**流程控制【是学习C语言的第一个重点】**

**1.什么是流程控制**

**程序代码执行的顺序**

**2.流程控制的分类**

**顺序**

**选择**

**定义**

**某些代码可能执行，也可能不执行，有选择的执行某些代码**

**分类**

**if**

1. **if最简单的用法**

**格式：**

**if（表达式）**

**语句**

**功能：**

**如果表达式为真，执行语句**

**如果表达式为假，语句不执行**

1. **if的范围问题**

**1.**

**if（表达式）**

**语句A**

**语句B**

**解释：if默认只能控制语句A的执行或不执行**

**if无法控制语句B的执行或不执行**

**2.**

**if（表达式）**

**{**

**语句A;**

**语句B;**

**}**

**此时if可以控制语句A和语句B**

**由此可见：if默认只能控制一个语句的执行或不执行**

**如果想控制多个语句的执行或不执行**

**就必须把这些语句用{}括起来**

1. **if…else…的用法**
2. **if…else if…else…的用法**

**格式：**

**if (1)**

**A;**

**else if (2)**

**B;**

**else if (3)**

**C;**

**else**

**D;**

1. **C语言对真假的处理**

**非零是真**

**零就是假**

**真用1表示**

**假用0表示**

1. **if举例-----求分数的等级**
2. **If的常见问题解析**
3. **空语句的问题**

**if (3 > 2);**

**等价于**

**If (3 > 2)**

**; //这是一个空语句**

1. **if （表达式）; //如果这里有分号，会导致程序有错误**

**A;**

**else //else前没有对应if**

**B;**

1. **if (表达式1)**

**A;**

**else if (表达式2)**

**//当表达式1和表达式2都成立时，输出表达式1**

**B;**

**else if (表达式3)**

**C;**

**else**

**D;**

**4.**

**if (表达式1)**

**A;**

**else if (表达式2)**

**B;**

**else if (表达式3)**

**C;**

**这样写语法不会出错，但逻辑上有漏洞**

**5.**

**if (表达式1)**

**A;**

**else if (表达式2)**

**B;**

**else if (表达式3)**

**C;**

**else (表达式4) //7行**

**D;**

**这样写是不对的，正确的写法是：**

**要么去掉7行的（表达式4）**

**要么在7行的else后面加if**

**6.**

**if (表达式1)**

**A;**

**else if (表达式2)**

**B;**

**else if (表达式3)**

**C;**

**else (表达式4);**

**D;**

**这样写语法不会出错，但逻辑上是错误的**

**else (表达式4);**

**D;**

**等价于**

**else**

**（表达式4）;**

**D;**

**switch**

**把文件 switch 看懂就ok了**

**循环**

**定义：**

**某些代码会被重复执行**

**分类**

**for**

1. **格式：**

**for(1; 2; 3)**

**语句A; //执行顺序12A32A32A32……2**

1. **执行的流程【重点】**

**单个for循环的使用**

**多个for循环的嵌套使用**

**1. for (1;2;3)**

**for (4;5;6)**

**A;**

**B; //一共两个语句，B是第二个语句**

**2.** **for (1;2;3)**

**for (4;5;6)**

**{**

**A;**

**B;**

**} //整体是一个语句**

**3. for (1;2;3)**

**{**

**A;**

**B;**

**for (4;5;6)**

**C;**

**} //整体是一个语句**

1. **范围问题**

**for 默认只循环一个语句**

1. **举例：**

**1+2+3+……+100**

**1+1/2+1/3+……+1/100**

**while**

1. **执行流程**

**格式：**

**while(表达式)**

**语句; //默认控制一个语句**

1. **与for的相互比较**

**for和while可以相互转换**

**for (1;2;3)**

**A;**

**等价于**

**1;**

**while(2)**

**{**

**A;**

**3;**

**}**

**for逻辑性更强，更不容易出错，推荐多使用for**

1. **while举例**

**从键盘输入一共数字，如果该数字是会回文数**

**则返回yes**

**否则返回no**

1. **什么时候使用while 什么时候使用for**

**没法说，用多了自然会知道**

**只可意会，不可言传**

**do……while**

**格式：**

**do**

**{**

**…… (语句)**

**} while(表达式); //先执行语句，再判别表达式**

**do……while 并不等价于for 和 while**

**主要用于人机交互**

**例子： 一元二次方程**

**break和continue**

**break**

**break如果用于循环是用来终止循环**

**break如果用于switch，则是用于终止switch**

**break不能直接用于if，除非if属于循环内部的一个子句**

**例子：**

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

{

if (3 > 2)

break;

