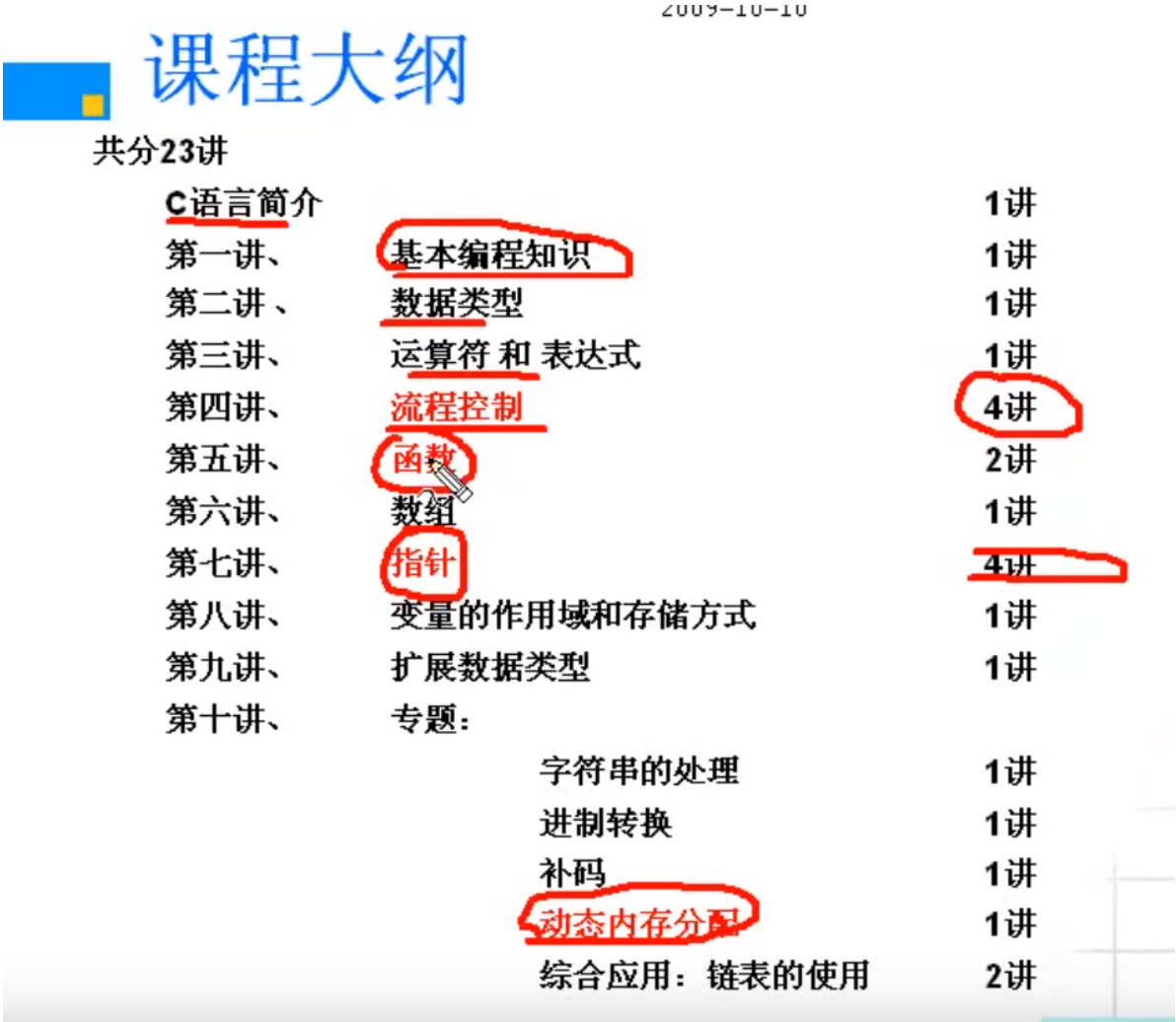
郝斌C语言

编程细节

- Visual Studio Code：选中要**缩进**的多行代码后，按下 `Tab` 键会将它们一起向右缩进，按下 `Shift + Tab` 键会将其向左缩进。
- Vc++6.0复制：右击—选择标记—选定复制内容—再右击—复制成功

课程大纲



C概述

[7.C概述 - 怎样学C语言哔哩哔哩bilibili]

