目 录

[1 基本概念 1](#_Toc53090271)

[2 类型、运算符和表达式（位运算比较有趣） 1](#_Toc53090272)

[3 控制流 2](#_Toc53090273)

[4 指针与数组 3](#_Toc53090274)

[5 结构 5](#_Toc53090275)

# 基本概念

1. c语言除了静态定义和栈自行分配以外没有定义其他的存储分配方式，没有堆或者垃圾回收。c++的指针可以用new的操作，分的内存就在堆上

void func()

{

int a;

int \* p=new int (10);

int b;

delete p;

}

1. c语言中所有的变量都必须在用之前声明，通常在函数开始处，在所有可执行语句之前
2. 用两个单引号‘’括起来的字符代表一个整型值，这个整型值与这个字符在机器字符集(machine’s character set)中对应的数值相同，这就是字符常量,其实就是比较小的整数的另一种写法。比如’c’代表99
3. 用&&和||连接的表达式从左到右求值，并保证在在求值过程中，只要能判断最终结果的真假，求值就会终止. if(a||b)
4. c语言中所有函数参数都是通过值传递。(c++里面有引用传递&)就是说被调用的函数的参数的值是通过临时变量的方式得到的，而不是主调函数中的原本的变量。则c语言中被调用函数不能直接更改主调函数中的值，它只能改变它私有的、临时的拷贝版本

void func1(int a) ; void func2(int &a)

int b=5;

func1(b);

func2(b);

1. 整型常量前面带0代表八进制，前面带0x或0X代表十六进制

# 类型、运算符和表达式（位运算比较有趣）

1. 外部变量与static默认情况初始化为0
2. atoi：数字字符串转换为数字
3. 隐式算数转换会进行低转高，但float不会自动转为double。

int a=1；

float b=a；(隐式转换) float b=(float)a;（显式转换）

1. (double)a 强制类型转换。若函数形参为float，也会对传入的int数进行强制类型转换。会丢失信息的转换可能会有警告但完全合法。
2. &用于屏蔽某些位，flags&0x111（0000 0100 1001 & 0000 0000 0111）可以屏蔽除最低三位外的数字；|可以置位，flags|1可以即最低位置1；~（求反码，1变0,0变1）可以清除所需的位，比如flags&~1111（0000 1111）（一元运算符优先级高于二元运算符）（问：为不写0000,要写~1111，因为flags不一定只有4位）可以清除1234位数字，变为0。

eg:getbits(unsigned int x,ingt p,int n)//将x从p位置开始的n位取出对齐

{return (x>>(p-n+1)) &~(~0<<n);}前面一部是是将目标位移到右边，后一部分是将原数前面的所有数置0

x&=(x-1)可以把x的最右边的一个1去掉，0x11100-1=0x11011,0x11100&0x11011=0x11000即将最后一位去掉

改写：

int bitcounts(unsigned x)//统计其中1的数量

{

int b=0;

while(x!=0)

{

x&=(x-1);

++b;

}

}

1. unsigned修饰保证右移时高位空位补0。（> a>>1 1000 1100>>1 :0100 0110）
2. **c语言没有明确定义同一操作符的几个运算分量的求值顺序（除了&&，||，？：，’,’这几个操作符）。例如x = f() + g();f和g谁先求不能确定下来。同样的，函数参数中的求值顺序也是不确定的**

**(a>b)?a:b**

# 控制流

1. if的嵌套尽量用{}
2. ,表达式，for循环中经常使用 for(i=0，j=1;j<5;++j,++i)
3. 分隔函数参数，变量声明等等的逗号不是逗号操作符，不保证从左到右求值。

Func(int a,int b) int a,b;

1. goto语句转向某个位置，对于退出深层嵌套很有用

for(;;)

for(;;)

{if() goto found;}//退出两层for循环

found:

{}

1. 不允许函数之内定义函数,可以调用其他函数
2. extern double val[] 只是声明一个外部数组，但是并不分配内存，所以不需要说明数组规模；double val[12]
3. 外部static常用于说明变量和说明函数，静态函数只能用在函数说明所在的文件之内。内部static变量只属于特定函数，不管函数是否调用都存在

Func（）

{

Static int a=1；

Printf(a);

++a；

}

For（int i=0;i<3;++i）

Func();

1. 寄存器变量register提醒编译程序所在变量所在频率高，则将其存在寄存器里面，访问更快。只适用于自动变量以及函数的形参。不论寄存器变量是不是存在寄存器里面，其地址都不能访问
2. 初始化：
   1. 没有显示初始化时，外部变量和static初始化为0，自动变量和register没初始化（初值是“垃圾”）。
   2. 外部变量和register的初始化符都必须是常量表达式，初始化只做一次(?)
   3. 自动变量和register初始化符可为常量表达式或者函数调用等任意表达式(?)
   4. int a[]={1,2,3}

int b[3]={1}其他为0

char pa[]=”he” char pa[]={‘h’,’e’,’\0’}结束符占一位

1. #define 名字 替换文本

替换文本可以为多行，但是每行末尾加\

变元做多次操作得小心： #define square(x) x\*x（×） 调用square(z+1)出问题

1. #undef 标识符 后面取消以前定义的宏定义，后面则不能调用这个宏，如果标识符未被定义为宏，则忽略此语句
2. 宏定义的形式参数不能用带引号的字符串替换。但是，如果在替换文本中以#为前缀，那么其将被由实际边缘替换的参数扩展成待引号的字符串。例如：

