
C尖琐碎笔记整理

程序填空题注意

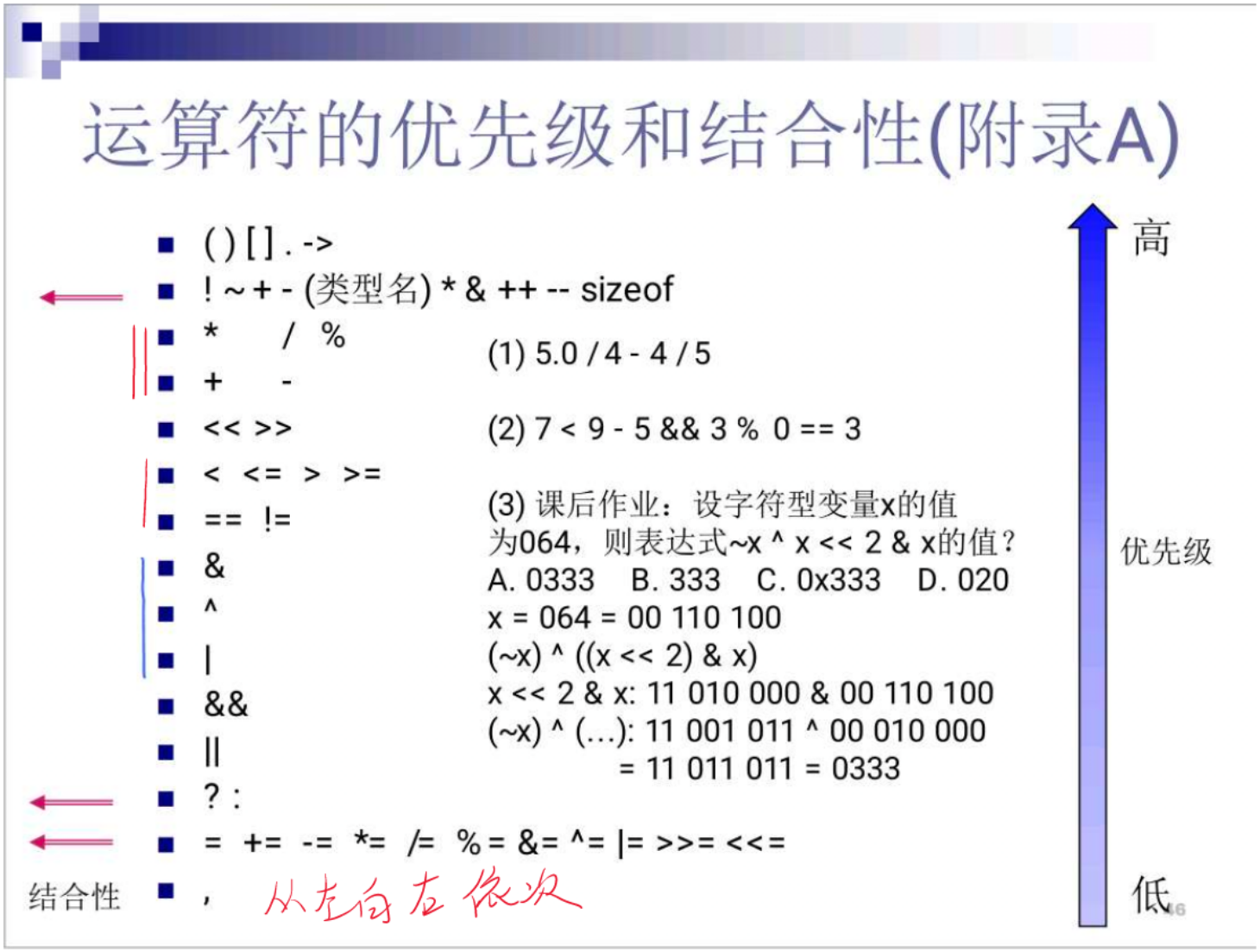
中英文括号 `()，；`
结尾的`；`
函数名别抄错
注意进制 `010==8`

1~7章琐碎

- 自定义函数时，数组名作为形参`fun(int a[]);`编译时自动转化为指针`fun(int *a);`
- 函数的定义不能嵌套, 即函数的定义不可完整地包含另一个函数的定义
- `sizeof()`为内存空间,包括`'\0'`; `strlen()`为有效长度,读至`'\0'`结束.上述两函数均以字节为单位
- 指针`p+1`等效于地址数值上`p+sizeof(*p)`
- 指针和数组名:数组名是指针常量, 不可更改或赋值, 指针可`p++`,但数组名不可`a++`
- 字符串常量实质上是一个指向字符串首字符的**指针常量**
- 形参与实参:一一对应, 数量相同, 类型尽量一致
  - 形参:必须是变量
  - 实参:变量, 常量或表达式
- 存储区
  - 系统存储区
  - 用户存储区

