计算机程序设计 期末机考 知识概要

目录

计算机程序设计 期末机考 知识概要 1
零、Visual Studio 2012 的基本使用 3
0.1 新建代码 3
0.2 编辑代码 3
0.3 编译程序 3
0.4 常见报错 3
0.5 程序调试 4
0.6 程序运行 5
0.7 常见错误 5
一、C 语言程序最基本的格式 6
1.1 示例（“插入”处为自定义部分） 6
1.2 语句 6
1.3 注释 6
1.4 缩进 6
1.5 匈牙利命名法 6
二、数据类型 6
2.1 数据类型一览 6
2.2 数的表示 7
2.3 常量 7
2.4 符号常量 8
2.5 常变量 8
2.6 类别转换 8
三、常见运算符 8
3.1 多个运算符时的使用原则 8
3.2 一些相关概念 8
3.3 常见简单运算符列表 8
四、输入函数的使用 10
4.1 格式化输入 scanf 10
4.2 字符输入 getchar 10
4.3 字符输入 getch 10
4.4 字符串输入 gets 10
五、输出函数的使用 10
5.1 格式化输出 printf 11
5.2 字符输出 putchar 11
5.3 字符串输出 puts 11
六、选择结构 11
6.1 if 语句 11
6.2 条件表达式 11
6.3 switch 语句 12

七、循环结构 12
7.1 循环的调控语句 12
7.2 while 事件控制当型循环语句 12
7.3 do-while 事件控制直到型循环语句 12
7.4 for 计数控制当型循环语句 12
八、其它简单的常用库函数 12
8.1 数学函数 12
8.2 时间函数 13
8.3 随机函数 14
8.4 字符函数 14
8.5 字符串函数 14
九、一些简单的算法 14
9.1 求迭代多次的结果——递推法 14
9.2 求符合条件的情况——穷举法 15
9.3 比大小——打擂台法 15
9.4 判断是否为质数 15
9.5 求最大公约数——辗转相除法 15
9.6 得到随机数 16
9.7 交换位置 16
十、ASCII 码 16
10.1 ASCII 码分配规律 16
10.2 ASCII 码总表 16
10.3 ASCII 码查询程序 19
十一、变量存储类型 19
11.1 动态区（栈区）变量 19
11.2 动态区（堆区）变量 19
11.3 静态区变量 19
十二、函数 19
12.1 函数的定义 19
12.2 函数的声明 19
12.3 函数的调用 20
12.4 函数参数的传递 20
12.5 函数的返回值 20
12.6 函数存储类型 20
12.7 函数指针 20
12.8 递归调用 20
12.9 排序算法 20
十三、指针和数组 22
13.1 指针变量概念 22
13.2 指针变量操作 22
13.3 变量访问方式 23
13.4 数组基本操作 23
13.5 数组退化为指针 23
13.6 一维数组元素值的表示 23

13.7 二维数组元素值的表示 23
13.8 空指针 24
13.9 指针变量定义的相关模糊表述一览 24
十四、结构体 24
14.1 结构体类型的申明、变量定义和初始化方法 24
14.2 结构体类型变量存储规则 24
14.3 结构体变量的引用 24
14.4 结构体的意义 25
14.5 链表 25
十五、其它自定义变量类型 25
15.1 枚举类型 25
15.2 定义类型别名 25

