num)
12          puts("该整数不是0");
13      else
14          puts("该整数是0");
15
16      return 0;
17  }
```

6、if语句的结构图

- `if`
- `else`

7、相等运算符

- `==运算符`
  - a == b　如果a和b的值相等则为1，不等则为0（结果的类型是int）
- `!=运算符`
  - a != b　如果a和b的值不相等则为1，相等则为0（结果的类型是int）

8、余数的判断

```
1   #define _CRT_SECURE_NO_WARNINGS 1
2   #include <stdio.h>
3
4   int main(void)
5   {
6       int num;
7
8       printf("请输入一个整数。\n");
9       scanf("%d", &num);
10
11      if ((num % 10) == 5)
12          puts("该整数的个位数是5");
13      else
14          puts("该整数是个位数不是5");
15
16      return 0;
17  }
```

9、关系运算符

- 比较两个操作数大小关系的运算符称为`关系运算符`,该类运算符共有四种
  - `< 运算符`
  - `> 运算符`
  - `<=运算符`
  - `>=运算符`

10、嵌套的if语句

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int no;

    printf("请输入一个整数。");
    scanf("%d", &no);

    if (no == 0)
        puts("该整数为0.");
    else if(no > 0)
        puts("该整数为正数");
    else
        puts("该整数为负数");

    return 0;
}
```

练习 3-2

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int no;

    printf("请输入一个整数。");
    scanf("%d", &no);

    if (no == 0)
        puts("该整数为0.");
    else if(no > 0)
        puts("该整数为正数");
    else if(no < 0)
        puts("该整数为负数");

    return 0;
}
```

练习 3-3

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int no;

```

```
    printf("请输入一个整数: ");
    scanf("%d", &no);

    // 使用printf输出格式化内容
    if (no >= 0)
        printf("绝对值是%d\n", no);
    else
        printf("绝对值是%d\n", -no);

    return 0;
}
```

练习 3-4

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int A,B;

    printf("整数A: ");
    scanf("%d", &A);
    printf("整数B: ");
    scanf("%d", &B);

    if (A == B)
        puts("A和B相等");
    else if (A > B)
        puts("A大于B");
    else
        puts("A小于B");

    return 0;
}
```

11、判断

- 表达式（极少部分特殊情况除外）都有值。程序执行时会对表达式的值进行检测，这就称为`判断`

12、计算较大值

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int n1, n2, max;

    puts("请输入两个整数。");
    printf("整数1: "); scanf("%d", &n1);
    printf("整数2: "); scanf("%d", &n2);
```

```
11
12      if (n1 > n2) max = n1; else max = n2;
13
14      printf("较大的数是%d.\n", max);
15
16      return 0;
17  }
```

13、计算三个数的最大值

```
1   #define _CRT_SECURE_NO_WARNINGS 1
2   #include <stdio.h>
3
4   int main(void)
5   {
6       int n1, n2, n3, max;
7
8       puts("请输入三个整数。");
9       printf("整数1: "); scanf("%d", &n1);
10      printf("整数2: "); scanf("%d", &n2);
11      printf("整数3: "); scanf("%d", &n3);
12
13      max = n1;
14      if (n2 > max)   max = n2;
15      if (n3 > max)   max = n3;
16
17      printf("最大值是%d.\n", max);
18
19      return 0;
20  }
```

14、条件运算符

```
1   #define _CRT_SECURE_NO_WARNINGS 1
2   #include <stdio.h>
3
4   int main(void)
5   {
6       int n1, n2, max;
7
8       puts("请输入两个整数。");
9       printf("整数1: "); scanf("%d", &n1);
10      printf("整数2: "); scanf("%d", &n2);
11
12      max = (n1 > n2) ? n1 : n2;
13
14      printf("较大的数是%d.\n", max);
15
16      return 0;
17  }
```

- 条件运算符 a ? b : c 如果a不为0，则结果是b的值，否则结果为c的值

- `(condition) ? value_if_true : value_if_false`

15、差值计算

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int n1, n2;

    puts("请输入两个整数。");
    printf("整数1："); scanf("%d", &n1);
    printf("整数2："); scanf("%d", &n2);

    printf("它们的差为%d.\n", (n1 > n2) ? n1 - n2: n2 - n1);

    return 0;
}
```

- 如果 n1 > n2，则为判断表达式n1 - n1所得的值
- 否则为判断表达式n2 - n1所得的值

练习 3-6

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int n1, n2, n3, min;

    puts("请输入三个整数。");
    printf("整数1："); scanf("%d", &n1);
    printf("整数2："); scanf("%d", &n2);
    printf("整数3："); scanf("%d", &n3);

    min = n1;
    if (min > n2)
        min = n2;
    if (min > n3)
        min = n3;

    printf("它们中的最小值为%d。", min);

    return 0;
}
```

练习 3-7

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int n1, n2, n3, n4, max;

    puts("请输入四个整数。");
    printf("整数1："); scanf("%d", &n1);
    printf("整数2："); scanf("%d", &n2);
    printf("整数3："); scanf("%d", &n3);
    printf("整数4："); scanf("%d", &n4);

    max = n1;
    if (max < n2)
        max = n2;
    if (max < n3)
        max = n3;
    if (max < n4)
        max = n4;

    printf("它们中的最大值为%d。", max);

    return 0;
}
```

练习 3-8

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int n1, n2, x;

    puts("请输入两个整数。");
    printf("整数1："); scanf("%d", &n1);
    printf("整数2："); scanf("%d", &n2);


    if (n1 > n2)
        x = n1 - n2;
    if (n1 < n2)
        x = n2 - n1;

    printf("它们的差是%d。", x);

    return 0;
}
```

练习 3-9

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int n1, n2, n3, min;

    puts("请输入三个整数。");
    printf("整数1："); scanf("%d", &n1);
    printf("整数2："); scanf("%d", &n2);
    printf("整数3："); scanf("%d", &n3);

    // 使用三目运算符进行比较
    min = (n1 < n2) ? ((n1 < n3) ? n1 : n3) : ((n2 < n3) ? n2 : n3);

    printf("它们中的最小值为%d。\n", min);

    return 0;
}

```

16、复合语句（程序块）

- 在大括号内并排书写的语句称为**复合语句**或者**程序块**
- `{0个以上的声明0个以上的语句}`

17、逻辑运算符

- 逻辑与运算符　`a&&b`　如果a和b都不为0，则表达式的结果为1，否则结果为0
- 逻辑或运算符　`a||b`　如果a和b中有一个不为0，则表达式的结果为1，否则结果为0

18、短路求值

- 短路求值
  - `&&`运算符在a的判断结果为0时不会对b进行判断
  - `||`运算符在a的判断结果不为0时不会对b进行判断

练习 3-10

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int A, B, C;

    puts("请输入三个整数。");
    printf("整数A："); scanf("%d", &A);
```

```
    printf("整数B: "); scanf("%d", &B);
    printf("整数C: "); scanf("%d", &C);

    // 使用 == 进行比较
    if (A == B && B == C)
        puts("三个值都相等");
    else if (A == B || A == C || B == C)
        puts("有两个值相等");
    else
        puts("三个值各不相同");

    return 0;
}

```

练习 3-11

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int a, b;

    // 输入两个整数
    printf("请输入两个整数。\n");
    printf("整数a: "); scanf("%d", &a);
    printf("整数b: "); scanf("%d", &b);

    // 使用逻辑或运算符判断差值是否小于等于10
    if ((a - b <= 10 && a - b >= 0) || (b - a <= 10 && b - a >= 0))
        printf("它们的差小于等于10。\n");
    else
        printf("它们的差大于等于11。\n");

    return 0;
}
```

（二）switch语句

1、switch语句和break语句

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int no;

    printf(" 请输入一个整数: ");
    scanf("%d", &no);

```

```
    switch (no % 3) {
    case 0 : puts("该数能被3整除。");   break;
    case 1 : puts("该数除以3的余数是1。");    break;
    case 2: puts("该数除以3的余数是2.");    break;
    }

    return 0;
}
```

- 像“case 1 : ”这样用来表示程序跳转的标识称为**标签**
- 1和：之间有没有空格都可以。但是case和1之间必须有空格，不可不加空格写成case1