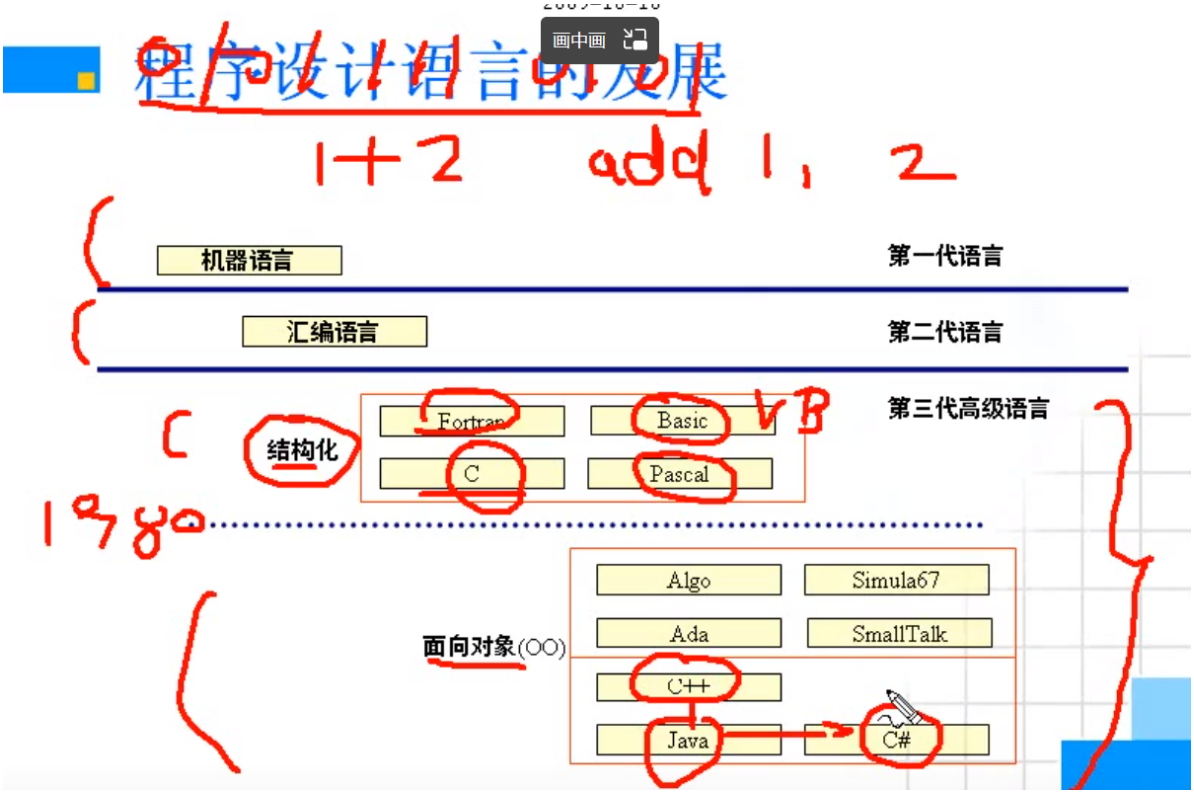
教材：C++PrimerPlus

如何使用visual c++编写代码?

创建source c++ file（文件）—输入代码—compile—build—run

```c
#include <stdio.h>
int main(void)
{
    printf("欢迎大家学习C语言!\n");

    return 0;
}
```

C语言的起源与发展



C语言特点

优点：

代码小，运行速度快。windows、unix和linux的内核都是C语言。

功能强大，有指针。

缺点：

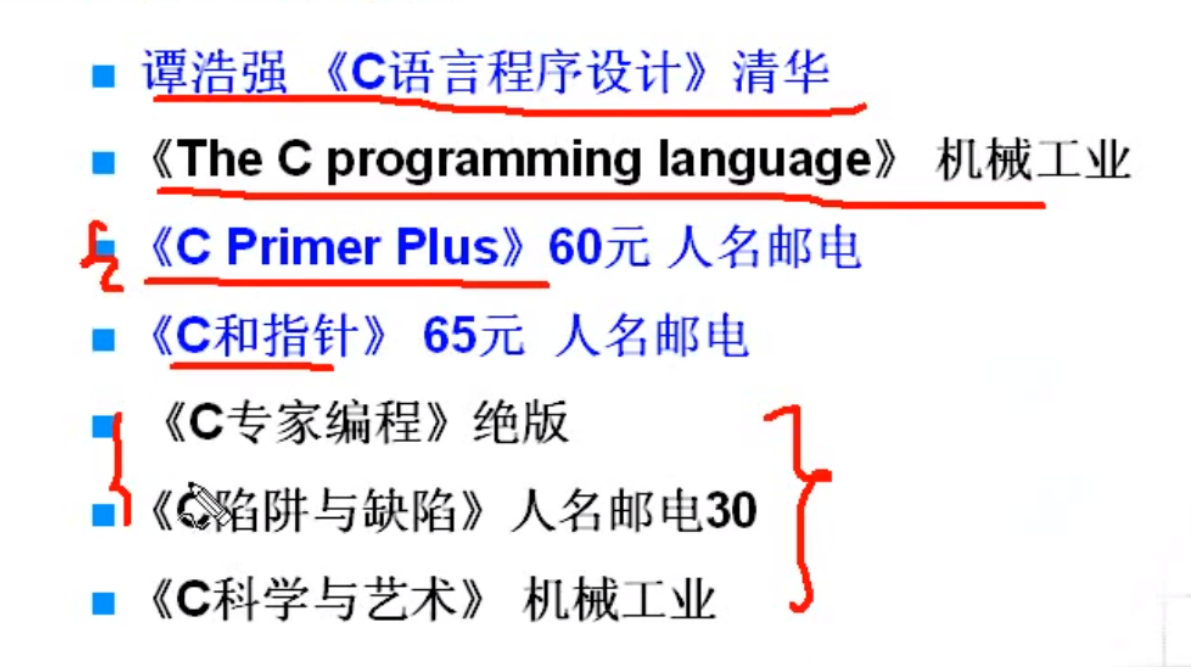
危险性大，非原则性错误不报错。

生产周期长，所以使用面向对象。

可移植性不强。Java可移植性强。

应用领域

参考资料



一元二次方程详解

1. 先写基本框架

```
#include <stdio.h>

int main(void)
{

  return 0;
}
```

2.再写思路框架，定义变量。

```
#include <stdio.h>

int main(void)
{
    //保存三个系数
    int a = 1;
    int b = 2;
    int c = 3;
    float delta;//b*b-4ac
    float x1,x2;//两个解
    delta = b*b-4ac;

    if(delta > 0)
    {
       两个解
    }else if(delta == 0)
    {
       唯一解
    }
    else
    {
       无解
    }


    return 0;
}
```