//break 虽然为 if 的子语句，但是却用来终止 for 语句

printf("嘿嘿！\n"); //不会输出

}

**在多层循环中，break只能终止离它最近的循环**

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

{

for (j = 1; j < 3; ++j)

break;

//break只能终止离它最近的循环

printf("哈哈！\n"); //输出三遍

}

**在多层switch嵌套中，break只能终止离它最近的switch**

**continue**

**用于跳过本次循环余下的语句**

**转去判断是否需要执行下次循环（严格说不是十分准确）**

**例子：**

**1. for(1;2;3)**

**{**

**A;**

**B;**

**continue;**

**//如果执行该语句，则执行完该语句后，会执行语句3，C和D都会被跳过去，不会被执行**

**C;**

**D;**

**}**

**2. while(表达式)**

**{**

**A;**

**B;**

**continue;**

**//如果执行该语句，则执行完该语句后，会执行表达式，C和D都会被跳过去，不会被执行**

**C;**

**D;**

**}**

**小算法的程序**

**判断一个数字是否是素数**

**判断一个数字是否是回文数（正着敲和倒着敲一样，如1221）**

**编程实现求一个十进制数字的二进制形式**

**求一个数字的每位是奇数的数字取出来组合形成新的数字**

**求一个数字倒过来的数字**

**C语言常见误区，如何看懂一个程序，分三步**

1. **流程**
2. **每个语句的功能**

1. **试数**

**如何学习一些小算法的程序【如何掌握一个程序】**

**尝试自己去编程解决它，大部分人都自己无法解决**

**如果解决不了，就看答案**

**关键是把答案看懂，这个是要花很大的精力，也是我们学习的重点**

**看懂之后尝试自己去修改程序，并且知道修改之后程序的不同输出结果的含义**

**照着答案去敲**

**调试错误**

**不看答案，自己独立把答案敲出来**

**如果程序实在无法彻底理解，就把它背会**

**强制类型转化**

**格式：**

**(数据类型）(表达式）**

**功能：**

**把表达式的值强制转化为前面所指定的数据类型**

**例子：**

**(int)(4.5 + 2.2) 最终值是 6**

**(float)(5) 最终值是 5.000000**

**数组**

**为什么需要数组**

**为了解决大量同类型数据的存储和使用问题**

**为了模拟现实世界**

**数组的分类**

**一维数组**

**怎样定义一维数组**

**为n个变量连续分配存储空间**

**所有的变量数据类型必须相同**

**所有变量所占的字节大小必须相等**

**一维数组名不代表数组中所有的元素**

**一维数组名代表数组第一个元素的地址**

**有关一维数组的操作**

**初始化**

**完全初始化**

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

**不完全初始化，未初始化的元素自动为零**

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

**不初始化,所以元素是垃圾值**

**int a[5];**

**清零**

**int a[5] = [0];**

**错误写法：**

**int a[5];**

**a[5] = {1,2,3,4,5}; //错误**

**只有在定义数组的同时才可以整体赋值，**

**其他情况下整体赋值都是错误的**

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

**a[5] = 100; //error 因为没有a[5]这 个元素最大只有a[4].**

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