2、复杂的switch语句

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int sw;

    printf(" 整数: ");
    scanf("%d", &sw);

    switch (sw) {
    case 1 : puts("A");     puts("B");       break;
    case 2 : puts("C");
    case 5 : puts("D");     break;
    case 6 :
    case 7 : puts("E");     break;
    default: puts("F");     break;
    }

    return 0;
}
```

- 在没有`break`语句的时候，程序会“落到”下一条语句上
- 如果改变本程序switch语句中标签的顺序，程序的执行结果也会发生改变，所以在使用switch语句的时候，一定要正确书写标签的额顺序

3、switch语句和if语句

```
if (p == 1)
    c = 3;
else if (p == 2)
    c = 5;
else if (p == 3)
    c = 7;
else if (q == 4)
    c = 9;
```

```
switch (p) {
    case 1 : c = 3; break;
    case 2 : c = 5; break;
    case 3 : c = 7; break;
    default : if (q == 4) c = 9;
}
```

- `switch`语句的格式更加清晰
- 通过**单一表达式**来控制程序流程分支的时候，使用**switch**语句的效果通常要比使用**if**语句的更好

4、选择语句

- `if`语句和`switch`语句都是用来实现程序流程的选择性分支的，因此统称为**选择语句**

练习 3-12

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
#include <stdlib.h> // 包含 abs() 函数的头文件

int main(void)
{
    int no;

    printf("请输入一个整数：");
    scanf("%d", &no);

    switch (abs(no % 2)) // 使用 abs() 确保结果为 0 或 1
    {
    case 0 : puts("该整数是偶数"); break;
    case 1 : puts("该整数是奇数"); break;
    }

    return 0;
}

```

练习 3-13

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int month;

    printf("请输入月份：");
    scanf("%d", &month);

    switch (month)
    {
    case 3 : puts("春季"); break;
    case 4 : puts("春季"); break;
```

```
    case 5 : puts("春季"); break;
    case 6 : puts("夏季"); break;
    case 7 : puts("夏季"); break;
    case 8 : puts("夏季"); break;
    case 9 : puts("秋季"); break;
    case 10 : puts("秋季"); break;
    case 11 : puts("秋季"); break;
    case 12 : puts("冬季"); break;
    case 1 : puts("冬季"); break;
    case 2 : puts("冬季"); break;
    default: puts("该月不存在"); break;
    }

    return 0;
}
```

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int month;

    printf("请输入月份：");
    scanf("%d", &month);

    switch (month)
    {
    case 3:
    case 4:
    case 5:
        puts("春季");
        break;
    case 6:
    case 7:
    case 8:
        puts("夏季");
        break;
    case 9:
    case 10:
    case 11:
        puts("秋季");
        break;
    case 12:
    case 1:
    case 2:
        puts("冬季");
        break;
    default:
        puts("该月不存在");
        break;
    }

    return 0;
```

```
}
```

总结

四、程序的循环控制

（一）do语句

1、do语句

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int retry;

    do{
        int no;

        printf("请输入一个整数：");
        scanf("%d", &no);

        if (no % 2)
            puts("该整数是奇数");
        else
            puts("该整数是偶数");

        printf("要重复一次吗？【Yes...0/No...9】：");
        scanf("%d", &retry);
    } while (retry == 0);

    return 0;
}
```

- ```c
  int no;

        printf("请输入一个整数：");
        scanf("%d", &no);

        if (no % 2)
            puts("该整数是奇数");
        else
            puts("该整数是偶数");
  ```

  这一部分可以按照自己的意愿任意循环执行，这就是`do语句`

- `do语句`循环的对象语句称为`循环体`
  ```

2、复合语句（程序块）中的声明

- 仅在复合语句中使用的变量要在该复合语句中进行声明

3、读取一定范围内的值

- 使用**do语句**，从键盘读取的数值是有限制的

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int hand;

    // 循环输入，直到输入有效的数字
    do
    {
        printf("请选择出什么拳【0…石头 / 1…剪刀 / 2…布】: ");
        scanf("%d", &hand);
    } while (hand < 0 || hand > 2);

    // 输出玩家的选择
    printf("你选择了 ");
    switch (hand)
    {
    case 0: printf("石头。\n"); break;
    case 1: printf("剪刀。\n"); break;
    case 2: printf("布。\n"); break;
    }

    return 0;
}

```

- `do-while` 循环确保用户输入的是 0、1 或 2，如果输入无效的值（如 3、-1），程序会继续提示输入，直到输入有效的值。

4、逻辑非运算符·德摩根定律

- 逻辑非运算符 `!a` 当a的值是0的时候值为1，当a的值不是0的时候值为0

5、德摩根定律

- 对各条件取非，然后将逻辑与变为逻辑或、逻辑或变为逻辑与，然后再取其否定，结果和原条件一样。这称为**德摩根定律**
- `x && y` 和 `!(!x || !y)` 相等
- `x || y` 和 `!(!x && !y)` 相等

6、求多个整数的和及平均值

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int sum = 0;    /*和*/
    int cnt = 0;    /*整数个数*/
    int retry;

        do {
            int t;

            printf("请输入一个整数：");
            scanf("%d", &t);
            sum = sum + t;
            cnt = cnt + 1;

            printf("是否继续？【Yes…0/No…1】：");
            scanf("%d", &retry);
        } while (retry == 0);

        printf("和为%d，平均值为%.2f。\n", sum, (double)sum / cnt);
        return 0;
}
```

练习 4-1

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int no;
    int retry;  // 控制是否继续的变量

    do {
        // 输入一个整数
        printf("请输入一个整数。\n");
        scanf("%d", &no);

        // 判断整数的类型
        if (no == 0)
            puts("该整数为0.");
        else if (no > 0)
            puts("该整数为正数");
        else
            puts("该整数为负数");

        // 询问是否继续
        printf("是否继续？【Yes…0/No…1】：");
        scanf("%d", &retry);

```

```
    } while (retry == 0);  // 当输入 0 时继续循环

    return 0;
}

```

练习 4-2

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
int main(void)
{
    int a, b, low, high, sum;

    printf("请输入两个整数。\n");

    printf("整数a: "); scanf("%d", &a);
    printf("整数b: "); scanf("%d", &b);

    if (a > b)
    {
        low = b;
        high = a;
    }
    else
    {
        low = a;
        high = b;
    }

    sum = 0;
    int t = low;

    do {
        sum = sum + t;
        t = t + 1;
    } while (t <= high);

    printf("大于等于%d小于等于%d的所有整数的和是%d。", low, high, sum);

    return 0;
}
```

7、复合赋值运算符

- 复合赋值运算符 `a@=b` 和a=a@b一样（只是对a的判断仅进行一次）
    - `@=`是这其中的一个:`*= /= %= += -= <<= >>= &= ^= |=`

8、后置递增运算符和后置递减运算符

- 后置递增运算符 `a++`　　使a的值增加1
- 后置递减运算符　　`a--`　　使a的值减少1

（二）while语句

1、while语句

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
int main(void)
{
    int no;

    printf("请输入一个正整数：");
    scanf("%d", &no);

    while (no >= 0) {
        printf("%d", no);
        no--;
    }

    printf("\n");

    return 0;
}
```

练习 4-3

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
int main(void)
{
    int no;

    printf("请输入一个正整数：");
    scanf("%d", &no);

    while (no >= 0) {
        printf("%d", no);
        no--;
    }

    return 0;
}
```

2、用递减运算符简化程序代码

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
int main(void)
{
    int no;

    printf("请输入一个正整数：");
    scanf("%d", &no);

    while (no >= 0)
        printf("%d", no--);

    printf("\n");

    return 0;
}
```

- 调用**printf**函数显示`no--`的值的时候会按照如下步骤执行
  - 显示no的值
  - 然后对no的值进行递减操作

练习 4-4

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int no;

    printf("请输入一个正整数：");
    scanf("%d", &no);

    if (no < 0) {
        printf("输入的数不是正整数。\n");
    }
    else {
        while (no > 0) {
            printf("%d\n", no--);
        }
    }

    return 0;
}
```

3、数据递增

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int i, no;

    printf("请输入一个正整数: ");
    scanf("%d", &no);

    i = 1;
    while (i <= no)
        printf("%d", i++);
    printf("\n");

    return 0;
}
```

练习 4-5

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int i, no;

    printf("请输入一个正整数: ");
    scanf("%d", &no);

    i = 1;
    while (i <= no)
        printf("%d\n", i++);

    printf("\n");

    return 0;
}
```