3.补充内容，补足引用。

```
#include <stdio.h>
#include <math.h>
int main(void)
{
    //保存三个系数
    int a = 1;
    int b = 2;
    int c = 3;
    double delta;//b*b-4ac
    double x1;
    double x2;//两个解
    delta = b*b-4*a*c;

    if(delta > 0)
    {
       x1 = (-b+sqrt(delta))/(2*a);
       x2 = (-b-sqrt(delta))/(2*a);
       printf("该一元二次方程有两个解,x1=%f,x2=%f\n",x1,x2);

    }else if(delta == 0)
    {
       x1 = -b/(2*a);
       printf("该方程只有唯一解x1=x2=%f\n",x1);
    }
    else
```

```
    {
       printf("无解\n");
    }


    return 0;
}
```

VC++ 6.0使用详解

1.保存：CTRL+S

2.代码规范：记得注释和空格、成对地打括号、注意缩进和换行（独立的部分）。

3.关闭程序，要使用关闭工作空间，否则影响第二个程序运行。

编程预备专业知识

hello world是如何运行起来的

compile+build 把文件变成了.exe形式，然后虹色感叹号请求操作系统运行。

数据类型

简单类型数据：

整数

整型 —— int、短整型 —— short、长整型 —— long int（4、2、8字节）

浮点数

float、double float（4、8）

字符

char （1）

```
//常见错误
char ch = 'a'
char ch = 'b'//error:重复定义

char ch = 'ab'//'ab'是错误的
```

复合类型数据

结构体

枚举

变量

数据是内容，变量是容器，数据类型是内容的性质。

或者说变量就是字母，对应内存的空闲单元，是我对内存单元的命名。

变量必须初始化

cpu将程序从外存调入内存运行，内存为其分配一定空间。运行时，空间不允许其他程序使用，运行结束后释放空间，但不清理数据。如果不初始化，就使用遗留的垃圾数据，没有意义。

常量的表示

整数（补码）

十六进制：前面加ox、八进制：前面加0.

小数（IEEE）

传统、科学计数法e

字符（ASCII码和补码）

单个字符用单引号括起来

字符串用双引号括起来

“A”表示一个‘A’和‘/0’，字符串用/0表示结束。

字节（B）

字节就是存储数据的单位，是硬件所能访问的最小单元。

ASCII码

ASCII码不是一个值，是一种规定。规定不同的字符用哪个整数值去规定。

所以，字符的存储本质上与整数的存储方式相同。

printf用法详解

```
printf("字符串\n");//\n表示换行

printf("输出控制符\n",输出参数);

printf("输出控制符1 输出控制符2\n", j, k);

printf("输出控制符  非输出控制符",x);#非输出控制符原样输出

为什么需要输出控制符
1.01代码可以表示数据，也可以表示指令。
2.输出数据需要指明格式。
```

1. 换行符（\n）：将输出移动到下一行的开头，并继续在新行上输出内容。
2. 制表符（\t）：在输出中插入一个制表符，通常等于八个空格字符，用于在不同的列位置对齐输出。
3. 回车符（\r）：将输出的位置移动到当前行的开头，覆盖之前的内容。
4. 退格符（\b）：将输出的位置向后移动一个字符，可用于删除之前的字符。
5. 换页符（\f）：将输出的位置移动到下一页的开头。

输出控制符：

1. 整数类型：
    - %d：输出带符号的十进制整数。
    - %u：输出无符号的十进制整数。

- %o：输出八进制整数。
- %x：输出十六进制整数，使用小写字母。//%#x输出时带有ox
- %X：输出十六进制整数，使用大写字母。

2. 浮点数类型：
   - %f：输出浮点数。
   - %e：用科学计数法输出浮点数（小写字母e）。
   - %E：用科学计数法输出浮点数（大写字母E）。
   - %g：根据数值的大小自动选择%f或%e格式输出浮点数。
   - %G：根据数值的大小自动选择%f或%E格式输出浮点数。
   - %.nf：限制浮点数输出的小数位数，n为小数位数。
3. 字符类型：
   - %c：输出单个字符。
   - %s：输出字符串。
4. 指针类型：
   - %p：输出指针的地址。

如何写出更有保存价值的代码

```
/*
时间
目的
功能
*/



/*
在某软件上的输出结果是：
--------------------

-------------------
*/
```

scanf的用法

```
#输入控制符必须有合法的输入
（1）scanf("输入控制符"，&输入参数)；
#非输入控制符，必须原样输入
（2）scanf("m%d",&i);//m123是正确输入，123是错误输入
---------------------------------------
注意：1.使用scanf之前可以先使用printf书写提示，告知用户如何输入。2.scanf中尽量不要使用非输入控制符。3对用户的非法输入做适当的处理。
---------------------------------------
```

