C语言32个关键词

关键字由系统定义，不可以重新作为其他的定义

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **auto** | **break** | **case** | **char** | **const** | **continue** | **default** | **do** |
| double | else | enum | extern | float | for | goto | if |
| int | long | register | return | short | signed | sizeof | static |
| struct | switch | typedef | unsigned | union | void | volatile | while |

C语言的限定符

|  |  |
| --- | --- |
| 限定符 | 含义 |
| extern | 声明一个变量，extern声明的变量没有建立存储空间。  extern int a; 声明一个变量，并不分配空间。 |
| const | 定义一个常量，常量的值不能修改。 |
| volatile | 防止编译器优化代码  编译器会对一些代码进行优化，大部分情况下是好的。然而和硬件/驱动/内核相关的时候可能负优化 volatile int sum； 防止对该变量进行优化 |
| register | 定义寄存器变量，提高效率。register是建议型的指令，而不是命令型的指令，如果CPU有空闲寄存器，那么register就生效，如果没有空闲寄存器，那么register无效。 有一些变量经常性的使用，如果有寄存器空余，就定义在寄存器中 |

extern int a 表示声明一个变量。但深层次含义其实是标识这个变量由外部引入，在其他文件已经进行定义了，这个文件不需要定义了。因此有了extern 之后就不需要再定义一次。

typedef 的应用

typedef 一般有3种应用：

1. 简化关键字，可以缩短关键字的长度
2. 区分数据类型
3. 增加可移植性，简便

简化关键字：可以将变量缩短

|  |
| --- |
| typedef unsigned int u32; // u32 a1 == unsigned int a1;  struct PERSON{};  typedef struct PERSON Person;//typedef单独写出来  // 写在一起  typedef struct PERSON{}Person;  // 没用typedef 之前创建变量要写 struct PERSON p1;  用了之后：Person p1; |

区分数据类型

|  |
| --- |
| typedef char \* charporter;  int main(){  char \*p1, p2; //指针是跟着变量的，不是跟着类型的,也就是说,这个定义的是char\* p1 和 char p2;  charporter p1, p2// 这下这两个都是char\* 类型了 |

可移植性：

很多个同类型修改. 下一次，当不想使用long long了，就直接再上面改，简便运算

typedef long long longer;

int main(){

    longer a2;

}

sizeof

**sizeof 是运算符，不是函数**，它是**编译器**直接提供的，不需要包含任何头文件。

功能：计算一个数据的类型所占据空间大小， 单位为**字节**。

返回值：size\_t ，它是个unsigned int

变量长度

C语言和C++并没有规定各种数据类型随占据储存单元的大小，这些由编译器自行决定但必须满足条件

一般来说，char (1B) short (2B) long win4B linux8B long long 8B float 4B double 8B

C语言的数据类型

一、数据类型的理解

数据类型是**固定大小内存的别名**，它是对内存的格式化操作，是创建变量的模具，是对数据的抽象。

* 对数据的抽象：所有的浮点型都抽象成float或者double类型，所有的整型都抽象成int类型
* 相同的数据类型一定具有相同的表现形式，存储格式，操作方式

C语言是强类型语言：C语言每个数据都必须对应内存中的一块空间，因此**所有的数据都必须有数据类型**。

* 弱类型：弱类型语言中没有类型的概念，所有变量全都是一个类型（一般都是字符串的），程序在用的时候再根据需要来处理变量
* 强类型语言中所有的变量都有自己固定的类型，这个类型有固定的内存占用，有固定的解析方法

比如 int a = 10; 其含义不仅仅是定义了一个int变量。还意味着在内存中为这个变量分配了int大小的空间，并且对这个变量操作可以使用这个变量名操作

数据类型作用：**告诉编译器需要为对象（变量）分配的内存空间大小**。

数据类型有：基本类型，构造类型，指针类型

1. 基本类型：整型、字符型、实型（浮点型）

* 整型：char、 short、 int、 long、 long long char是最小的整型单位
* 字符型：char
* 实型（浮点型）：单精度（float）、双精度（double）
  + 浮点型在系统中存储遵循的是IEEE754标准：float: 精度为6位有效数字 double 15位有效数字