练习 4-6

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int i, no;

    printf("请输入一个整数: ");
    scanf("%d", &no);

```

```
    i = 2;
    while (i <= no-2)
        printf("%d\n", i+=2);

    return 0;
}
```

练习 4-7

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int i, no;

    printf("请输入一个整数：");
    scanf("%d", &no);

    i = 1;
    while (i <= no/2)
        printf("%d\n", i*=2);

    return 0;
}
```

4、限定次数的循环操作

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int i, no;

    printf("请输入一个整数：");
    scanf("%d", &no);

    while (no-- > 0)
        putchar('*');
    putchar('\n');

    return 0;
}
```

5、字符常量和putchar函数

- 被用单引号括起来的字符称为**字符常量**，字符常量时**int**类型
- 字符常量`‘*’`和字符串常量`‘**’`的区别如下
  - 字符常量`‘*’`表示单一的字符*
  - 字符串常量`‘**’`表示单纯由字符*构成的一连串连续排列的字符

6、do语句和while语句

- do语句……先循环后判断：执行循环体之后进行判断
- while语句……先判断后循环：执行循环体之前进行判断

练习 4-8

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int no;

    printf("请输入一个整数：");
    scanf("%d", &no);

    if (no >= 1) {
        while (no-- > 0)
            putchar('*');
        putchar('\n'); // 只在输入大于等于1时输出换行符
    }

    return 0;
}
```

7、前置递增运算符和前置递减运算符

- 前置递增运算符 `++a`　　使a的值增加1（该表达式的值是递增后的值）
- 前置递减运算符　　`--a`　　使a的值减少1（该表达式的值是递减后的值）

8、do语句的显示

- 行开头有}，do语句的一部分
- 行开头没有}，while语句的一部分

练习 4-9

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
```

```
{
    int no;

    printf("请输入一个整数：");
    scanf("%d", &no);

    if (no > 0) {  // 只在输入的整数大于0时执行
        int i = 0;
        while (i < no) {
            switch (++i % 2)
            {
            case 1: printf("+"); break;
            case 0: printf("-"); break;
            }
        }
    }

    return 0;
}

```

练习 4-10

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int no;

    printf("请输入一个整数：");
    scanf("%d", &no);

    if (no > 0) {
        while (no-- > 0) {
            putchar('*');
            printf("\n");
        }
    }

    return 0;
}
```

9、逆向显示整数值

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int no;

```

```
    do {
        printf("请输入一个整数：");
        scanf("%d", &no);
        if (no <= 0)
            puts("\a请不要输入非正整数。");
    } while (no <= 0);

    printf("该整数逆向显示的结果是：");
    while (no > 0) {
        printf("%d", no % 10);
        no /= 10;
    }
    puts("。");

    return 0;
}
```

练习 4-11

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int no;

    do {
        printf("请输入一个整数：");
        scanf("%d", &no);
        if (no <= 0)
            puts("\a请不要输入非正整数。");
    } while (no <= 0);

    printf("%d逆向显示的结果是：", no);
    while (no > 0) {
        printf("%d", no % 10);
        no /= 10;
    }
    puts("。");

    return 0;
}
```

练习 4-12

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int no, digits = 0;

    // 输入正整数的验证循环
    do {
```

```
        printf("请输入一个正整数：");
        scanf("%d", &no);
        if (no <= 0)
            puts("\a请不要输入非正整数。");
    } while (no <= 0);

    // 计算位数
    int temp = no;  // 保留原始输入值用于显示
    while (temp > 0) {
        temp /= 10;  // 每次将数字除以10
        digits++;    // 计数器递增
    }

    printf("该整数的位数是：%d\n", digits);

    return 0;
}

```

（三）for语句

(一)for语句

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int i, no;

    printf("请输入一个正整数：");
    scanf("%d", &no);

    for (i = 0; i <= no; i++)
        printf("%d", i);
    putchar('\n');

    return 0;
}
```

- for (A; B; C)
    - A预处理,表达式A仅在循环执行之前执行一次.当程序无需预处理的时候,该表达式可以省略
    - B控制表达式,表达式B是用来判定循环操作是否继续执行的表达式,如果该表达式成立(判断结果不为0),则执行循环体
    - C收尾处理,表达式C作为“收尾处理”或“下个循环的准备处理”,会在循环体执行后被判断执行.如果没有需要执行的内容,则该表达式可以省略

(二)使用for语句实现固定次数的循环

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int i, no;

    printf("请输入一个正整数：");
    scanf("%d", &no);

    for (i = 1; i <= no; i++)
        putchar('*');
    putchar('\n');

    return 0;
}
```

练习 4-13

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int i, no, sum = 0;  // 新增sum变量用来存储总和

    printf("请输入一个正整数：");
    scanf("%d", &no);

    for (i = 1; i <= no; i++)
        sum = sum + i;  // 累加每次的i到sum中

    printf("从1到%d的和是：%d\n", no, sum);  // 输出结果

    return 0;
}

```

练习 4-14

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int n;  // 用来存储输入的整数
    int i;  // 循环控制变量

    printf("请输入一个正整数：");
    scanf("%d", &n);

```

```
    for (i = 0; i < n; i++)
        printf("%d", (i % 10) + 1);  // 循环输出1到10的数字

    printf("\n");  // 输出完后换行

    return 0;
}

```

(三)偶数的枚举

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int i, n;

    printf("请输入一个正整数：");
    scanf("%d", &n);

    for (i = 0; i <= n; i += 2)
        printf("%d", i);
    putchar('\n');

    return 0;
}
```

(四)约数的枚举

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int i, n;

    printf("请输入一个正整数：");
    scanf("%d", &n);

    printf("%d 的因数是：", n);
    for (i = 1; i <= n; i++)
    {
        if (n % i == 0)  // 检查 i 是否是 n 的因数
            printf("%d ", i);
    }
    putchar('\n');

    return 0;
}
```

```
21
```

(五)表达式语句和空语句

- 在表达式的末尾加上分号组成的语句称为**表达式语句**
- 只包含一个分号的语句，称为**空语句**

(六)循环语句

- **do**语句，**while**语句，**for**语句，都是循环执行程序流程的语句。这样的语句统称为**循环语句**

练习 4-16

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int i, n;

    printf("请输入一个正整数：");
    scanf("%d", &n);

    for (i = 1; i <= n; i++)
    {
        if (i % 2 == 1)  // 检查 i 是否是 n 的因数
            printf("%d ", i);
    }
    putchar('\n');

    return 0;
}
```

练习 4-17

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int i, n;

    printf("请输入一个正整数：");
    scanf("%d", &n);

    for (i = 1; i <= n; i++)
            printf("%d ", i * i);
    putchar('\n');

    return 0;
}
```

```
17
```

练习 4-18

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int i, n;

    printf("请输入一个整数值：");
    scanf("%d", &n);

    for (i = 1; i <= n; i++) {
        printf("*");
        if (i % 5 == 0)  // 每5个星号后换行
            printf("\n");
    }

    // 如果总数不是5的倍数，最后一行不自动换行，手动换行
    if (n % 5 != 0)
        printf("\n");

    return 0;
}
```

练习 4-19

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int i, n;
    int a = 0;  // 计数器a用于统计约数的个数，初始化为0

    printf("请输入一个正整数：");
    scanf("%d", &n);

    printf("%d 的因数是：", n);
    for (i = 1; i <= n; i++)
    {
        if (n % i == 0) {  // 如果i是n的因数
            printf("%d ", i);  // 打印因数
            a++;  // 每找到一个因数，a加1
        }
    }
    printf("\n约数的个数为：%d\n", a);  // 输出约数的总个数
    return 0;
}

```

（四）多重循环

1、二重循环

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int i, j;

    for (i = 1; i <= 9; i++) {  // i 表示行
        for (j = 1; j <= 9; j++)  // j 表示列
            printf("%3d", i * j);  // 输出 i * j 的值，宽度为3，保证对齐
        putchar('\n');  // 每一行输出完后换行
    }

    return 0;
}
```

> `i <= 9`：最外层循环用于控制行，表示 1 到 9 的乘法表行数。
>
> `j <= 9`：内层循环用于控制列，表示每行的乘法计算和输出。
>
> `printf("%3d", i \* j);`：输出乘积，并保持宽度为 3 格，以确保格式对齐。
>
> `putchar('\n');`：每行结束后换行，保证下一个 `i` 的乘积输出在新行。