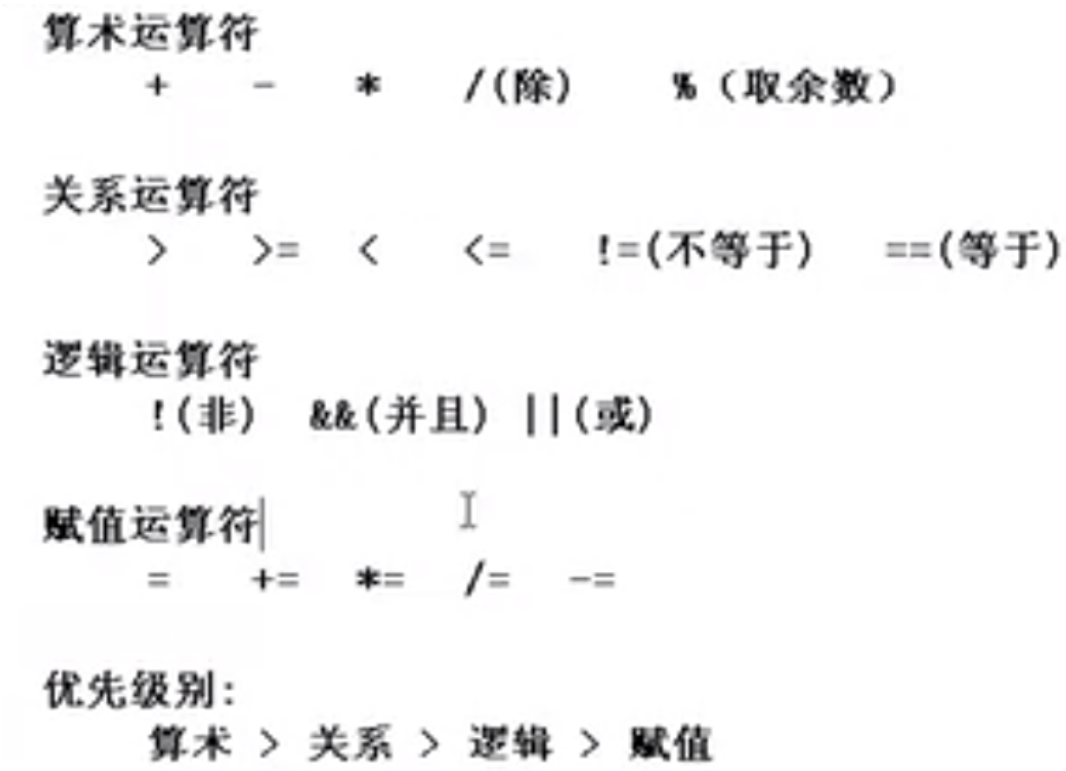
```
#include <stdio.h>

int main(void)
{
    int i;

    scanf("%d",&i);

    printf("i = %d\n", i);
```

```
    int j;

    scanf("%d",&j);

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

    return 0;

}
```

输入123m时，输出i=123,j为乱数
输入m123时，i和j都是乱数

总结：输错的字符不会自动丢弃。
读到合法的会继续往后面读，读到错误的就停止了。把剩下的给后面的scanf。
一开始读到不合法的，直接报错。传给后面的scanf。

运算符



注意：（1）被除数和除数都是int，那么商也是int。如果有一个是float，那么商就是float。（2）取余的运算对象必须是整数。（3）在C语言中非零是真，零是假，真用“1”表示，假用“0”表示。（4）&&左边为假，右边不执行。||左边为真时，右边不执行。

自增（自减）

分类：前自增——++i

后自增——i++

相同点：都是使得i加1

区别：前自增整体表达式的值是i+1之后的值

后自增整体表达式的值是i+1之前的值

```
#include <stdio.h>

int main(void)
{
    int i=3;
    int j;
    int k;
    int m=3;

    j = ++i;
    k = m++;

    printf("i=%d j=%d m=%d k=%d\n",i, j, m, k);

    return 0;
}

/*
i=4 j=4 m=4 k=3
*/
```

注意：1.编程时尽量屏蔽前自增和后自增的区别

2.自增单独成句，不要成为复合句的一部分

3.i++是个临时值，i+=1是个运算赋值表达式

三目运算符

```
A ? B : C
等价于
if（A）
  B;
else
  C;
```

逗号表达式

```
格式
（A, B, C, D）
功能：
从左到右执行
最终表达式的值是最后一项的值
```

流程控制

什么是流程控制?