**int b[5];**

**如果要把a数组中的值全部赋给b数组**

**错误的写法**

**b = a; // error**

**正确的写法**

**for (i = 0 ;i < 5;++i)**

**b[i] = a[i];**

**赋值**

**排序**

**求最大/最小值**

**倒置**

**查找**

**插入**

**删除**

**二维数组**

**int a[3][4];**

**总共是12个元素，可以当做三行四列看待，这12个元素的名字 依次是a[0][0] a[0][1] a[0][2] a[0][3]**

**a[1][0] a[1][1] a[1][2] a[1][3]**

**a[2][0] a[2][1] a[2][2] a[2][3]**

**a[i][j] 表示第i + 1行 j + 1列的元素**

**int a[m][n]; 该二维数组右下角最大位置的元素只能是a[m-1][n-1]**

**初始化**

**int a[3][4] = {1,2,3,4,5,6,7,8,9,10,11,12};**

**int a[3][4] = {**

**{1,2,3,4},**

**{5,6,7,8},**

**{9,10,11,12}**

**};**

**操作**

**输出二维数组内容： 数组-4**

**对二维数组排序**

**求每一行的最大值**

**判断矩阵是否对称**

**矩阵的相乘**

**多维数组**

**是否存在多维数组**

**不存在**

**因为内存是线性一维的**

**n维数组可以当做每个元素是n-1维数组的一维数组**

**比如：**

**int a[3][4];**

**该数组是含有三个元素的一维数组**

**只不过每个元素都可以分为四个小元素**

**int a[3][4][5];**

**该数组是含有三个元素的一维数组**

**只不过每个元素都是4行5列的二维数组**

**函数【C语言的第二个重点】**

**为什么需要函数**

**避免了重复性操作**

**有利于程序的模块化**

**什么叫函数**

**逻辑上：能够完成特定功能的独立的代码单元**

**物理上：**

**能够接收数据（当然也可以不接收数据）**

**能够对接收的数据进行处理（当然也可以不处理）**

**能够将数据处理的结果返回（当然也可以不返回任何值）**

**总结： 函数是个工具，它是为了解决大量类似问题而设计的**

**函数可以当做一个黑匣子**

**如何定义函数**

**函数的返回值 函数的名字（函数的形参列表）**

**{**

**函数的执行体**

**}**

**函数定义的本质是详细描述函数之所以能够实现某个特定功能的具体方法**

**return 表达式；的含义**

1. **终止被调函数，向主调函数返回表达式的值**
2. **如果表达式为空，则只终止函数，不向主调函数返回任何值**
3. **break是用来终止循环和switch的，而return用来终止函数**

**例子：**

**viod f()**

**{**

**return;**

**}**

**int f()**

**{**

**return 10;**

**}**

**函数返回值的类型也称为函数的类型，因为如果 函数名前面的返回值类型和 函数执行体中的return 表达式；中表达式的类型不同的话，则最终函数返回值的类型 以函数名前的返回值类型为准 见文件函数-3**

**函数的分类**

**有参函数 和 无参函数**

**有返回值函数 和 无返回值函数**

**库函数 和 用户自定义函数**

**值传递函数 和 地址传递函数**

**普通函数 和 主函数（main函数）**

**一个程序必须有且只有一个主函数**

**主函数可以调用普通函数 普通函数不能调用主函数**

**普通函数可以相互调用**

**主函数是程序的入口，也是程序的出口**

**注意的问题**

**函数调用和定义的顺序**

**如果函数调用写在了函数定义的前面，则必须加函数前置声明**

**函数前置声明：**

1. **告诉编译器即将可能出现的若干个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个字母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，末尾必须加分号**
4. **对库函数的声明是通过 # include <库函数所在的文件的名字.h>来实现的**