2、用break语句强制结束循环

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int i, j;

    for (i = 1; i <= 9; i++) {
        for (j = 1; j <= 9; j++) {
            int seki = i * j;
            if (seki > 40)  // 如果乘积大于40，则跳出当前行的循环
                break;
            printf("%3d", seki);  // 输出乘积，宽度为3，保持对齐
        }
        putchar('\n');  // 打印换行符
    }

    return 0;
}

```

3、显示图形

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int i, j;
    int height, weight;

    printf("高");    scanf("%d", &height);
    printf("宽");    scanf("%d", &weight);

    for (i = 1; i <= height; i++){
        for (j = 1; j <= weight; j++)
            putchar('*');
        putchar('\n');
    }

    return 0;
}
```

4、多重循环

- **for**语句、**do**语句、**while**语句均可通过嵌套结构实现多重循环

专题 4-1 continue语句

- 执行**continue**语句后，循环体的剩余部分就会被跳过

（五）程序的组成元素和格式

1、关键字

- 像**if**和**else**这样的标识符被赋予了特殊的意义。这种具有特殊意义的标识符称为**关键字**，它们是不能用作变量名的

2、运算符

- +、-等

3、标识符

- **标识符**是赋予程序中的变量和函数等的名称
- 必须以**非数字**开头，之后可以是非数字和数字的组合，这里的非数字包括大小写字母和下划线

4、分隔符

- 用来分隔关键字和标识符的符号称为**分隔符**
- `[] {} () * , : = ; # …`

5、常量和字符串常量

- 字符常量、整数常量、浮点数常量和字符串常量都是程序的构成要素

6、自由的书写格式

- C语言原则上允许开发人员以自由的格式编写程序
- 一点注意
  - 构成语句的单位中间不能插入空格类字符
  - 预处理指令中间不能换行
  - 字符串常量和字符常量中间不能换行

7、连接相邻的字符串常量

- 可以把被空格类字符以及注释分隔开的相邻字符串常量作为一个整体来看待

8、缩进

- 方便阅读

总结

五、数组

（一）数组

1、数组

- 通过“号码”把相同数据类型的变量集中起来进行管理，擅长处理数据
- 可以用数组实现相同类型的对象的集合

2、数组的声明（使用数组前的准备）

- 数组的声明通过指定元素类型、变量名、元素个数来进行。另外，[ ]中的元素个数必须是常量
- ```
  1 int a[5];
  ```

3、访问数组（数组的使用方法）

- **下标运算符** a[b]　　从数组a的首个元素算起，访问b个元素后的元素
- 数组声明中使用的[ ]仅仅式一个分隔符，而访问各个元素时使用的[ ]则是运算符
  ```

4、数组的遍历

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int i;
    int v[5];

    for (i = 0; i < 5; i++)
        v[i] = i + 1;

    for (i = 0; i < 5; i++)
        printf("v[%d] = %d\n", i, v[i]);
    return 0;
}
```

- 按顺序逐个查看数组的元素，称为**遍历**
- 一般情况下，元素类型为**Type**的数组，称为**Type数组。**
    - 元素类型为**Type**类型、元素个数为n的数组，写作**Type[n]型**

5、数组初始化

- 在最后一个初始值的后面，也要加上逗号

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

- 可以在声明数组的时候不指定元素个数，数组会根据初始值的个数自动进行设定

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

- 用0对{ }内没有赋初始值的元素进行初始化

```c
int v[5] = {1, 3};  /*用{1, 3, 0, 0, 0}初始化*/
```

- 不能通过赋值语句进行初始化

```c
int v[3];
v = {1, 2, 3};
```

6、数组的复制

- C语言不支持使用基本赋值运算符=为数组赋值

```c
b = a;  /*错误：不能为数组赋值*/
```

- 数组的复制，必须通过使用循环语句等对所有元素逐一赋值来进行

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int i;
    int a[5] = { 17, 23, 36 };  // 剩余元素将自动初始化为0
    int b[5];

    // 使用分号而非逗号来分隔 for 循环的表达式
    for (i = 0; i < 5; i++)
        b[i] = a[i];

    puts(" a   b");
    puts("-------");

    // 使用分号分隔 for 循环的表达式
    for (i = 0; i < 5; i++)
        printf("%4d%4d\n", a[i], b[i]);  // 在每行末尾添加换行符

    return 0;
}

```

7、输入数组元素的值

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int i;
    int x[5];

    for (i = 0; i < 5; i++) {
        printf("x[%d]:", i);
        scanf("%d", &x[i]);
    }

    for (i = 0; i < 5; i++)
        printf("x[%d]=%d\n", i, x[i]);

    return 0;
}
```

- 使用**scanf**函数存储键盘输入值的方法，与其它（数组以外）变量的情况完全一样

8、对数组的元素进行倒叙排序

- 不可以像下面这样进行两个值的交换，这样一来变量a和b的值都会变为b的初始值

> a = b; b = a

- 想要交换a和b的值，必须使用一个额外的变量来存储

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int i;
    int x[7];

    for (i = 0; i < 7; i++) {
        printf("x[%d]:", i);
        scanf("%d", &x[i]);
    }

    for (i = 0; i < 3; i++) {
        int temp = x[i];
        x[i] = x[6 - i];
        x[6 - i] = temp;
    }
    for (i = 0; i < 7; i++)
    printf("x[%d]=%d\n", i, x[i]);
    return 0;
}
```

9、使用数组进行成绩处理

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int i;
    int tensu[5];   /*5名学生的分数*/
    int sum = 0;

    printf("请输入5名学生的分数。\n");
    for (i = 0; i < 5; i++) {
        printf("%2d号：", i + 1);
        scanf("%d", &tensu[i]);
        sum += tensu[i];
    }

    printf("总分：%5d\n", sum);
    printf("平均分：%5.1f\n", (double)sum / 5);

    return 0;
```

```
}

```

- 当学生人数从5人增加道8人时，需要将学生人数由5替换为8，但显示的位数5则不能替换

10、对象式宏

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
#define NUMBER 5

int main(void)
{
    int i;
    int tensu[NUMBER];   /*5名学生的分数*/
    int sum = 0;

    printf("请输入%d名学生的分数。\n", NUMBER);
    for (i = 0; i < NUMBER; i++) {
        printf("%2d号：", i + 1);
        scanf("%d", &tensu[i]);
        sum += tensu[i];
    }

    printf("总分：%5d\n", sum);
    printf("平均分：%5.1f\n", (double)sum / NUMBER);

    return 0;
}

```

- ```c
  #define a b     /*将该指令之后的a替换为b*/
  ```

- a称为**宏名**，为了易于和通常的变量名等进行区分，宏名一般用大写字母来表示
- 在程序中使用宏，不仅能够在一个地方统一管理，而且通过为常量定义名称，还可以使程序阅读起来更容易。如果能够加上恰当的注释，效果会更加明显
- 程序中的5等常量，称为幻术（不清楚具体表示什么的数值）。引入对象式宏后，就可以消除程序中的幻数了

11、数组元素的最大值和最小值

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
#define NUMBER 5

int main(void)
{
    int i;
    int tensu[NUMBER];   /* 5名学生的分数 */
    int max, min;

    printf("请输入%d名学生的分数。\n", NUMBER);
```
```

```
    for (i = 0; i < NUMBER; i++) {
        printf("%2d号: ", i + 1);
        scanf("%d", &tensu[i]);
    }

    // 初始化最大值和最小值
    min = max = tensu[0]; // 使用 tensu[0] 进行初始化

    for (i = 1; i < NUMBER; i++) {
        if (tensu[i] > max)  max = tensu[i];
        if (tensu[i] < min)  min = tensu[i];
    }

    printf("MAX: %d\n", max); // 输出格式调整
    printf("MIN: %d\n", min); // 输出格式调整

    return 0;
}
```

12、赋值表达式的判断

- 赋值表达式的判定结果，和赋值后左操作数的类型和值相同

13、数组的元素个数

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
#define NUMBER 80

int main(void)
{
    int i, j;
    int num;
    int tensu[NUMBER];   /* 学生的分数 */
    int bunpu[11] = { 0 }; // 初始化分布数组，大小为11，用于统计0-100的分数

    printf("请输入学生人数: ");

    do
    {
        scanf("%d", &num);
        if (num < 1 || num > NUMBER)
            printf("\a请输入1至%d的数: ", NUMBER);
    } while (num < 1 || num > NUMBER);

    printf("请输入%d人的分数。\n", num);

    for (i = 0; i < num; i++) {
        printf("%2d号: ", i + 1);
        do {
            scanf("%d", &tensu[i]); // 需要使用 &tensu[i] 获取地址
```

```
            if (tensu[i] < 0 || tensu[i] > 100)
                printf("\a请输入0至100的数：");
        } while (tensu[i] < 0 || tensu[i] > 100);
        bunpu[tensu[i] / 10]++; // 统计分数
    }

    puts("\n---分布图---");
    printf("100:");

    for (j = 0; j < bunpu[10]; j++)
        putchar('*');
    putchar('\n');

    for (i = 9; i >= 0; i--) { // 从9到0
        printf("%3d - %3d:", i * 10, i * 10 + 9);
        for (j = 0; j < bunpu[i]; j++)
            putchar('*');
        putchar('\n');
    }

    return 0;
}