    - 程序区:代码
    - 数据区:
        - 静态存储区:静态变量
        - 动态存储区:自动变量(局部变量)
- 自动变量未赋初值:随机值;静态变量未赋初值:自动初始化为0
- 全局变量的初始化在进入`main()`函数之前;
- `int k=11;printf("%d,%o,%x",k,k,k);`输出`11,13,b`而非`11,013,0xb`(不输出前导符)
- 溢出:运算结果超出本身的表示范围, 有符号整数和无符号整数均会溢出
  `while(i>0){i++;}`不会死循环
- 整型变量和字符型变量的定义和值都可以互换(互换时, 整型数据的取值范围是有效的ASCII码)
- 字符常量:
    - `'a'`
    - 数字:65
    - 转义字符:`'\n'`, `'\ddd'`(八进制), `'\xdd'`(十六进制)**不得超出255**
- `%m.nf`输出格式符, 输出宽度至少为m (包括.) 的保留n位小数的数
- 类型转换
    - 自动类型转换
        - 非赋值运算:保证精度不降低
        - 赋值运算:右侧类型转化为左侧
    - 强制类型转换:临时改变
      `int x=0;double y=1.0;y+=(double)x;`此时`x`仍为`int`型
- 表达式
    - 自增运算符`++`和自减运算符`--`的运算对象只能是变量,不能是常量或者表达式
    - 赋值运算符左边必须是一个变量
    - 逻辑运算符:
        - 优先级:`!` > `&&` > `||`
        - 逻辑短路:求用`&&`或`||`连接的逻辑表达式时,按从左到右顺序计算, 一旦能得到结果立即停止计算
          `int x=5;int y=0&&(x+=1);int z=1||(x+=1);`
          运行结束后`x==5`
    - 逗号表达式:`(),(),();`;结果为最后一项
    - 条件表达式:三目运算符`()?():()`
    - 位运算符:操作数只能是整型或字符型数据及其变体
        - 位逻辑:`^`, `&`, `|`, `~`
            - `a^=b^=a^=b;`等效于`int temp=a;a=b;b=a;`
            - `~a==-(a+1)`
        - 位移:`>>`, `<<`

        - 位移运算不改变原值, 需另赋值: `a=a>>2;`
- 数组初始化:
    - 不初始化:静态数组自动初始化为0;动态数组为随机值
    - 部分初始化:其余元素为0
    - 初始化时, 若对全部元素赋了初值, 则可省略数组长度
- 二维数组赋初值
    - 分行赋初值: `int a[3][3]={{1,2,3},{1,2}{1,,3}};`
    - 顺序赋初值: `int a[3][3]={1,2,3,4,5,6,7,8,9}`
    - 省略行长度: `int a[][3]={1,2,3,4,5,6}`
- 字符串初始化:
    i. `char str[6]="hello";`
    ii. `char str[6]={"hello"};`
    iii. `char str[6]={'h','e','l','l','o'};`
- 内存
    - 栈区:存放参数值，局部变量等，编译器负责自动分配和释放
    - 堆区:通过`malloc`, `realloc`, `calloc`分配的内存块,编译器不负责释放, 内存泄漏发生在此处
    - 静态区:全局变量和静态变量放在此处, 程序结束后, 由系统释放
    - 常量区:常量存储在此(如字符串常量)
    - 代码区:代码存储在此
- `malloc(size)`:返回指针
  `if((int*p=(int*)malloc(n*sizeof(int))==NULL){printf();exit(1);}`
- `calloc(n,sizeof())`:返回指针,初始化为0
- `realloc(void* ptr,size)`
    - `ptr`必须指向`malloc`或`calloc`内存分配得到的内存
    - `size`为现在所需的空间大小
    - 失败返回`NULL`,原空间不变;成功返回**新指针**(**不一定为原地址**)
      一般`if((s1=(Point*)realloc(s,20*sizeof(Point)))!=NULL){s=s1;}`而非`s=(Point*)realloc(...)`
- `free(p)`:p只能是首地址, 不能释放p+10(部分内存)



指针, 结构

- 相同类型的指针 `p` , `q` ,可进行赋值(`p=q`), 比较(`p!=q`)和算数运算(`p-q` , `p+1`),但不能 `p+q`
- `put(s)` 等效于 `printf("%s\n",s)` ,结尾输出换行
- `char *p="hello";strcpy(p,"world");` 报错,段错误
- `void` 是一种基本数据类型,可用 `void*` 定义指针变量,但不能用 `void` 定义变量
- 在定义嵌套的结构类型时, 必须先定义成员的结构类型, 再定义主结构类型
- `(*p).num` 等效于 `p->num`
  - `*p.num` ,错, 注意优先级
  - `(*p)->num` ,错, 混用
- C语言中结构类型变量在程序执行期间所有成员一直驻留在内存中
- 指针变量自身所占的内存空间大小和它所指的变量数据类型无关
- 指针名是变量, 所代表的地址可以改变
- 数组名是(指针)常量, 所代表的地址不能改变