**形参和实参**

**个数相同 位置一一对应 数据类型必须相互兼容**

**如何在软件开发中合理的设计函数来解决实际问题**

**一个函数的功能尽量独立，单一**

**多学习，多模仿牛人的代码**

**函数是C语言的基本单位，类是Java，C#，C++的基本单位**

**常用的系统函数 （Turboc 2.0 使用大全）**

**double sqrt（double x）**

**求x的平方根**

**int abs（int x）**

**求x的绝对值**

**double fabs （double x）**

**求x的绝对值**

**专题：**

**递归**

**变量的作用域和存储方式：**

**按作用域分：**

**全局变量**

**在所有函数外部定义的变量叫全局变量**

**全局变量使用范围： 从定义位置到整个程序结束**

**局部变量**

**在一个函数内部定义的变量或者函数的形参 都统称为局部变量**

**void f (int i)**

**{**

**int j = 20;**

**}**

**i 和 j 都属于局部变量**

**局部变量使用范围： 只能在本函数内部使用**

**注意的问题：**

**全局变量和局部变量命名冲突的问题**

**在一个函数内部如果定义的局部变量的名字和全局变量的名字一样时，局部变量会屏蔽掉全局变量**

**按变量的存储方式**

**静态变量**

**自动变量**

**寄存器变量（CPU内部存储）**

**指针：**

**指针的重要性**

**表示一些复杂的数据结构**

**快速的传递数据 减少内存的耗用【重点】**

**使函数返回一个以上的值【重点】**

**能直接访问硬件**

**能否方便的处理字符串**

**是理解面向对象语言中引用的基础**

**总结：指针是C语言的灵魂**

**指针的定义**

**地址**

**内存单元的编号**

**从零开始非负整数**

**范围：8G【0---8G-1】**

**指针**

**指针就是地址，地址就是指针**

**指针变量就是存放内存单元编号的变量**

**或者说指针变量就是存放地址的变量**

**指针和指针变量是两个不同的概念**

**指针的本质就是一个操作受限的非负整数（只能进行相减运算）**

**指针的分类**

1. **基本类型指针**

**程序： 指针-2**

**附注：**

**\*的含义**

**1.乘法**

**2.定义指针变量**

**int \* p**

**3.指针运算符**

**该运算符放在已经定义的指针变量的前面**

**如果p是一个已经定义好的指针变量**

**则 \*p表示以p的内容为地址的变量**

**如何通过被调函数修改主调函数普通变量的值**

1. **实参必须为该普通变量的地址**
2. **形参必须为指针变量**
3. **在被调函数中通过**

**\*形参名 = ……**

**的方式就可以修改主调函数的普通变量的值**

1. **指针和数组**

**指针和一维数组**

**一维数组名**

**一维数组名是个指针常量**

**它存放的是一维数组第一个元素的地址**

**下标和指针的关系**

**如果p是个指针变量，则**

**p[i] 永远等价于 \*(p + i)**

**确定一个一维数组需要几个参数 或者说：**

**如果一个函数要处理一个一维数组，则需要接收该数组的那些信息**

**需要两个参数：**

**数组第一个元素的地址**

**数组的长度**

**指针变量的运算**

**指针变量不能相加，不能相乘，不能相除**

**如果两个指针变量指向的是同一块连续空间中的不同存储单元**

**则这两个指针变量才可以相减**

**一个指针变量到底占几个字节【非重点】**

**预备知识：**

**sizeof(数据类型)**

**功能：返回值就是该数据类型所占的字节数**

**例子：sizeof(int) = 4 sizeof(char) = 1**

**sizeof(double) = 8**

**sizeof(变量名)**

**功能;返回值是该变量所占的字节数**

**假设p指向char类型变量（1个字节）**

**假设q指向int类型变量（4个字节）**

**假设r指向double类型变量（8个字节）**

**p q r本身所占的字节数一样**

**总结：**

**一个指针变量，无论它指向的变量占几个字节**

**该指针变量本身只占四个字节**

**一个变量的地址是用该变量首字节的地址来表示**

**指针和二维数组**

**3. 指针和函数的关系**

**4. 指针和结构体的关系**

**5. 多级指针**

**专题：**

**动态内存分配【重点难点】**

**传统数组的缺点：**

1. **数组长度必须事先制定，且只能是常整数，不能是变量**

**例子：**

**int a[5]; //OK**

**int len = 5; int a[len]; //error**

1. **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分配的空间一直存在，直到该函数运行完毕时，数组空间才会被系统释放**

1. **数组的长度不能在函数运行的过程中动态的扩充或缩小**

**数组的长度一旦定义，其长度就不能再更改**

1. **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函数运行完毕之后，A函数中的数组将无法再被其它函数使用-----传统方式定义的数组不能跨函数使用**