零、Visual Studio 2012 的基本使用

注：Visual Studio 2008 可能略有出入！

0.1 新建代码

第一步-新建文件：（起始页-开始）新建项目-（选择）空项目、修改文件名-确定；

第二步-新建代码：（右键左栏）源文件-添加-新建项-修改名称（*把.cpp 改成.c*）-添加；

第三步-修改页面格式：（顶栏）工具-选项-（左栏）环境-字体和颜色-修改为自己习惯的格式；

0.2 编辑代码

缩进：左大括号回车自动增加缩进，右大括号自动取消缩进，也可以手动 tab 键增加缩进；

红色波浪线：随时显示可能的输入错误，修改正确或打完整个代码会自动消失；

提示框：如果想用提示框中的函数名/变量名，需要先按↑/↓进入选择，然后选择到再回车；

0.3 编译程序

操作：（顶栏）绿三角（本地 windows 调试器）自动开始保存、编译、运行；

报错：弹出“发生生成错误……”窗口，选择否，**底栏选择错误列表**，双击错误可定位；

结束：在运行界面退出或在程序页面顶栏选择红色方形；

0.4 常见报错

原则：一般按照报错的提示进行修改，但是报错提示的内容和位置可能不准确！

事实：同一个错误可能每次报错情况不一样，且可能不报错；一个错误可能导致连带多条报错！

一些常见报错可能的原因：

C1075 **与{匹配前文件结束**：某处漏打}；

C2008 **宏定义中的意外**：宏定义的格式错误；

C2015 **常量中字符太多**：把双引号打成单引号；

C2054 **在 main 之后应输入(**：情况可能复杂——

情形 1：main 之后忘记打小括号了；

情形 2：结构体申明没打最后一个分号或结构体类型没写 struct；

C2059 **语法错误符号**：情况可能复杂——

情形 1：把 python 的语言习惯代入了 C；

情形 2：或把变量名/函数名起成了保留字；

情形 3：结构体类型申明或使用时语法错误；

C2061 **语法错误标识符**：情况可能复杂——

情形 1：需要用()的地方没有打小括号；

情形 2：（如果后面还跟了一长串报错）结构体类型使用时未在前面声明 struct；

C2065 未声明的标识符：报错会提示哪个“标识符”出现了问题——

情形 1：变量没有定义或者在定义前已经使用了；

情形 2：变量名打错了；

情形 3：使用了全角的标点符号；

情形 4：运算符或结构性单词打错了；

情形 5：函数没有声明；

情形 6：结构体申明没有打最后一个分号；

C2106 左操作数必须为左值：赋值运算符=左右的变量类型不匹配——

情形 1：指针变量赋值为值或数据变量赋值为指针；

情形 2：指针变量的数据类型不对应（可以通过强制转换解决）；

情形 3：结构体分量的理解错误导致不不对应；

情形 4：试图修改常变量或常量；

情形 5：==写成了=；

C2114 左侧指针；需要右侧的整数值：复合赋值运算符操作于指针也需要右侧为整数；

C2143 语法错误缺少符号在类型前：情况可能复杂——

情形 1：在本行或（一般是）上一行报错位置漏打了这个符号；

情形 2：有的时候在 for 的()里定义函数不被承认，需提前定义变量之后再在 for()里使用；

情形 3：有的时候编译器会要求所有变量的定义都发生在函数的最开始；

情形 4：符号常量定义的时候多打了分号；

C2146 语法错误缺少符号在变量前：在该行漏打了这个符号；

C2181 没有匹配 if 的非法 else：某处漏打}或 if 语句的结构打错了；

C2231 左操作数指向 struct，使用->：.和*运算优先级搞错了，需要添加括号；

C2297 非法，右操作数包含某某类型：复合赋值运算符需要右侧数为整数；

C2371 重定义不同基类型：函数没有声明；

C2440 无法从某指针类型转换为某类型：赋值运算符=一侧是指针一侧不是指针；

C4013 某某函数未定义：报错会提示哪个函数出现了问题——

情形 1：没有包含运用该函数需要的头文件；

情形 2：没有定义或声明这个函数就使用了；

情形 3：函数名打错了；

C4716 必须返回一个值：函数体没有写 return；

LNK2019 无法解析的外部符号：函数名不存在或未声明或打错了；

运行过程中出现弹窗：

操作：中断→顶栏红色方形→（右上角如果有紫色的文件显示就关掉）→修改程序；

Run-Time Check Failure #2 - Stack around the variable '**' was corrupted.**：scanf%后变量格式错误；

Run-Time Check Failure #3 - The variable '**' is being used without being initialized.**：变量未初始化——

情形 1：scanf 漏打&；

情形 2：没有对变量赋初值（或野指针），随机初值使后续运行出错；

0x****** (***.dll) (***.exe 中)处有未经处理的异常**：存入数据失败——

情形 1：scanf 漏打&；

情形 2：数组越界、死循环等算法错误使存入数据的地址波及到禁止使用的部分；

情形 3：试图改变常变量；

触发了一个断点：指针运算异常，可能是某处引用错了指针；

0.5 程序调试

原则：报错不一定就准，没有报错不一定就对（算法问题）!