```

（二）多维数组

1、多维数组

- 多维数组是以数组为元素的数组
- ```
  int[4][3];
  ```
  - 以”元素类型为int型、元素个数为3的数组“为元素、元素个数为4的数组
- 二维数组就像是一个由”行“和”列“构成的表单，其中各元素纵横排列
- 在多维数组的声明中，最先集中起来的元素个数（二位数组的列数）放在末尾
- 构成数组的最小单位的元素，称为**构成元素**
- 多维数组的构成元素排列时，首先从末尾的下标开始递增

总结

六、函数

一、什么是函数

1、main函数和库函数

2、什么是函数
  ```

3、函数定义

4、函数调用

5、三个数中的最大值

6、将函数的返回值作为参数传递给函数

7、调用其他函数

8、值传递

二、函数设计

1、没有返回值的函数

2、通用性

3、不含形参的函数

4、函数返回值的初始化

5、作用域

6、文件作用域

7、声明和定义

8、函数原型声明

9、头文件和文件包含指令

10、函数的通用性

11、数组的传递

12、函数的传递和const类型的修饰符

13、线性查找（顺序查找）

14、哨兵查找法

15、多维数组的传递

三、作用域和存储期

1、作用域和标识符的可见性

2、存储期

七、基本数据类型

（一）基本数据类型和数

1、算数类型和基本数据类型

- 整数型数据类型：只表示整数
  - 枚举型
  - 字符型
  - 整型
- 浮点型：可表示具有小数部分的数值
  - float型
  - double型
  - long double型
- 字符型、整型和浮点型只需使用**int**或**double**等关键性就能表示其数据类型，因此将它们统称为**基本数据类型**

2、基数

- 二进制
- 八进制
- 十进制
- 十六进制

3、基数转换

- 由八进制数、十六进制数、二进制数向十进制数转换
  - 十进制:$1998 = 1 * 10^3 + 9 * 10^2 + 9 * 10^1 + 8 * 10^0$
  - 八进制:$123 = 1 * 8^2 + 2 * 8^1 + 3 * 8^0$
  - 十六进制：$1FD = 1 * 16^2 + 15 * 16^1 + 13 * 16^0$
  - 二进制：$101 = 1 * 2^2 + 0 * 2^1 + 1 * 2^0$
- 由十进制数向八进制数、十六进制数、二进制数转换
  - 二进制数中的偶数的末位数字为0，奇数的末位数字为1
  - 辗转相除法，%求得末尾数字，/将数字右移一位

（二）整型和字符型

1、整型和字符型

- C语言最擅长处理的是整型和字符串
- **整型**和**字符型**是用来表示限定范围内连续整数的数据类型
- **无符号整型（signed）**：表示0和正数的整型
- **有符号整型（unsigned）**：表示0和正负数的整型

- 4种类型的整型
  - **char　short int　int long int**
- 对于单独的**short**和**long**，可以认为是省略了**int**。
- 对于单独的**signed**和**unsigned**，可以认为是**int**

signed char	char	unsigned char
signed short int		unsigned short int
signed int		unsigned int
sigened long int		unsigned long int

- **char**型比较特殊，存在既不带**signed**又不带**unsigned**的“单独”的**char**型

2、<limits.h>头文件

- C语言编译器在<limits.h>头文件中以宏定义的形式定义了字符型以及其它整型所能表示的数值的最小值和最大值

3、字符型

char型式用来保存“字符”的数据类型

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
#include <limits.h>

int main(void)
{
    printf("这个编译器中的char型是：");

    if (CHAR_MIN)
        puts("有符号的。");
    else
        puts("无符号的。");

    return 0;
}
```

4、位和CHAR_BIT

- C语言中“位”(bit)的定义如下所示
  - “位”是具有大量内存空间的运行环境的数据存储单元，可保存具有两种取值的对象。对象中各二进制位的地址不需要表示
  - “位”可取两种值，其中一种是0。将位设为0以外的值，称为”设置位”
- 根据编译器的不同，**char**型在内存上占据的位数也不同。该位数作为对象式宏**CHAR_BIT**定义在<limits.h>中

```
#define CHAR_BIT 8
```

- 字符型和整型能够表示的数值范围之所以因编译器而异，是因为在内存上占据的位数因编译器而异

5、sizeof运算符

- **sizeof运算符　sizeof a　求a（对象、常量、类型名等)的长度**
- **sizeof(char)**必定为1
- **sizeof(short)** $\leq$ **sizeof(int)** $\leq$ **sizeof(long)**

6、size_t型和typedef声明

- 由**sizeof**运算符生成的值的数据类型是在<stdded.h>头文件中定义的**size_t**型
- 在许多编译器中用**typedef声明**来定义**size_f**

```
tpyedef A   B;
```

- typedef声明为已有的数据类型A创建别名B，该名称一般称为typedef名

7、整型的灵活运用

- sizeof(类别名)　了解数据类型的长度
- sizeof 表达式　了解变量或表达式的长度

```
sizeof(a) / sizeof(a[0])    /*求数组a的元素个数的表达式*/
```

8、整型的内部表示

- 整型内部的位表示使用的是**纯二进制计数法**
- 对于构成整型的位序列列的解释，无符号类型和有符号类型是完全不同的

9、无符号整数的内部表示

- 无符号整数的数值在计算机内部是以二进制数来表示的，该二进制数与各二进制位一一对应
- 在多数编译器中，整型占有的内存的位数都是8，16，32，64……这样的8的倍数

位数	最小值	最大值
8	0	255

位数	最小值	最大值
16	0	65535
32	0	4294967295
64	0	18446744073709551615

- 一般来说，n位可以表示的无符号整数由0~2^n-1共2^n种

10、有符号整数的内部表示

- 有符号整数的内部表示因编译器而不同。最常用的内部表示法由**补码、反码、符号和绝对值**3种
- 共同之处：用最高位表示数值的符号
  - 如果该数为负，则符号位为1；如果该数不为负，则符号位为0
- 补码
  - $-B_{n-1}*2^{n-1}+B_{n-2}*2^{n-2}+\ldots+B_1*2^1+B_0*2^0$
  - 如果位数为n,则能表示-2^{n-1}到$2^{n-1}-1$之间的值
- 反码
  - $-B_{n-1}*(2^{n-1}-1)+B_{n-2}*2^{n-2}+\ldots+B_1*2^1+B_0*2^0$
  - 如果位数为n,则能表示$-2^{n-1}+1$到$2^{n-1}-1$之间的值，只比补码少一个
- 符号和绝对值
  - $(1-2*B_{n-1})*(B_{n-2}*2^{n-2}+\ldots+B_1*2^1+B_0*2^0$
  - 能够表示的值的范围和反码一样
- 无论是3种表示方法中的哪一种，有符号整型和无符号整型的共通部分，即非负数部分的位串都是一样的

11、按位操作的逻辑运算

- 逻辑与 两者都为1时结果为1 a & b
- 逻辑或 只要有一个为1结果就为1 a | b
- 逻辑异或 有且只有一个为1结果才为1 a ^ b
- 反码 如果是0结果则结果为1，如果是1则结果为0 ~a

专题 7-3 逻辑运算符和按位逻辑运算符

- &、|、~运算符会根据1为真、0为假的规则对操作数的各二进制位进行逻辑运算
- &&、||、！运算符会根据非0为真、0为假的规则对操作数的值进行逻辑运算

12、位移运算符

- << a << b 将a左移b位。右面空出的位用0填充 >> a >> b 将a右移b位
- 左移n位，乘2^n,右移n位，除2^n所得的商的整数部分
- 当x是有符号整型的负数时，位移运算的结果因编译器而异。在许多编译器中，会执行**逻辑唯一或算术位移**