1. 构造类型：数组类型，结构体类型（struct），联合类型（union），枚举类型（enum）

* 枚举类型的每一项在内存中也是以整数形式存储的

1. 指针类型：char\* int\*

注意**，C++和java**有专门的**bool类型**，**C语言没有bool**类型，整型数字1和0代表true和false

一、C语言的数据

C语言的数据有不同的数据类型。数据还可以分成两种表现形式：**常量**和**变量**

* 常量：程序运行中值不可以被改变的量
* 变量：程序运行中值可以被改变的量

**（一）常量**

程序运行中值不可以被改变的量。常量可以被赋值给变量

常量的值**不一定被初始化**，但其实不初始化系统也会给它一个随机值，但是没有实际意义了。

常量定义过后，无论初始化与否，它的值都不能被改变了。

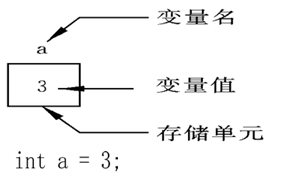
常量可以被分为：整型，字符型，字符串常量，浮点型（实型），符号常量

1. 整型：100 200 300
2. 实型常量：**小数形式 123.12312**  **科学计数法/指数形式**

* 科学计数法中E的大小写都可以，但是E**前后都必须有数字**，而且**都必须为整数**

1. 字符型：C语言字符型是char类型，单引号引起来。，，. C语言中字符都是以ASCII码的形式的整型储存的，但只能存0-128的数字。C语言char类型完全可以当成整数类型
2. 字符串：C语言字符串是是char类型数组实现的，双引号引起来，其后面默认存储.
3. 符号常量：**#define PI 3.1415926** 在预处理的时候将所有**宏**替换为后面的表达式，**没有分号**

**（二）变量**



变量：程序运行中值可以被改变的量。

如果想要获取变量的值或者操作变量，可以使用变量的地址，或者直接使用它的变量名进行操作。

变量**必须先定义再使用**。定义的时候要规定数据的**数据类型**。

定义，声明和初始化：

* 声明：告诉编译器变量的存在。变量不建立存储空间，如：extern int a; 只声明没定义
* 定义：声明 + 为变量在内存中开辟存储空间，如：int b; int b 既声明又定义
  + int b; 定义了一个int类型的变量，它占据4个字节的存储空间
  + **定义的模式：数据类型 + 变量名；**
* 初始化：在定义的时候就进行了赋值
  + int b=4; 定义的时候进行了赋值操作

**标识符（变量名，方法名，类名，自建类型名）的要求：**

1. 标识符不能是关键字
2. 变量名由字母数字，下划线组成。第一个字符必须是字母或者下划线，不能是数字
3. 区分大小写

**（三）符号常量和常变量 都是常量**

常变量：具有变量基本的属性：类型，占据储存单元，但是在运行期间它的值不能被改变。常变量最好在定义的初始化，不然系统会为自动分配一个随机数，这就失去了它的意义。

然而符号常量在预编译时候会被替换，在替换之后符号就不存在了，**不会为符号常量的名字分配储存空间。**下面的Pi不是变量，内存也不会为Pi分配空间，只会进行替换的操作。

|  |
| --- |
| #define Pi 3.1415926; //定义符号常量，也就是定义了一个宏  const int b = 0123; //定义常变量并初始化  const int c; //定义常变量，但是不赋值，编译器会自动给他赋个随机数  printf(“%d”, c); //打印这个常量 |

Const 定义的变量并不是一定不可以进行修改的，只是原则上不，使用指针进行修改还是可以的

|  |
| --- |
| const int b = 10; //定义了一个常量，其值为10  // b = 11; 直接修改，错误  int p\* = &a;  \*a = 15 // 使用指针是可以进行改变的 |

C语言的进制

C语言中**不可以书写二进制**，可以书写十进制，八进制和十六进制。

* C语言中八进制以0开头，十进制以0x / 0X开头都是可以的
* 八进制范围为0-7，十六进制范围为0-9，A-F
* 十进制输出%d, 八进制%o，十六进制%x

|  |
| --- |
| //全局变量  int a = 123; //以十进制的方式赋值  int b = 0123; //八进制方式赋值，以数字0开头  int c = 0xABC; //十六进制方式赋值，以0x或者0X开头 |

在输出的时候，如果想把八进制和十六进制弄出来，在百分号和字母间加#，也可以控制x大小写

|  |
| --- |
| printf("%d",a); // 结果：123  printf("%o",b); // 结果：123  printf("%x",c); // 结果：abc  printf("%X",c); // 结果：ABC  printf("%#o",b); // 结果：0123  printf("%#X",c); // 结果：0XABC |

进制之间也是可以**相互打印转换**的。因为数据在内存中都是以**补码**的形式存在的，储存形式都是一样的，只是输出形式发生了变化

|  |
| --- |
| int a = 123; // 以十进制方式赋值  prinf("%d",a); // 以十进制输出  printf("%o",a); //以八进制输出  printf("%x",a); //以十六进制输出 |

**（一）原码，反码，补码**

**计算机中的数值一律采用补码来储存，不管正数还是负数**

原码，反码，补码是对于**带符号整数储存**方式。**无符号数，没有原码，反码，补码。**

原码：数用二进制数来表示。

* 最高位为符号位，用符号位区分正负，0表示正数，1表示负数
* 问题：
  + 0的状态有两种 +0 0000 -0 1000
  + 不同符号的两个数相加会有问题，两个正数相减也有问题，负数相减没问题

反码

1. 正数的反码和原码相同,
2. 负数符号位不变, 其他位取反, 1 变0 0 变 1
3. 计算结果正确, 但是还是有2 个 0 的状态

补码

计算机中的数值一律采用补码来储存，不管正数还是负数

1. 正数 **的反码 原码 补码** 都是一样的
2. 负数的补码 等于反码 + 1（符号位不动, 其他位求反, 然后整个加1）
3. 已知补码, 符号位不动, 其他位求反, 然后整个加1, 得到原码
4. 计算结果正确, 0的状态只有一种