#define dprint(expr) printf(#expr “=%g\n”,expr)（宏定义的形参不能为字符串）

调用dprint(x/y)，其会被扩展printf(“x/y””=%g\n”,x/y),即为printf(“x/y=%g\n”,x/y)

3/2.0=1.5

1. 条件包含：

在预处理阶段，可以进行条件控制。例如下面代码可以根据电脑版本包含不同的头文件

#if system==sysv

#define hdr “sysv.h”

#elif sysytem==bsd

#define hdr “bsd.h”

#else

#define hdr “default.h”

#endif

#include hdr

需要测试一个文件是否已经定义

#ifndef hdr

#define hdr

#endif

# 指针与数组

1. 取址&只用于内存中的对象，不能操作表达式(x+y)、常量(5)、寄存器变量
2. 指针运算比数组下标运算更快

int a[10];

int \*pa;

pa=a;//pa=&a[0]

\*(pa+1);//a[1],\*(a+1)

但是数组和指针还是有区别，指针是变量，所以pa=a和pa++合法，但是数组名称不是变量，所以a=pa，a++非法

1. 所有的指针运算都会自动考虑它所指向的对象的大小

有效指针运算：同类型指针之间赋值；指针值加减整数；指向同组数组元素的指针间的减或者比较运算；指针赋0或与0做比较。其他都非法

int a[10];

int \*p=a,\*q=a+6;

q-p+1表示位于pq之间的元素个数

1. 字符指针与数组

char amessage[]=”hello”;

char \*pa=amessage;//pa指向amessage的首地址，并没有复制字符串 strcpy

1. 指针数组与指向指针的指针

char \*pa[10];//是一个有10个元素（假如为文本）的一维数组，且数组中每个元素都是指向char型字符的指针。如果我们要交换实际的文本行的时候，只需要交换指针里面的值即可，而不必实际去移动真正的文本。

指针数组的每行可以有不同的长度，比如字符串指针数组，每一个指向的字符串长度都可以不一样。

1. 二维数组：

int a[2][3];

f(int a[2][3]);//f(int a[][3];f((\*a)[13])

必须要括号，否则\*a[13]表示有13个指针的指针数组，因为[]优先级比\*高。函数传参的列数必须指明，行数可以不指明

1. 二维指针和指针数组：二维数组真真正正的拥有那么多可以操作的存储空间，而指针数组，在指针未被赋值之前只有存储指针的空间。指针数组的好处是列可以不同长
2. 指向函数的指针

任何类型的指针都可以转换为void\*，并且当它转换回原来的类型时不会丢失信息。

char \*pa[]; void \*pa[]

int (\*comp)(void\*,void\*)；表示comp是一个指向函数的指针，\*comp是一个函数，参数为两个void\*；这个括号很重要，否则变成返回一个指向int类型的指针的函数

调用：(\*comp)(v[i],v[j])

1. int (\*daytab)[13];//指向由13个整数类型元素组成的一维数组的指针

int \*daytab[13] 由13个指向整数类型对象的指针组成的一维数组

char (\*(\*x())[])()

x:返回值为指向一维数组的指针的函数，该一维数组由指向返回字符类型

char (\*(\*x[3])())[5]

x:由3个指向函数的指针组成的一维数组，该函数返回指向由5个字符组成的一维数组的指针、

# 结构

1. struct point{

int x;

int y;

};

定义对象：struct point p1{1,1};（struct必须写，跟c++不一样）

struct point \*pa;

1. 结构体成员运算符(. ->)、函数调用的()以及下标[]优先级最高。
2. 结构体中包含指向自身类型的指针合法，包含自身类型的实例非法

eg: struct tnode

{

struct tnode \*left;//合法

struct tnode right;//非法

}

1. typedef建立新的数据类型名字，以大写字母开头，以示区别。未增加新内容，类似于#define
2. 联合：使单一变量可以在不同时刻保存不同类型和大小的对象的变量

union u\_tag{

int ival;

float fval;

char \*sval;

} u；

变量需要大到可以存储三种类型中的最大的一个，具体大小依赖于实现。使用中里面任意类型的值都可以赋值给联合变量，只要满足一致性：读取的类型必须是最近一次存入的类型

成员访问：联合名.成员；联合名->成员

一个联合只能用它第一个成员类型的值来初始化，比如上述u只能用int型变量来初始化

1. 位字段（bit-field）：是一个由具有特定数量的位组成的整数变量。结构或联合的成员也可以是位字段。如果连续声明多个小的位字段，编译器会将它们合并成一个机器字（word）。这使得小单元信息具有更加紧凑的存储方式。当然，也可以使用位运算符来独立处理特定位，但是位字段允许我们利用名称来处理位，类似于结构或联合的成员。

声明格式： 类型 [成员名称]：宽度；

类型：指定一个整数类型，决定该字段被解释的方式，可以是\_bool/int/unsigned int/signed int

成员名称：可以不写，但是无名位字段无法获取，只能用于填充

宽度：小于等于指定类型的位宽，无名字段宽可以为0，这时下一个声明的位字段就会从新的可寻内存单元开始

eg：

struct date {

unsigned int month : 4; // 1是1月；12是12月(1111:15四位可以表示0~15)

unsigned int day : 5; // 月份中的日（1~31）

signed int year : 22; // （-2097152~+2097151）

\_bool isdst : 1; // 如果是夏令时，则为true

};

struct date birthday = { 5, 17, 1982 };则birthday占据的存储空间大小与一个32位的int整数对象一样。

Birthday.year

和结构中其他成员所不同的是，位字段通常不会占据可寻址的内存位置，因此无法对位字段采用地址运算符（&）或宏 offsetof。