专题 7-4 逻辑位移和算术位移

- 逻辑位移
  - 不考虑符号位，所有二进制位都进行位移
- 算术位移
  - 保留最高位的符号位，只有其它位进行位移，用位移前的符号位来填补空位，位移前后符号不变

13、整型常量

- 十进制常量
- 八进制常量，以0开头
- 十六进制常量，以0x或者0X开头

14、整型常量的数据类型

- 整型后缀
  - **u**和**U**……表示该整型常量为无符号类型
  - **l**和**L**……表示该整型常量为long型
- 整型常量的数据类型由三个因素决定
  - 该整型常量的值
  - 该整型常量的后缀
  - 所在编译器中各数据类型的表示范围

15、整数的显示

- printf( )
  - %d十进制
  - %o八进制
  - %x或%xx十六进制

16、数据溢出和异常

- 因**数据溢出**（溢位）使运算结果超出可表示的数值范围或违反数学定义（除以0等）时会发生**异常**
  - 发生异常时程序如何运行由编译器决定
- 无符号整型的运算中不会发生数据溢出。当运算结果超出最大值时，结果为“数学运算结果%”

（三）浮点型

1、浮点型

- **浮点型**用来表示带有小数部分的实数
  - float　　double long double
- 浮点型的“表示范围”由**长度**和**精度**共同决定
  - 长度为12位数字，精度为6位有效数字
  - 1234567890

- 精度在6位时无法正确表示，所以将第7位四舍五入，得到12345670000
- $1.23457 * 10^9$
- 1.23457称为**尾数**，9称为**指数**。尾数的位数相当于“精度”，指数的值相当于”长度“

2、浮点型常量

- 浮点型常量末尾可以加上指定类型的**浮点型后缀**
  - 后缀**f**或**F**表示**float**型，后缀**l**或**L**表示**long double**型

3、<math.h>头文件

- <math.h>头文件包含数学函数的声明

4、循环的控制

- 循环判断基准所使用的变量应为整数而不要用浮点数
- 计算机不能保证其内部转换为二进制的浮点数的每一位都不发生数据丢失，会将误差积累

（四）运算和运算符

1、运算符的优先级和结核性

2、优先级

- 与生活中使用的数学规则一样

3、结核性

4、数据类型转换

总结

八、动手编写各种程序吧

（一）函数式宏

1、函数和数据类型

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int sqr_int(int x)
{
    return x * x;
}

double sqr_double(double x)
{
    return x * x;
}

```

```c
int main(void)
{
    int n;
    double x;

    printf("请输入一个整数：");
    scanf("%d", &n);  // 读取整数
    printf("该数的平方是 %d。\n", sqr_int(n));

    printf("请输入一个实数：");
    scanf("%lf", &x);  // 使用 %lf 读取实数
    printf("该数的平方是 %.2f。\n", sqr_double(x));  // 使用 %.2f 输出实数

    return 0;
}

```

2、函数式宏

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
#define sqr(x) ((x) * (x))

int main(void)
{
    int n;
    double x;

    printf("请输入一个整数：");
    scanf("%d", &n);
    printf("该数的平方是 %d。\n", sqr(n));

    printf("请输入一个实数：");
    scanf("%lf", &x);
    printf("该数的平方是 %.2f。\n", sqr(x));

    return 0;
}
```

- 当出现sqr(O)形式的表达式，将其展开为((O) * (O))

3、函数和函数式宏

- 函数式宏sqr是在编译时展开并填入程序的
- 函数进行复杂的处理
  - 参数传递（将实参的值复制到形参）
  - 函数调用和函数返回值操作（程序流程的控制）
  - 返回值的传递
- 函数宏所做的工作只是宏展开和填入程序，并不进行上述处理
- 函数式宏或许能使程序的运行速度稍微提高一点，但是程序自身却有可能变得臃肿

- 在定义和使用函数式宏时，仔细考虑是否会产生副作用

4、不带参数的函数式宏

- 函数式宏也可以向函数那样进行不带参数的定义

```
#define alert( ) putchar('\a')
```

5、函数式宏和逗号运算符

- 逗号运算符 a, b 按顺序判断a和b，整个表达式最终生成b的判断结果
- 如果宏定义中要代换两个以上的表达式，则使用逗号运算符连接，使其在语法上构成一个表达式

（二）排序

1、冒泡排序法

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
#define NUMBER 5

void bsort(int a[], int n)
{
    int i, j;

    for (i = 0; i < n - 1; i++) {
        for (j = n - 1; j > 1; j--) {
            if (a[j - 1] > a[j]) {
                int temp = a[j];
                a[j] = a[j - 1];
                a[j - 1] = temp;
            }
        }
    }
}
int main(void)
{
    int i;
    int height[NUMBER];

    printf("请输入%d人的身高。\n", NUMBER);
    for (i = 0; i < NUMBER; i++) {
        printf("%2d号：", i + 1);
        scanf("%d", &height[i]);
    }
bsort(height, NUMBER);

puts("按升序排列。");
for (i = 0; i < NUMBER; i++)
    printf("%2d号：%d\n", i + 1, height[i]);
return 0;
}
```

```
36
37
38
```

- 从倒数的两个数字开始进行比较，比较完为一趟
- 有n个元素的情况下，只需重复进行n-1趟，就可以完成排序

（三）枚举类型

1、枚举类型

```
1  enum animal{Dog, Cat, Mongkey, Invalid}; /*enum：枚举*/
```

- 这是枚举类型的声明，它表示了所有可用值的集合
- animal被称为**枚举名**
- 写在{ }中的Dog、Cat、Monkey、Invalid是**枚举常量**

2、枚举常量

- 在枚举常量的名称后面写上赋值运算符“=”和值就能为枚举常量赋值
- 通过赋值运算符”=“赋值的枚举常量，其值即为给定值。没有给定值的枚举常量，其值为前一个枚举常量加1
- 多个枚举常量允许具有相同的值
- 程序的枚举名也是可以省略的
- 能用枚举类型表示的数据类型是，应尽量用枚举类型来表示

3、命名空间

- 枚举名和变量名分别属于不同的**命名空间**，因此即便名称相同也能正确区分

（四）递归函数

1、函数和类型

- **递归**：将自己包含在内，或者用自己来定义自己

2、阶乘

```
1  #define _CRT_SECURE_NO_WARNINGS 1
2  #include <stdio.h>
3
4  int factorial(int n)
5  {
6      if (n > 0)
7          return n * factorial(n - 1);
8      else
9          return 1;
10 }
11
12 int main(void)
```

```
{
    int num;

    printf("请输入一个整数：");
    scanf("%d", &num);

    printf("该数的阶乘为%d。\n", factorial(num));

    return 0;
}
```

- 为了求得n-1的阶乘，函数factorial调用函数factorial。像这样的函数调用称为**递归函数调用**
  - 递归函数调用与其说是"调用该函数本身"，理解为"调用和该函数同样的函数"更加自然，如果真的是调用函数本身，则会一直调用下去，进入死循环
- 在使用树结构、图表、分治法的程序等中，递归算法被广泛应用

（五）输入输出和字符

1、getchar函数和EOF

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int ch;

    while ((ch = getchar()) != EOF)
        putchar(ch);

    return 0;
}
```

- getchar函数的功能时读取字符并将其返回。输入结束或读取过程中发生错误时，就会返回EOF值
- 对象式宏EOF的名称来源于End OF File。在<stdio.h>头文件中，EOF被定义为负值

2、从输入复制到输出

- 上面的程序实际上仅由while语句构成
- 将读取的字符赋值给ch

3、数字字符计数

专题 8-4 缓冲和重定向

- c语言的输入输出一般会将读入的字符以及待输出的字符暂时保存在缓存中，当达到下列条件时才进行实际的输入输出操作
  - 全缓冲：缓存已满
  - 行缓冲：输入换行符
  - 无缓冲：立即输出

- 重定向：输入输出的地址，在程序启动时能够发生改变

4、字符

- C语言中的字符都作为非负整数值来处理。因此，每一个字符都有与之对应的编码（即整数值）
- C语言中的“字符”就是该字符的字符编码（即整数值）

5、转义字符

总结

九、字符串的基本知识

（一）什么是字符串

1、字符串字面量

- 像“ABC”那样带双引号的一系列字符称为**字符串字面量**
- 在字符串字面量的末尾会被加上一个叫做**null字符**的值为0的字符
- 双引号中没有任何字符的字符串字面量“ ”表示的就是**null**字符