流程：选择尽可能全的多种情况进行实验，检查输出是否符合要求，若不符合，找出错误修改；

常见简单的调试改错方法：

注释法：把一些语句前面加上//变成注释，使它在检查时不运行，后续删除//恢复语句——

情形 1：把已知正确的语句加上//，如果运行出错，则确定错误出现在未注释的部分；

情形 2：把想检查是否出错的语句加上//，如果运行正常，则确定错误是注释的语句；

输出法：在各个节点添加 *printf*，查看各处运行情况——

情形 1：*printf* 输出变量值，查看这个地方的变量是否是按照预想规律计算的；

情形 2：*printf* 输出字符串，确定发生错误的点在这一句之前还是之后；

断点法：设置断点检查程序运行到某步前的各变量值——

设置断点：在检测代码左侧细灰栏中点击，出现红点即为断点（编辑、调试时均可操作）；

删除断点：再次点击红点即删除（编辑、调试时均可操作）；

调试过程：

点击绿箭头“本地 Windows 调试器”开始调试；

遇到断点会自动暂停，下栏返回各变量值；

点击绿箭头“继续”执行直到下一个断点；

点击上栏蓝箭头选择逐语句/逐过程/跳出继续缓慢调试；

点击上栏红方块结束调试；

遇到输入处调试界面无法操作，只能输入值并回车，此后继续；

数据查看：

下栏“自动窗口”和“局部变量”自动显示部分变量、数组和指针的值；

数组还会在大括号中显示数组内的各值，左边的加号点开可以看到各个分量；

指针会在地址后面显示指向的值；

调试过程中右键代码区的变量，选择“添加监视”，可在“监视 1”窗口查看；

0.6 程序运行

多道题连续验收时的保存处理方式：

分项目保存：每道题目代码单独设一个文件保存，切换文件前记得保存代码！

同项目保存：两个 *main* 函数不能同时编译——

用注释区分：每个程序运行时把另一个程序开头结尾使用/*和*/注释掉；

用属性区分：右键左侧栏源文件名-选择“属性”，在“从生成中排除”中填“是”或“否”；

0.7 常见错误

野指针：指针没有指定指向地址即开始对其进行操作；

全局变量不修改：在函数中使用全局变量后没有恢复初值，导致再次调用时已经不是初值了；

循环变量重复使用：循环嵌套循环用了同一个循环变量；

缺少退出条件：递归或循环进入死循环；

变量类型错误：不注意函数返回值或指针的变量数据类型；

用了不还：*malloc* 借用空间不 *free*，或读写文件之后不 *fclose*；

结构体取分量和取指针的运算优先级：.的优先级高于*，注意打括号；

指针变量内容搞错：到底是几级指针、指针对应空间长度是多少；

忘记强制转换：*malloc* 等涉及到指针变量类型的地方类型不匹配；

指针/数组和一般的变量名搞混：特别出在输入输出包含字符串/数组的时候；

判断相等写成赋值：==写成了=；

未进行初始化：没有对变量赋初值或者变量的存储类型错误定义造成初值与自己所想不同；

字符串终止符遗漏：字符串定义空间不足或赋值错误造成末尾不存在’\0’无法正常处理；

分号处理错误：漏打分号或把逗号和分号搞混；

一、C 语言程序最基本的格式

1.1 示例（“插入”处为自定义部分）

```
#include<stdio.h>/*正常运行必需的头文件*/
#include<stdlib.h>/*system pause所需的头文件*/
#pragma warning(disable:4996)/*取消把输入输出函数加_s的报错*/
/*插入其它文件、头文件的文件包含*/
/*插入宏定义、全局变量定义或声明、函数的定义或声明*/
int main(){/*定义主函数*/
    /*插入主函数体程序*/
    system("pause");/*使运行页面停留方便检查*/
    return 0;/*主函数回复值，如果定义的主函数类型是void，则不写*/
}
/*插入全局变量的定义、函数的定义*/
```

1.2 语句

注：从这里开始，后续文字中的“[]”均只表示分隔，无实义，不出现在程序中；

结构：[表达式];;