**整型**

**（二）有符号数和无符号数**

**只有整型分有符号数和无符号数**。浮点型和整型存储方式是不同的。

有符号数：可以表示正数或者负数。最高位为符号位，1代表正数，0代表负数。默认：有符号数

无符号数：只能表示正数。所有的位都是数据位

有符号数和无符号数能表达的数的个数是相同的，但是范围不同

|  |
| --- |
| int a = 123; // 默认有符号数  signed int a; // 定义有符号数  unsigned int a //定义无符号数 |

表示范围：

以char为例，char 1B有8-bit。如果是有符号，最高位是符号位，然后其他7位是数据位。

当最高位为0，也就是正数：表示范围是0 000000 – 0 111111。注意符号那位代表。因此正数范围是. 负数范围是1 000000 – 1 111111 计算机规定1 0000000 表示 , 因此这个范围就是

综合起来，char表示的范围就是

而无符号数全是数据位，最大数是8个1因此就是

|  |  |  |  |
| --- | --- | --- | --- |
| 种类 | 占据空间 | 有符号 | 无符号 |
| char | 1B |  |  |
| short | 2B |  |  |
| int | 4B |  |  |
| long | 4B |  |  |

定义无符号整型后，不能赋予一个负数，否则会错误

|  |
| --- |
| unsigned int btr = -1; // 定义有符号数  printf(, btr); //65535  printf(, btr); //65535 |

**进制和储存的转换细节**

C语言在进行变量赋值的时候：

* 十进制赋值：给的是原码
* 八进制或者十六进制：给的是补码

C语言在进行格式输出的时候：

* 十进制输出：要的是原码
* 八进制或者十六进制输出：要的的是补码

前文提到过，数据类型的作用只是声明以及开辟空间，其实什么类型的数并不影响这个数在内存中的存储。有的时候类型和赋值不符可能会导致类型转换，或者长度转换。比如

Int a = 3.14; 这时候会发生四舍五入的情况

short a = 10; 会发生int转short

但是当转换后，从数据到储存其实和类型关系就不大了，int无论是short，unsigned还是别的，数据储存的形式和类型是无关的。类型只是在使用的时候效果才显现出来

比如 unsigned int ch = -1; unsigned int 确实只表示正数，但并不影响其存储，-1仍旧还是以有符号数补码形式储存的 也就是1111.

但是在输出的时候是结合数据类型和格式输出类型一起来看的，对已有储存的反码进行处理

**（一）溢出和赋值不正常分析**

1. 无符号的赋值不当

|  |
| --- |
| unsigned short btr = -1; // 定义有符号数  printf(, btr); //  printf(, btr); |

首先，-1是个负数赋给正数不需要类型转换，在内存中以补码存在，为16个1。

无论是以%d，还是%u，因为类型是unsigned， 因此无论是%d还是%u，都会当成一个正数，来看待，就是

字符型：char 类型

char 既是字符型又是最小整型 (char 的本质就是一个字节大小的整型)

字符型变量赋值需要单引号括起来 'A' '\n' **在C/C++中字符就是整数**

char 类型既是可以存字符又是可以存数字的， 但本质上在内存中存的是字符对应的ASCII码。

ASCII码代表的字符有128个：

* 注意'\0'和数字0是等价的，和字符'0'是不同的. '\0'的ASCII值是0，字符'0'的ASCII为48
* ASCII ‘\0’-0 ‘0’-48 ‘A-Z’-65~90 ‘a-z’-97~122 空格字符 32

注意字符'1'和整数1是完全不同的两个概念.字符占据一个字节，整数占据四个字节，两者含义完全不同

转义字符也是有ASCII码的，算一个字符

|  |
| --- |
| char ch = 'a';  printf("%d",ch); // 输出值97  char ch = '\n';  printf("%d",ch2); //输出值10 |

单引号和数字本身就是通用的

|  |
| --- |
| char ch = 'a';  char ch = '97';  printf("%d",ch); // 输出值97  printf("%c",ch); // a |

**字符的加减法本质上就是ASCII码值的整数的加减法**

大写转小写加32，小写转大写减32，**32代表着空格字符**。

|  |
| --- |
| char ch = 'a'; //ASCII: 97  printf("A = %c\n", ch - 32); //小写a转大写A  char ch2 = 'A'; //65  printf("a = %c\n", ch + 32); //大写A转小写a  printf("A = %c\n", ch - ' '); //小写a转大写A---里面有个空格  printf("A = %c\n", ch2 + ' '); //小写a转大写A---里面有个空格 |

**字符转数字值变小，减48，数字转字符加48，字符'0' 就是48**

|  |
| --- |
| char ch = '8';  int a = ch – ‘0’; |

正因为字符在内存中是以ASCII码储存的，会出现以下的情况

|  |
| --- |
| char ch = 'a';  printf("%d\n",sizeof(ch))//1  printf("%d\n",sizeof('a'))//4 |

因为char类型占据的是1格字节，但是如果直接使用字符相当于操作ASCII，也就是sizeof(97)。数字默认为int，因此占据4B的空间