2、字符串字面量的长度

- 字符串字面量的长度，和包括末尾的**null**字符在内的字符数一致
- 具有静态声明周期
- 对于同一字符串字面量的处理方式依赖于编译器

3、字符串

- 字符串最适合放在**char**数组中存储。字符串的末尾是首次出现的**null**字符

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    char str[4];

    str[0] = 'A';
    str[1] = 'B';
    str[2] = 'C';
    str[3] = '\0';

    printf("字符串str为\"%s\"。\n", str);

    return 0;
}
```

4、字符数组的初始化赋值

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    char str[] = "ABC";

    printf("字符串str为\"%s\"。\n", str);

    return 0;
}
```

- 除了初始化赋值的时候，不能将数组的初始值或字符串直接赋予数组变量

5、空字符串

- 一个字符也没有的字符串，称为**空字符串**

6、字符串的读取

```c
scnaf("%s, name")
```

scanf函数不能加上&

7、格式化显示字符串

```c
%9.9s
```

- 上面%后的从左到右分别是输出最小宽度、精度、转换说明符号
    - 输出最小宽度：表示至少要输出指定的位数。如果设置了-标志，则表示左对齐，否则表示右对齐
    - 精度：指定显示位数的上限
    - 转换说明符:s表示输出字符串

（二）字符串数组

1、字符串数组

- 类型相同的数据集合适合用数组来实现。字符串的集合是字符串数组

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int i;
    char cs[][6] = { "Turbo", "NA", "DOHC" };

    for (i = 0; i < 3; i++)
        printf("cs[%d] = \"%s\"\n", i, cs[i]);

    return 0;
}
```

2、读取字符串数组中的字符串

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int i;
    char s[3][128];

    for (i = 0; i < 3; i++) {
        printf("s[%d] ; ", i);
        scanf("%s", s[i]);
    }
    for (i = 0; i < 3; i++)
        printf("s[%d] = \"%s\"\n", i, s[i]);

    return 0;
}
```

（三）字符串处理

1、字符串长度

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int str_length(const char s[])
{
    int len = 0;

    while (s[len])
        len++;
    return len;
}
```

```c

int main(void)
{
    char str[128];

    printf("请输入字符串：");
    scanf("%s", str);

    printf("字符串\"%s\"的长度是%d。\n", str, str_length(str));

    return 0;
}
```

- 注：const声明不改变所接收的数组的元素的值

2、显示字符串

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

void put_string(const char s[])
{
    int i = 0;
    while (s[i])
        putchar(s[i++]);
}

int main(void)
{
    char str[128];

    printf("请输入字符串：");
    scanf("%s", str);

    printf("你输入了");
    put_string(str);
    printf("\n");

    return 0;
}
```

对每个字符进行遍历，直到出现**null**字符为止

3、数字字符的出现次数

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

void str_dcount(const char s[], int cnt[])
{
    int i = 0;
    while (s[i]) {
        if (s[i] >= '0' && s[i] <= '9')
            cnt[s[i] - '0']++;
```

```
            i++;
    }
}

int main(void)
{
    int i;
    int dcnt[10] = { 0 };
    char str[128];

    printf("请输入字符串：");
    scanf("%s", str);

    str_dcount(str, dcnt);

    puts("数字字符的出现次数");
    for (i = 0; i < 10; i++)
        printf("'%d':%d\n", i, dcnt[i]);

    return 0;
}
```

4、大小写字符转换

- ```
  int toupper(int c);
  ```

  将小写英文字母转换为相应的大写英文字母

- ```
  int tolower(int c);
  ```

  将大写英文字母转换为相应的小写英文字母

5、字符串数组的参数传递

总结

十、指针

（一）指针

1、函数的参数

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

void sum_diff(int n1, int n2, int sum, int diff)
{
    sum = n1 + n2;
    diff = (n1 > n2) ? n1 - n2 : n2 - n1;
}

```

```c
int main(void)
{
    int na, nb;
    int wa = 0, sa = 0;

    puts("请输入两个整数：");
    printf("整数A:"); scanf("%d", &na);
    printf("整数B:"); scanf("%d", &sa);

    sum_diff(na, nb, wa, sa);

    printf("两数之和为%d，之差为%d。\n", wa, sa);

    return 0;
}
```

```
请输入两个整数：
整数A:31
整数B:21
两数之和为0，之差为0。
```

- 值传递：实参传给形参

2、对象和地址

- 变量是“保存数值的盒子”，有序地排列在内存空间里
- “变量“具有多个侧面或者说多个属性，如数据类型长度、数据类型、存储器、变量名
- **对象**也具具备多个性质和属性
- 在广阔的内存空间上，存在着很多对象，这就需要用某种方式来表示各个对象在内存中的”位置“，这就是**地址**
- 对象的地址是指对象在内存中的存储位置编号

3、取址运算符

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    int n;
    double x;
    int a[3];

    printf("n的地址：%p\n", &n);
    printf("x的地址：%p\n", &x);
    printf("a[0]的地址：%p\n", &a[0]);
    printf("a[1]的地址：%p\n", &a[1]);
    printf("a[2]的地址：%p\n", &a[2]);

    return 0;
}
```

```
n的地址：0000000FA377F854
x的地址：0000000FA377F878
a[0]的地址：0000000FA377F898
a[1]的地址：0000000FA377F89C
a[2]的地址：0000000FA377F8A0
```

- 对象的地址通常是用十六进制数表示的。但不同的编译器或不同的运行环境下，基数、位数等显示形式以及具体数值都会有所不同
- 取址运算符**&**的功能是取得对象的地址
- 表示对象地址的转换说明是**%p**，pointer的首字母

4、指针

- **int**型变量：保存"整数"的盒子
- 指向**int**型变量的指针变量：保存"存放整数对象的地址"的盒子
- 当指针p的值为对象x的地址时，一般说"p指向x"。
- 将取址运算符**&**写在**Type**型对象x前得到的**&**x为**Type *** 型指针，其值为x的地址

5、指针运算符

- *a　a指向的对象
- 当p指向x时，*p就是x的别名
- 指针运算符，也称间接访问运算符

（二）指针和函数

1、作为函数参数的指针

- 通过在指针前写上指针运算符*来访问该指针指向的对象，称为**解引用**

2、计算和差

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

void sum_diff(int n1, int n2, int *sum, int *diff)
{
    *sum = n1 + n2;
    *diff = (n1 > n2) ? n1 - n2 : n2 - n1;
}

int main(void)
{
    int na, nb;
    int wa = 0, sa = 0;

    puts("请输入两个整数：");
    printf("整数A:"); scanf("%d", &na);
    printf("整数B:"); scanf("%d", &nb);

    sum_diff(na, nb, &wa, &sa);  // 传递地址
```

```
    printf("两数之和为%d, 之差为%d。\n", wa, sa);

    return 0;
}

```

- 将指向对象的指针作为形参，并在指针前写上指针运算符*，就可以访问该对象本身。充分利用这一点，就可以在被调用处修改进行调用处的对象的值

3、二值互换

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

void swap(int* px, int* py)
{
    int temp = *px;
    *px = *py;
    *py = temp;
}

int main(void)
{
    int na, nb;

    puts("请输入两个整数。");
    printf("整数A: "); scanf("%d", &na);
    printf("整数B: "); scanf("%d", &nb);

    swap(&na, &nb);

    puts("互换了两数的值。");
    printf("整数A是%d。\n", na);
    printf("整数B是%d。\n", nb);

    return 0;
}
```

4、将两个值排序

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

void swap(int* px, int* py)
{
    int temp = *px;
    *px = *py;
    *py = temp;
}

void sort2(int* n1, int* n2)
```

```c
{
    if (*n1 > *n2)
        swap(n1, n2);/*不需要&*/
}

int main(void)
{
    int na, nb;

    puts("请输入两个整数：");
    printf("整数A：");      scanf("%d", &na);
    printf("整数B：");      scanf("%d", &nb);

    sort2(&na, &nb);

    puts("将两数的值按升序排列");
    printf("整数A是%d。\n", na);
    printf("整数B是%d。\n", nb);

    return 0;
}
```

- n1和n2这两个指针指向的是保存待排列数值的变量，因此将它们直接传入swap函数

5、scanf函数和指针

- **scanf**函数接收的是指针（具有地址的"值"），由该指针所指对象保存从标准输入（一般为键盘）读到的值
- 调用**scanf**函数的一方必须发出请求：请放入该地址中存储的对象所读取的值