- `printf("%s","string"+1)` ⇒tring
- 改变数组 `sa` 所代表的字符串,只能改变数组元素的内容
  `strcpy(sa,"hello_world")`
  `scanf("%s",sa)`
  ~~`sa="hello_world"`~~
- 改变指针 `sp` 所代表字符串,通常直接改变指针的值, 让它指向新的字符串
  `sp="hello_world"`
  ~~`scanf("%s",sp)`~~
- C语言的**全局变量, 静态局部变量**的存储是在**编译时**确定的, 其**存储空间的实际分配**是在**程序开始执行前**完成
- C语言的**局部自动变量**则是**在进入变量定义所在的复合语句时**为它们分配存储单元, 这种变量的大小也是静态确定的
- `malloc( n*sizeof( ) )`; `calloc( n , sizeof( ) )` 初始化为0; `realloc( ptr , n*sizeof( ) )` 重新分配存储空间
- 结构的定义**以分号`;`结束**,因为C语言把结构的定义看作一条语句
- **同结构类型**的变量之间可以直接赋值
- 字符串长度不包括 `'\0'` ,字符数组长度包括 `'\0'`

函数与程序结构

- 当多个函数发生多层嵌套调用时, 最后调用的最先结束
- 一个函数的定义不可以完整地包含另一个函数的定义
- 函数递归调用自身时, 每次调用都会得到一个与以前的变量合集不同的新变量合集
- 递归并不节省存储器开销, 因为递归调用过程中必须在某个地方维护存储处理值的栈
- 函数调用不可以作为一个函数的形参
- 一个递归算法*必须*包含*递归部分*和*终止条件*
- **编译预处理**:**文件包含**, **宏定义**, **条件编译**
- 预处理命令行不一定要在C源程序的起始位置(但一般是)
- 程序的一行中不能出现多个有效的编译预处理命令
- 宏名*无类型*, 宏替换只是*字符替换*, **不占用运行时间**, 但**占用编译时间**
- 宏定义可以出现在函数内部
- 宏定义时**声明的**参数类型并**不会限制**在使用宏时传递的参数类型
- 编译系统对宏命令的处理在对源程序中**其他语句正式编译之前**进行
- 一个大程序可由几个程序文件模块组成, 每个程序模块又可能包含多个函数,但**只能有一个** `main( )`
- `#define 宏名(不带空格) 宏定义字符串(可含空格,以'\n'结束)` 允许宏的嵌套定义
- 文件包含的作用是将指定文件插入至 `#include` 位置,编译连接时拼接形成可执行代码

- 文件包含只需提供头文件, 不需要源文件
- 文件包含时: `#include<>` 在指定文件夹中寻找; `#include""` 先在当前文件夹中找,再到指定文件夹中寻找
- 条件编译: `#if` , `#else` , `#ifdef` , `#ifndef` 等
- 全局变量在整个程序的所有文件模块都起作用, 但**只能定义一次**,故可用 `extern` 声明外部变量(**只声明, 不分配存储单元**)
- 静态全局变量, 作用域仅为该文件
- 外部函数声明: `extern int fun();` ,一般情况下外部函数声明可以省略, 一个成熟系统会自己找外部函数(
- 将一个函数定义为 `static` 静态函数后, 其他.c文件不能直接调用该函数, *增加 `extern` 外部函数声明后, 也不能*
- **头文件**可包含:**类型声明**, **全局变量声明**, **函数声明**和**常量定义**(*均不占用存储空间*)

指针进阶

- 源文件.c→编译, 连接→可执行文件.exe
- **可执行文件**可直接在**操作系统环境**下以**命令行**的方式运行
- main函数命令行形参表示: `main( int argc , char*argv[] )` 或者 `main( int argc , char **argv)` argc表示命令行参数的数量
- 用命令行方式运行程序时, 系统根据输入的命令行参数的数量和长度自动分配存储空间存放这些参数(包括命令)
- **二级指针**: `char**ptr` , 指向指针的指针
  - `int x=0; int**ptr=&&x;` ? nonono~~~
  - `int x=0; int*p=&x; int**ptr=&p;` ? yes!!!
- **指针数组**: `char *colors[5]` , 每个数组元素中存放的内容都是指向的地址
- **数组指针**: `char (*colors)[5]` , 指向数组的指针
- **函数指针**: `int (*funptr)( int ,char )`
  - 在使用函数指针前, 先对其赋值.赋值时, 将一个函数名赋给函数指针, 但该函数必须*未定义或者未声明*, 且返回值类型一致
  - 调用: `(*funptr)(a,b)` 或者 `funptr(a,b)`
- 指针作为返回值:
  - 能返回:主调函数或静态存储区中,动态内存分配且未 `free()`
  - 不能返回:函数内定义的自动变量地址
- 函数名代表**函数的入口地址**(常量), 是一个指针