代码和程序的执行顺序。

顺序

循环

选择

定义： 有些代码可能执行，也可能不执行。

分类:

if

1.if最简单的用法

```
if(表达式)
        语句;
```

2.if的范围问题

if默认只能控制一个语句的执行或不执行。若想控制多个语句的执行，则使用{}把这些语句括起来。

3.if...else...的用法

4.if...else...if...else...的用法

注意语句的连贯性

5.if举例——求分数的等级

6.if常见问题解析

```
//空语句
if(表达式);

//没有else开头的语句
if();
A;
else B;

//else （表达式）
if （表达式）
A;
else （表达式）
B;
```

看懂程序

1.流程

2.每个语句的功能

3.试数

循环

定义：某些代码会被重复执行

分类：**for**

```
//for的简单使用
for (变量赋值; 循环条件; 变量变化)
{
循环体
}

//for和if的嵌套使用，注意范围问题

//多个循环的嵌套使用,注意顺序和范围
for (1; 2; 3)
  for (4; 5; 6)
     A;
     B;
```

float和double不能保证准确存储一个小数。

循环中更新的变量不能定义为浮点型。

while

```
while（表达式）
    语句;

//for和while可以相互转换
for（1； 2； 3）
    A;
等价于
1;
while（2）
{
   A;
   3;
}

//do...while主要用于人机交互
格式:
do
{
...
}while（表达式）
功能：先执行do里面的，然后判断while，再决定是否执行do语句。
```

强制类型转换

```
格式:（数据类型）（表达式）
```

switch

```
switch(表达式)
{
case 常量表达式1:
    printf("");
    break;
case 常量表达式2:
```

```
    printf("");
    break;
case 常量表达式3:
    printf("");
    break;
default:
    printf("");
    break;
}
//case是个入口，进入后程序从上往下执行，break是个出口。
```

break和continue

break：

（1）用来终止循环和switch。在嵌套的循环和switch中终止最近的那个。

（2）只能间接用于if

continue：

```
//功能：跳过循环体的后续语句，转而进行判断。
//对用户非法输入的处理
while ((ch = getchar()) != "\n")
    continue;
```

数组

```
int a[3] = {1, 2, 3}
//int是数组元素的类型，a是数组的名字，3表示元素的个数，分别用a[0]、a[1]、a[2]表示
```

为什么需要数组

（1）解决大量同类型数据的存储和使用问题

（2）为了模拟现实世界

分类：

（1）存储空间必须是连续分配

（2）所有元素类型相同并且占有相同的字节数

（3）只有在定义数组的时候才可以整体赋值

一维数组

```
//初始化
//完全初始化
int a[3] = {1, 2, 3};
//不完全初始化，未被初始化的元素自动为零
int a[3] = {1, 2};
//不初始化，所有元素都是垃圾
int a[3];
//清零
int a[3] = 0;
//把a数组的元素全部赋值给b数组
for(i=1; i<4; i++)
    b[i] = a[i];
```

```
//赋值
int a[5];
scanf("%d", &a[1]);
printf("a[1]=%d", a[1]);
```

二维数组

```
//int a[i][j]表示数组的i+1行，j+1列
//初始化
int a[3][4]={
     {1, 1, 1, 1},
     {2, 2, 2, 2},
     {3, 3, 3, 3}
};
//输出数组
int a[3][4]={
     {1, 1, 1, 1},
     {2, 2, 2, 2},
     {3, 3, 3, 3}
};

int i.j;

for(i=0; i<3; ++i)
{
    for(j=0; j<4; ++j)
        printf("%d  ", a[i][j]);
    printf("\n");
  }
```

多维数组

不存在，因为内存是一维的。

n维数组可以当做每个元素是n-1维数组的一维数组。

函数

（1）简单使用

```
#include <stdio.h>

void max(int i,int j)
{
    if(i > j)
        printf("最大值是%d\n", i);
    else
        printf("最大值是%d\n", j);
}
//格式：void表示函数不返回值，max是函数名，i和j是形参
//符合参数条件，才可能调用函数
//函数的作用：对不同数据对象的相同功能的操作的封装

int main(void)
{
    int a, b, c, d, e, f;
    a=1, b=3, c=5, d=0, e=20, f=-10;
```

```
    max(a,b);
    max(c,d);
    max(e,f);

    return 0;
}
//main是函数名，void表示函数不能接收数据，int表示返回值是int类型

/*
运行过程：
从main进入
调用max函数，为参数分配空间，把值赋给形参
执行操作，从max函数出来，释放空间
执行完毕，从main函数出来
*/

//后续分配的空间不一定是之前分配的空间
```

（2）为什么需要函数

避免了重复性操作，有利于程序的模块化。

（3）定义函数

函数返回值类型 函数的名字（函数的形参列表）

{

函数的执行体

}

注意：（1）return 表达式；的类型与函数返回值类型不同，以后者为准。

（2）return终止被调函数，而break终止循环和switch

（4）函数分类

返值函数和不返值函数

普通函数和主函数

1.一个程序必须有且只有一个主函数

2.主函数可以调用普通函数，普通函数不能调用主函数，普通函数之间可以互相调用。

3.主函数是程序的入口，也是程序的出口。

（5）函数声明

当函数调用出现在函数定义前面时，需要函数前置声明。

```
#include <stdio.h>

void f(void);

int main(void)
{
    f();
    return 0;
}
```

```
void f(void)
{
    printf("哈哈! \n");
}

//对库函数的声明是通过#include <stdio.h>实现的。
```

功能:

1.告诉编译器即将可能出现的若干字母代表的是一个函数。

2.告诉编译器即将可能出现的若干字母代表所代表的函数的形参和返回值的具体情况。

3.函数声明是一个语句，后面必须加分号。

在开发中合理设计函数

```
//求某个数以内的素数
//只用一个main函数
#include <stdio.h>

int main(void)
{
    int i;
    int j;
    int k;
    scanf("%d",&k);
    //把2到k之间的数判断是否是素数
    for(i=2; i<k; ++i)
    {
    //这个for判断是否是素数
    for(j=2; j<i; ++j)
        {
            if(i%j == 0)
               break;
        }
        if(j == i)
            printf("%d\n", i);
    }

    return 0;
}

//进阶版
#include <stdio.h>

bool Isprime(int k)
{
    int i;
    for(i=2; i<k; ++i)
        {
            if(k%i == 0)
                break;
        }
        if(k == i)
            return true;
        else
```

```
            return false;

}

int main(void)
{
    int i;
    int val;

    scanf("%d",&val);
    for(i=2; i<val; ++i)
    {
        if(Isprime(i))
        //if(Isprime(i) == true)是错误的
            printf("%d\n", i);

    }

    return 0;
}

//高阶版
#include <stdio.h>

bool Isprime(int k)
{
    int i;
    for(i=2; i<k; ++i)
        {
            if(k%i == 0)
                break;
        }
        if(k == i)
            return true;
        else
            return false;

}

//把1到val的素数全部输出
void prime(int val)
{
    int j;
    for(j=2; j<val; ++j)
    {
        if(Isprime(j))
            printf("%d\n", j);

    }

}

int main(void)
{
    int m;

    scanf("%d",&m);
    prime(m);
```

```
    return 0;
}

/*
（1）用功能独立单一的函数来实现，代码的可重用性提高。
（2）只使用main函数的局限性：
     1.代码的重用性不高。
     2.代码不容易理解。
（3）bool函数返回true和false，与if表达式结果一致，不需要判断。
（4）定义多个函数，方便调用需要的功能。
（5）注意是否需要使用函数。
*/
```

变量

按作用域分

全局变量：定义在所有函数外的变量。

作用范围：定义位置到函数结束处。

局部变量：定义在函数内部的变量或者是函数的形参。

作用范围：只能在函数内部使用。

注意：当局部变量名与全局变量名相同时，局部变量会屏蔽全局变量。

指针

简单介绍

1.指针就是地址，地址就是指针。

2.地址是内存单元的编号，是个值。

3.指针变量是存放地址的变量。

4.指针和指针变量是两个不同的概念，但是有时会把指针变量简称指针。

5.指针是一个操作受限的非负整数。

```
#include <stdio.h>

int main(void)
{
    int * p;//p是变量的名字，int *表示p变量存放的是int类型变量的地址。
    int i = 3;

    p = &i;//p只能存放变量地址。

    /*
    1.p存放i的地址，p就指向i。
    2.p不是i，i也不是p。修改i不改变p，修改p也不改变i。
    3.前提：如果一个指针变量指向了某个普通变量
    则，*指针变量  完全等同于  普通变量
    也就是说，*p可以和i互换。
```

```
    */

    return 0;
}
```

常见错误

```
# include <stdio.h>

int main(void)
{
    int * p;
    int i = 3;
    *p = i;
    printf("%d\n", *p);

    return 0;
}
/*
p变量没有赋值，存储了乱值。就等同于随便指向了一个地址。
此时，*p = i，就改变了指向地址的数据，那么可能导致程序崩溃。
*/
```

```
# include <stdio.h>

int main(void)
{
    int * q;
    int * p;
    int i = 3;
    p = &i;
    *q = p;
    //*q = *p 错误同1
    printf("%d\n", *q);

    return 0;
}
/*
p是一个int*型的变量。
*q是一个int型的变量。
*/
```

互换

```
# include <stdio.h>
/*
void change(int a,int b)
{
    int t;
    t = a;
    a = b;
    b = t;
}
输出结果：3 5
原因：从主函数进入，为变量m，n分配空间。
```

```
然后进入到函数change，为形参分配空间，并把m和n的值赋给形参。
形参的值进行互换，执行完后释放空间，跳出change。
m和n的空间还在，打印输出，然后释放空间。
*/

/*
void change(int * a,int * b)
{
    int * t;
    t = a;
    a = b;
    b = t;
}
互换了变量a和b储存的m和n的地址，没有改变m和n的值。
*/

void change(int * a,int * b)
{
    int t;
    t = * a;
    * a = * b;
    * b = t;
}

/*
输出结果：5 3.
如何通过被调函数修改主函数的值：
（1）实参是地址，形参是int *
（2）* 形参 = ...
*/

int main(void)
{
    int m=3;
    int n=5;
    change(&m,&n);
    printf("%d %d\n",m,n);

    return 0;
}
```

数组

一维数组名的含义

```
# include <stdio.h>

int main(void)
{
    int a[5];
    printf("%#X\n", &a[0]);
    printf("%#X\n", a)

    return 0;
}

/*
```

```
输出结果：
0X19FF20
0X19FF20

一维数组的名字是个指针常量
它存放的是一维数组的第一个元素的位置
*/
```

确定一个一维数组需要两个参数及其原因

```c
//输出任何一个一维数组
# include <stdio.h>

void f(int * p,int len)
{
    int i;
    for(i=0; i<len; ++i)
        printf("%d\n", *(p+i));
}

int main(void)
{
    int a[5] = {1, 2, 3, 4, 5};
    int b[6] = {-1, -2, -5, 66, 20};
    int c[100] = {22, 25, 34, 46};

    f(a,5);
    f(b,6);
    f(c,100);

    return 0;
}
/*
因为数组是连续存放的，只需要数组变量名（第一个元素的存放地址）和数组的长度。
*/
```

```c
//换值
# include <stdio.h>

void f(int * p,int len)
{
    p[3] = 88;

}

int main(void)
{
    int a[5] = {1, 2, 3, 4, 5};
    printf("%d\n",a[3]);
    f(a,5);
    printf("%d\n",a[3]);

    return 0;
}
/*
a是数组名（第一个值的存放地址）。a[3]等同于*（a+3）。
```

```
*/
```