**为什么需要动态分配内存**

**动态数组很好的解决了传统数组的这4个缺陷**

**传统数组也叫静态数组**

**动态内存分配举例-动态数组构造**

**静态内存和动态内存的比较**

**跨函数使用内存的问题**

**结构体**

**为什么需要结构体**

**为了表示一些复杂的事物，普通的一些基本类型无法满足实际需求**

**什么叫结构体**

**把一些基本类型数据组合到一起形成一个新的复合数据类型**

**如何定义一个结构体**

**3种方式，推荐使用第一种：**

**//第一种**

**struct**

**{**

**int age;**

**float score;**

**char sex;**

**};**

**怎样使用结构体变量**

**赋值和初始化**

**定义的同时可以整体赋初值**

**如果定义完之后，则只能单个赋初值**

**如何取出结构体变量中的每一个成员【重点】**

1. **结构体变量名.成员名**
2. **指针变量名->成员名**

**指针变量名->成员名 在计算机内部会被转化成 (\*指针变量名) .成员名的方式来执行**

**所以这两种方式是等价的**

pst->age 在计算机内部会被转化成 (\* pst).age

**pst->age 等价于 (\*pst).age 等价于 st.age**

**我们之所以知道 pst->age等价于st.age，是因为pst->age是被转化为了(\*pst).age来执行**

**pst->age 的含义：**

**pst所指向的那个结构体变量中的age这个成员**

**结构体变量和结构体变量指针作为函数参数传递的问题**

**推荐使用结构体指针变量作为函数参数来传递**

**结构体变量的运算**

**结构体变量不能相加，不能相减，也不能相互乘除**

**但结构体变量可以相互赋值**

**举例：**

**动态构造存放学生信息的结构体数据**

**动态构造一个数组，存放学生的信息**

**然后按分数排序输出**

**枚举**

**什么是枚举**

**把一个事物所有可能的取值一一列出来**

**怎么使用枚举**

**枚举的优缺点**

**代码更加安全**

**书写麻烦**

**位运算符：**

**& --- 按位于**

**&& 逻辑与 也叫并且**

**&& 与 & 的含义完全不同**

**1 & 1 = 1;**

**1 & 0 = 0;**

**0 & 1 = 0;**

**0 & 0 = 0;**

**| --- 按位或**

**1 | 1 = 1;**

**1 | 0 = 1;**

**0 | 1 = 1;**

**0 | 0 = 0;**

**~ --- 按位取反**

**~i就是把i变量所以的二进制位取反**

**^ --- 按位异或**

**相同为零**

**不同为1**

**1 ^ 1 = 0;**

**1 ^ 0 = 1;**

**0 ^ 1 = 1;**

**0 ^ 0 = 0;**

**<< --- 按位左移**

**i << 1 表示把i的所有二进制位左移1位**

**左移n位相当于乘以2的n次方**

**>> --- 按位右移**

**i >> 3 表示把i的所有二进制位右移3位，左边一般补0，也可以补1**

**右移n位相当于除以2的n次方，前提数据不能丢失**

**位运算的现实意义**

**通过位运算符我们可以对数据的操作精确到每一位**

**专题：**

**补码：**

**原码**

**也叫 符号-绝对值码**

**最高位0表示正， 1表示负。其余二进制位则是该数字的绝对值 的二进制位**

**原码简单易懂**

**加减运算复杂**

**存在加减乘除四种运算，增加了CPU的复杂度**

**零的表示不一【计算机没有应用】**

**反码**

**反码运算不便，也没有在计算机中运用起来**

**移码**

**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，直至商为零，余数倒叙排列**

**负整数转二进制**

**先求与该负数相对应的正整数的二进制代码**

**然后将所有位取反，末尾加一，不够位数是，左边补一**

**零转二进制**

**全是零**

**二进制转十进制**

**如果首位是0，则表明是正整数，按普通方法求**

**如果首位是1，则表明是负整数**

**将所有位取反，末尾加1，所得数字就是该负数的绝对值**

**如果全是零，则对应的十进制数字就是零**

**学习目标：**

**在VS 2017中一个int类型的变量所能存放的数字范围是多少**

**int类型变量所能存储的最大整数用十六进制最大的整数是： 0X7FFFFFFF**

**int类型变量所能存储的绝对值最大的负整数用十六进制最大的整数是：0X800000000**

**最小的负数的二进制代码是多少**

**最大的正数的二进制代码是多少**

**已知一个整数的二进制代码求出原始的数字**

**数字超过最大的正数会怎样**

**不同类型数据的相互转化**

**进制转化**

**字符串的处理**

**链表**

**算法：**

**通俗定义：**

**解题的方法和步骤**

**狭义定义：**

**对存储数据的操作**

**对不同的存储结构，要完成某一个功能所执行的操作是不一样的**

**比如：**

**要输出数组中所有的元素的操作和**

**要输出链表中所有元素的操作肯定是不一样的**

**这说明：**

**算法是依附于存储结构的**

**不同的存储结构，所执行的存储结构是不一样的**

**广义定义**

**广义的算法也叫泛型**

**无论数据是如何存储的，对该数据的操作都是一样的**

**我们至少可以通过两种结构来存储数据**

**数组**

**优点：**

**存取速度快**

**缺点：**

**需要一个连续的很大的内存**

**插入和删除元素的效率很低**

**链表**

**专业术语：**

**头结点**

**头结点的数据类型和首节点的数据类型是一模一样的**

**头结点是首节点前面的那个节点**

**头结点并不存放有效数据**

**设置头结点的目的是为了方便对链表的操作**

**头指针**

**存放头结点地址的指针变量**

**确定一个链表需要一个参数---头指针**

**首节点**

**存放第一个有效数据的节点**

**尾节点**

**存放最后一个有效数据的节点**

**优点：**

**插入删除元素效率高**

**不需要一个很大的内存**

**缺点：**

**查找某个位置的元素效率很低**

**二进制全部为零的含义 ------ 0000000000000000000000000000 的含义**

1. **数值零**
2. **字符串结束标记符 ’\0’**
3. **空指针 NULL**

**NULL本质也是零，而这个零不代表数字零，而表示内存单元的编号零**

**我们计算机规定了，以零位编号的存储单元的内容不可读，不可写**