文件

- C语言把文件看作数据流, 并将数据按照顺序以一维方式组织存储
- C语言源程序是文本文件, 目标文件和可执行文件是二进制文件
- 对于缓冲文件系统, 在进行文件操作时, 系统自动未每个文件分配一块文件内存缓冲区(内存单元)
- 文件指针指向存放了用文件结构体所定义的对象起始地址, 而不是文件缓冲中文件数据的存取位置
- **字符或字符串**在文本文件和二进制文件中存储字节数**相同**, 而**整型或浮点型**在文本文件和二进制文件中存储字节数**可能不同**
- 文本文件读取: `fgetc()` , `fgets()` , `fscanf()`
- 二进制文件读取: `fread()`
- 缓冲文件系统的*文件缓冲区*位于*内存数据区*中
- 保存:内存写入文件 ; 打开:把磁盘文件的内容读取到内存
- 文件:
  - 程序文件:源程序.c, 目标文件.o, 可执行文件.exe
  - 数据文件:输入输出数据(文本, 图像, 声音等)
- 文本文件:字符流(源文件)
- 二进制文件:二进制流(目标文件, 可执行文件)
- 缓冲文件系统:自动为每一个文件分配一块文件内存缓冲区(内存单元)
- 非文件缓冲系统:不自动分配, 由C语句实现分配
- 自定义类型: `typedef <已有类型名>  <新类型名>;`
  - `typedef int INT ;`
  - `typedef int NUM[10] ;NUM a;`
- fp->curp的改变隐含于文件读取中, `fp++` 指向下一个FILE结构
- 文件控制块FCB会与缓冲区相关联, FCB直接对应于磁盘
- 打开文件
  i. `FILE *fp ; fp=fopen("文件名","打开方式") ;` \ ⇒ \\
  ii. `FILE *fp ; char*p="文件名" ; fp=fopen( p , "r" ) ;` 字符指针表示文件
- 文件打开的实质是将磁盘文件与文件缓冲区对应起来, 后面的文件操作只需要使用指针
- 无法正常打开的原因:1.abc.txt不存在 2.路径不对 3.文件已被别的程序打开 4.文件存储问题
- 一旦fopen打开, 操作方式就确定了
- 可同时打开多个文件, 不同文件采用不同指针, 但不允许同一文件关闭前再次打开
- fclose(fp):返回0正常关闭, 非零不正常关闭
- 字符方式读写函数:fgetc, fputc
  - fgetc: `ch = fgetc(fp) ;` 成功返回读取的字符, 不成功返回EOF
  - fputc: `fputc( ch , fp );`
- 字符串方式读写函数:fgets, fputs
- fputs: `fputs( s, fp );` 向文件写入字符串,成功返回最后一个字符, 失败返回EOF

    - fgets: `fpets( s , n , fp )`最多读取 n-1 个字符,自动添加`'\0'`,达到 n-1 个或接受到 `'\n'` 或接受到EOF结束, **保留`'\n'` **不保留EOF ; 成功返回读取到的字符串, 失败返回NULL
- 格式化读取方式 fscanf, fprintf
    - fscanf: `fscanf( fp , "%d" , &x );`
    - fprintf: `fprintf( fp , "%d" , x );`
- 数据块读取方式 fread, fwrite
    - fread: `fread( fa , size , count , fp );`
    - fwrite: `fwrite( fb , size , count , fp );`
- 其他相关函数:
    - rewind: `rewind( fp )`回到打开文件时读写位置指针指向位置
    - fseek: `fseek( fp , offset , from );`,offset为long类型的偏移量
    - ftell: `ftell( fp );`， 相对文件开头的位移量(字节), 出错返回-1L
    - feof: `feof( fp );`, 达到结束位置返回1, 未结束返回0
    - ferror: `ferror( fp );`, 检查是否出错, 无错返回0, 有错返回非0
    - clearerror: `clearerror( fp );`清除出错标志和文件结束标志,使其为0
- 数组(如double a[10])元素存储为文件时, 使用文本文件存储时, 文件大小与sizeof(double)无关(因为以字符形式存储); 使用二进制文件存储时, 文件大小与sizeof(double)有关(二进制存储, 故有关)