指针的运算

不能相加、相减和相除。

两个向量只有指向同一连续空间存储的不同存储位置时才可以相减。

其结果是两个单元之间所隔单元数。

指针变量的大小

```
# include <stdio.h>

int main(void)
{
    char i = 'A';
    int j = 2;
    double m = 66.6;
    char * p = &i;
    int * q = &j;
    double * r = &m;

    printf("%d %d %d\n", sizeof(p), sizeof(q), sizeof(r));

    return 0;
}
/*
输出结果：4 4 4
（1）指针变量大小都是相同的，都是4B.
（2）指针变量都是指向存储空间的第一个地址，所以需要定义类型指明长度。这样 *指针变量 表示占数个byte的变量。
*/
```

动态内存分配

（1）传统数组的缺点（不灵活）

- 传统数组需要预先设定长度，且不能用变量表示。
- 传统形式定义的数组，该数组的内存程序员无法手动释放。在一个函数运行期间，系统为该函数中数组所分配的空间会一直存在，直到该函数运行完毕时，数组的空间才会被系统释放。
- 数组的长度一旦定义，其长度不能改变。
- A函数定义的数组，在A函数运行期间可以被其他函数使用，但A函数运行完毕后，A函数的数组将无法被其他函数使用。

（2）malloc函数使用的简单介绍

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

int main(void)
{
    int i = 5;//11行
    int * p = (int*)malloc(4);//12行
    *p = 5;//*p是一个int型的变量，由12行分配了4B的空间，与11行静态分配不同。
    free(p);
```

```
    return 0;
}
/*
1.malloc函数定义在<malloc.h>中，开头必须有# include <malloc.h>。
2.malloc函数只有一个形参，并且形参一定是整型。
3.malloc函数返回给p向量第一个字节的地址，所以前面要定义类型。
4.4表示请求系统为本程序分配4个字节。
5.12行分配了8个字节，动态4个，静态4个。
*/
```

(3) 动态创造一维数组

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

int main(void)
{
    int i;
    int len;
    printf("数组的个数：");
    scanf("%d",&len);
    int * parr = (int*)malloc(4*len);

    for(i=0; i<len; ++i)
        scanf("%d",parr+i);//&parr[i]

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

    return 0;
}
```

(4) 多级指针

```
# include <stdio.h>

int main(void)
{
    int i = 5;
    int * p = &i;
    int ** q = &p;
    int *** r = &q;
    printf("%d\n", p);
    printf("%d\n", *q);
    printf("%d\n", ***r);
    return 0;
}
/*
int ** q
*q等于p
q是个int **类型的变量，指向指针的指针，也就是地址的地址。
*/
```

(5) 静态变量不可以跨函数使用详解

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

```c
void f(int ** q)
{
    int i = 5;
    *q = &i;//*q=p，*p=i
}

int main(void)
{
    int * p;//p是一个一级指针变量
    f(&p);//一级指针变量的地址，需要二级指针储存
    printf("%d\n",*p);//语法没错误但是逻辑有错误
    return 0;
}
/*
当函数执行完毕，为静态变量i申请的空间就会释放，*p无权访问i变量。
*/
```

（6）动态内存可以跨函数使用

```c
# include <stdio.h>
# include <malloc.h>

void f(int ** q)
{
    //int * i = (int*)malloc(sizeof(int));
    //*i = 5;
    //**q = *i;
    *q = (int*)malloc(sizeof(int));
    **q = 5;
}

int main(void)
{
    int * p;
    f(&p);
    printf("%d\n",*p);
    return 0;
}
/*
（1）动态分配是在堆里面分配，而函数执行是入栈出栈，静态空间是在栈里面分配的。
（2）指针变量没有初始化，*p没有值。
*/
```

结构体

什么是结构体

```c
# include <stdio.h>

struct student
{
    int age;
    float score;
    char sex;
};
```

```c
int main(void)
{
    struct student st = {80, 66.6, 'F'};

    return 0;
}
/*
（1）为什么需要结构体
结构体可以表达基本数据类型不能表达的复杂事物。
（2）什么是结构体
结构体就是普通数据类型的集合，组成了一个新的复合数据类型。
*/
```

如何定义结构体

```c
//推荐
struct student1
{
    int age;
    float score;
    char sex;
};
struct student st1 = {80, 66.6, 'F'};

//不推荐
struct student
{
    int age;
    float score;
    char sex;
}st2;

//不推荐
struct
{
    int age;
    float score;
    char sex;
}st3;
```

初始化和赋值

```c
# include <stdio.h>

struct student
{
    int age;
    float score;
    char sex;
};

int main(void)
{
    struct student st = {80, 66.6, 'F'};//定义的时候可以整体赋初值。
    struct student st1;//定义后只能分别赋值。
```

```
    st1.age = 40;
    st1.score = 44.4;
    st1.sex = 'F';

    printf("%d %f %c\n",st.age, st.score, st.sex);
    printf("%d %f %c\n",st1.age, st1.score, st1.sex);

    return 0;
}
```