6、指针的类型

7、空指针

- **空指针**是能够和指向对象的指针明确区分的"什么也不指向"的特殊指针
- 表示空指针的对象式宏，是称为**空指针常量**的**NULL**

8、标量型

- 可以将表示地址编号的指针视为一种数量
- 第7章中介绍的数值型和本章中介绍的指针型统称为**标量型**(scalar type)
- scalar有"数""数量"的意思。标量虽然有大小，但是没有方向（有方向的称为vecror)

（三）指针和数组

1、指针和数组

- 数组名原则上会被解释为指向该数组起始元素的指针
- 注意指针p指向的是"起始元素"，而不是"整个数组"。
- 指针p指向数组中的元素e时，
    - p + i为指向元素e后第i个元素的指针

    - p - i为指向元素e前第i个元素的指针
- 当p指向a[0]的时候，指向各元素的指针&a[i]和p + i的值是一致的

专题 10 - 2 数组名在什么情况下不被视为指向起始元素的指针

- 作为**sizeof**运算符的操作数出现时，生成数组整体的长度
- 作为取址运算符**&**的操作数出现时，指向数组整体的指针

2、指针运算符和下标运算符

- **Tpye***型指针p指向**Type**型数组a的起始元素a[0]时，指针p的行为就和数组a本身一样

3、数组和指针的不同点

- 赋值表达式的左操作数不可以是数组名

4、数组的传递

总结

十一、字符串和指针

一、字符串和指针

1、用数组实现的字符串和用指针实现的字符串

- 指针指向的不是字符串字面量，而是字符串字面量的首个字符
- 用指针实现的字符串比用数组实现的字符串需要更多的内存空间

2、用数组实现的字符串和用指针实现的字符串的不同点

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    char s[] = "ABC";
    printf("s = \"%s\"\n", s);
    s = "DEF";
    printf("s = \"%s\"\n", s);
    return 0;
}
```

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    char *p = "123";
    printf("p = \"%s\"\n", p);
    p = "456";
    printf("p = \"%s\"\n", p);
    return 0;
}
```

- 可以为指向字符串字面量（中的字符）的指针赋上指向别的字符串字面量（中的字符）的指针。赋值后，指针指向新的字符串字面量（中的字符）

3、字符串数组

- 指针指向字符串的首个字符，再通过首个字符访问整个字符串（不准确的表达）

二、通过指针操作字符串

1、判断字符串长度

```c
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int str_length(const char* s)
{
    int len = 0;

    while (*s++)
        len++;
    return len;
}

int main(void)
{
    char str[128];
    printf("请输入字符串：");
    scanf("%s", str);

    printf("字符串\"%s\"的长度是%d。\n", str, str_length(str));

    return 0;
}
```

- 指向数组元素的指针递增后将变为指向下一个元素，递减后将变为指向上一个元素
- 不管p是否为指针，`p++即p = p + 1`，`p--即p = p - 1`都成立

2、字符串的复制

将字符串s复制到d

```
char* str_copy(char* d, const char* s)
{
    char* t = d;

    while (*d++ = *s++)
        ;
    return t;
}
```

- 通过*d = *s进行赋值。指针s指向的字符会被赋值给指针d指向的字符
- 指针d和s递增。赋值结束后，d和s递增，指针d和s分别指向了下一个字符

3、不正确的字符串复制

- 不要改写字符串字面量，也不要对超过字符串字面量的内存空间进行写入操作

4、返回指针的函数

三、字符串处理库函数

1、strlen函数：求字符串的长度

- 返回不包含**null**字符在内的字符串长度

2、strcpy函数、strncpy函数：复制字符串

- 复制字符串的函数，使用后者还可以对要复制的字符数设限

3、strcat函数、strncat函数：连接字符串

- 在已有的字符串后连接别的字符串的函数，使用后者还可以对要连接的字符串个数设限

4、strcmp函数、strncmp函数：比较字符串的大小关系

- 对两个字符串的大小关系进行比较，使用后者还可以对要比较的字符数设限

5、转换字符串

- **atoi**：将指向的字符串转换为**int**型表示
- **atol**：将指向的字符串转换为**long**型表示
- **atof**：将指向的字符串转换为**double**型表示

十二、结构体

一、结构体

1、数据关联性

- 如果忘记交换，就会变得混乱

2、结构体

```
struct student{
    char name[64];
    int height;
    float weight;
    long schols;
};
```

- 结构体的名字student称为**结构名**，{}中声明的name、height等称为**结构体成员**

3、结构体成员和.运算符

- .运算符 a.b 表示结构体a的成员b
- ```
  sanaka.height   /*对象名.成员名*/
  ```

4、成员的初始化

- 声明结构体时所赋的初始值的形式是，将各个结构体成员的初始值依次排列在{}里面，并用逗号分隔。未赋初始值的成员被初始为0

5、结构体成员和->运算符

- ->运算符　　a->b　　用指针访问结构体a中的成员b
- 运算符形如箭头，因此通常称为**箭头运算符**
- 在表示指针p指向的结构体成员m时，推荐使用->运算符将（*p).m简写成p->m

6、结构体和typedef

```
tpyedef struct student{
    char name[NAME_LEN];
    int height;
    float weight;
    long schols;
} Student;
```

- 可以使用**typedef**声明为结构体赋上简洁的**typedef**名
```

7、结构体和程序

8、聚合类型

- 数组和结构体在处理多个对象的集合方面具有诸多相同点，它们统称为**聚合类型**
- 不同点
    - 数组处理“相同类型”数据的集合，结构体处理“不同类型”数据的集合
    - 数组不能相互赋值，相同类型的结构体可以相互赋值

9、返回结构体的函数

- 因为结构体可以进行赋值，所以可用作函数的返回值类型

10、命名空间

- 只要命名空间不同，就可以使用拼写相同的标识符（名字）
- 四类
    - 标签（lable）名
    - 小标签（tag）名
    - 成员名
    - 一般性标识符

11、结构体数组

12、派生类型

- 在C语言中可以组合各种方法创建出无穷的数据类型，通过这种方式创建的数据类型称为**派生类型**
- 数组类型：将某一种元素类型对象的集合分配在连续的内存单元中
- 结构体类型：按成员的声明顺序分配内存单元。各成员的数据类型可以不同
- 共用体类型：不同的成员可以放入同一段内存单元，使之相互重叠
- 函数类型：由1个返回类型和0个以上的形参及其数据类型构成
- 指针类型:创建为指向对象或函数的数据类型

二、作为成员的结构体

1、表示坐标的结构体

2、具有结构体成员的结构体

十三、文件处理

一、文件与流

1、文件与流

- 要将程序运行结束后仍需保存的数值和字符串等数据保存在**文件**中
- 针对文件、键盘、显示器、打印机等外部设备的数据读写操作都是通过**流**进行的

2、标准流

标准流有以下三种

- stdin——标准输入流
  - scanf和getchar等函数
- stdout——标准输出流
  - printf、puts与putchar等函数
- stderr——标准错误流

3、FILE型

- 文件位置指示符
- 错误指示符
- 文件结束指示符

4、打开文件

```
fp = fpoen("abc.txt", "r");
```

打开文件时可以指定以下四种模式

- 只读模式
- 只写模式
- 更新模式——既从文件输入，也向文件输出
- 追加模式——从文件末尾处开始向文件输出

5、关闭文件

```
fclose(fp)
```

6、打开与关闭文件示例

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>

int main(void)
{
    FILE* fp;

    fp = fopen("abc", "r");

```

```
    if (fp == NULL)
        printf("\a无法打开文件\"abc\"。\n");
    else
    {
        printf("\a成功打开了文件\"abc\"。\n");
        fclose(fp);
    }

    return 0;
}

```

7、文件数据汇总

从流fp中读取十进制的整数值并保存至变量x

```
fscanf(fp, "%d", &x);
```

8、写入日期和时间

向流fp写入整数x的十进制数值

```
fprintf(fp, "%d", x);
```

9、获取上一次运行时的信息

10、显示文件内容

- fgetc(fp)函数

11、文件的复制

- fputc(ch, dfp)函数

二、文本和二进制

1、在文本文件中保存实数

2、文本文件和二进制文件

3、在二进制文件中保存实数

- fwirte函数：数据写入
- fread函数：数据读取

4、显示文件自身

- isprint函数：判断字符是否为可打印字符（含空格）

三、printf函数与scanf函数

1、printf函数：带格式输出

2、scanf函数：带格式的输入