如何取出结构体变量中的每一个成员

```
# include <stdio.h>

struct student
{
    int age;
    float score;
    char sex;
};

int main(void)
{
    struct student st = {80, 66.6, 'F'};
    st.age = 10;//变量名.成员名
    struct student * pst = &st;
    pst->score = 56.6f;//指针变量名->成员名，后面f是把double类型变成float类型。

    printf("%d %f %c\n",st.age, pst->score, st.sex);

    return 0;
}
/*
这两种方式等价
因为指针变量名->成员名 等同于 （*指针变量名）.成员名。（计算机内部会强制转化）
*/
```

通过函数完成对结构体变量的输入和输出

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

struct student
{
    int age;
    float score;
    char name[100];//这意味着您可以在该数组中存储长度为99个字符以内的字符串。
};

void input(struct student * pst)
{
    (*pst).age = 40;
    pst->score = 56.6f;
    strcpy(pst->name,"张三");//不能直接赋值，要用<string.h>里的strcpy函数,把值复制到变量
中。
}
```

```
void output(struct student ss)
{
    printf("%d %f %s\n",ss.age, ss.score, ss.name);
}

int main(void)
{
    struct student st;
    input(&st);//函数使用时，不要带类型。
    output(st);

    return 0;
}
/*
输出时不改变值，不需要用到指针。
*/
```

运算

结构体变量不能加减乘除，但可以相互赋值。

冒泡排序

```
# include <stdio.h>

/*
先选定一个值，然后后面的值跟他比较，然后互换位置，再跟后面的值比较，确定了一个值的位置。如此循环，可以确定所有值的位置。
*/
void sort(int * a, int len)
{
    int i, j, t;
    for(i=0; i<len; ++i)
    {
        for(j=0; j<len-i-1; ++j)
        {
            if(a[j] < a[j+1])//>就是升序排列。
            {
                t = a[j+1];
                a[j+1] = a[j];
                a[j] = t;
            }
        }
    }

}

int main(void)
{
    int i;
    int a[5] = {3, 8, 4, 6, -1};
    sort(a, 5);
    for(i=0; i<5; ++i)
        printf("%d\n", a[i]);

    return 0;
```

```
}
```

学生管理系统

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

struct student
{
    int age;
    float score;
    char name[100];
};//没有struct student st这种定义方式。

void input(struct student * p, int l)//写完函数注意引用了什么外界变量，作为形参。
{
    int i;
    for(i = 0; i < l; ++i)
    {
        printf("请输入第%d个学生的信息：", i+1);
        printf("请输入学生年龄：");
        scanf("%d", &p[i].age);
        printf("请输入学生分数：");
        scanf("%f", &p[i].score);
        printf("请输入学生姓名：");
        scanf("%s", p[i].name);       // 修正读取学生姓名方式
    }
}

void sort(struct student * p, int k)
{
    int i, j;             // 声明循环计数器，记得检查变量是否定义完全。
    struct student t;     // 声明临时变量
    for(i = 0; i < k; ++i)
        for(j = 0; j < k-i-1; ++j)
        {
            if(p[j].score < p[j+1].score)
            {
                t = p[j+1];
                p[j+1] = p[j];
                p[j] = t;
            }
        }
}

void output(struct student * p, int m)
{
    int i;
    for(i = 0; i < m; ++i)
    {
        printf("第%d个学生的信息为：\n", i+1);
        printf("姓名是%s\n", p[i].name);    // 修正输出学生姓名格式，字符串是s
        printf("学生分数=%f\n", p[i].score);
        printf("学生年龄=%d\n", p[i].age);
    }
}
```

```c
int main(void)
{
    //输入
    int len;
    printf("统计的学生数量：");
    scanf("%d", &len);
    struct student * parr = (struct student*)malloc(len * sizeof(struct
student));
    input(parr, len);

    //排序
    sort(parr, len);

    //输出
    output(parr, len);

    free(parr); // 记得释放内存
    return 0;
}
/*
功能：统计学生信息，然后按照分数排名输出。
算法：输入、排序、输出
*/
```

枚举

把一个事物可能的取值一一列举出来。

```c
# include <stdio.h>

enum weekday//enum就是枚举，格式类似结构体。
{
    Monday=4, Tuesday, Wednesday, Thurday, Friday, Saturday, Sunday
};

//enum weekday//enum就是枚举，格式类似结构体。
//{
//    Monday, Tuesday, Wednesday, Thurday, Friday, Saturday, Sunday
//};

int main(void)
{
    enum weekday day = Sunday;
    printf("%d\n",day);

    return 0;
}
```

补码

原码：符号-绝对值码。

反码：计算机不用。

移码：表示数值平移n位，n称为移码量。移码主要用于浮点数的阶码的存储。

补码

十进制转二进制

正整数转二进制

除二取余法

负整数转二进制

先求负整数对应正整数的二进制代码，然后将所有位取反，末尾+1.不够位数时，左边补1。

零转二进制

全是0.

二进制转十进制

如果首位是0，按原码求。

如果首位是1，将所有位取反，末尾加1，所得数字就是该负数的绝对值。