意义：C 程序的基本单位；

复合语句：{[表达式 1];[表达式 2];…}——

定义：只要打花括号一般就是复合语句，即几个语句的组合，一般可以代替单个语句；

注意：复合语句中定义的变量和声明的函数只在花括号内作用，一定不要忘右括号；

说明：从这里开始，后续文字中无特殊说明“[语句]”均包含复合语句的情形；

1.3 注释

格式：/*[注释内容]*/或//[注释内容]；

作用：编译时直接忽略，但是可以增强可读性，便于检查；

位置：任意位置均可；

1.4 缩进

方式：tab 键插入制表符，编译器在之后的输入中也会按规律自动添加；

作用：编译时直接忽略，但是可以增强可读性，体现结构，便于检查；

位置：有无、在什么位置理论上都无所谓，但是一般使用在——

情形 1：复合语句内部增加相同缩进表示同层，编译器会对大括号自动产生、结束；

情形 2：选择结构 else if 太多的时候表示判断的次数，需要自己添加；

1.5 匈牙利命名法

方式：小写类型缩写+首字母大写变量/函数意义单词；

作用：方便一眼认出变量/函数的意思，增强可读性；

二、数据类型

2.1 数据类型一览

注 1：占用内存空间长度为 vs2012 中的情形，各个编译器或有差异，但是大小关系出入不大；

注 2：%x/%X 为十六进制格式，%o 为八进制格式，%e/E 为指数形式格式，输出大小写同说明；

注 3：实型的取值范围为理论上的大约取值范围，其实根本取不到；

注 4：嫌 short 还是太长可以用 char 存储整型变量；

注 5：bool 类型在 vs2012 中无法使用；

注 6：指针用整型格式输出也是差不多的，32 位中长度为 4B；

注 7：指针的空间占用指保存其需要的空间，非其增量的基本长度；

注 8：表中给出的仅为常见数据类型，C 语言还可以自定义数据类型；

大类	类型	关键字	格式说明	空间	取值范围	编码方式
基本类型						
整型	无符号短整型	unsigned short	%hu	2B	0~2^16-1	补码
	短整型	short	%h	2B	-2^15~2^15-1	
	无符号整型	unsigned int	%u	4B	0~2^32-1	
	整型	int	%i(%x/%X/%o)	4B	-2^31~2^31-1	
	无符号长整型	unsigned long	%lu	4B	0~2^32-1	
	长整型	long	%ld	4B	-2^31~2^31-1	
	无符号超长整型	unsigned __int64	%I64u/%llu	8B	0~2^64-1	
	超长整型	__int64	%I64d/%lld	8B	-2^63~2^63-1	
实型（浮点型）	单精度实型	float	%f(%e/%E)	4B	1E-37~1E38	偏移码
	双精度实型	double	%lf(%le/%E)	8B	1E-307~1E308	
	长精度实型	long double	%llf(%lle/%llE)	8B	1E-307~1E308	
字符型	字符型	char	%c	1B	0~2^8-1	ASCII 码
布尔型	布尔型	bool	——	1B	0 和 1	——
指针类型						
指针	一级指针	[类型]*	%p	8B	——	二进制数
	n 级指针	[类型][n 个*]		8B		
	行指针	[类型](*)[长度]		8*[长度]B		
空类型						
空类型	空类型	void	——	0B	——	——
构造类型						
数组	一维数组	[类型][[长度]]	——	[类型长]*[长度]B	——	与类型同
	二维数组	[类型][[长度]][[长度]]		[类型长]*[总长]B		
	字符串	char*[长度]	%s	[长度]B		ASCII 码
结构体	结构体	struct[类型]	——	有关对齐	——	与分量同
共用体	共用体（联合体）	union[类型]	——	最大分量	——	与分量同
枚举型	枚举型	enum[类型]	——	4B	-2^31~2^31-1	补码

2.2 数的表示

注：计算机不知道什么原码、反码之类的，它只知道二进制机器码！这些是方便人理解制造的！

原码：原数；

反码：正数反码和原码相同，负数反码是对原码除符号位外取反；

补码：整数补码和原码相同，负数补码是反码加 1，规定首位为 1 其它位为 0 为最低的负数；

亦即：对最大的二的幂次取模；

偏移码：补码的符号位取反（保持对最大的二的幂次取模的顺序不变）；

加法运算：补码之和为和的补码，偏移码之和符号位取反为和的偏移码，原码反码不能做加法；

减法运算：减数取相反数进行加法运算；

2.3 常量

注：常量储存在代码区！

整型常量：输入整数默认整型，0 开头默认八进制，0x 开头默认十六进制；

实型常量：输入小数点默认小数形式双精度实型，中间加 E 默认指数形式双精度实型；

调整实型常量数据类型：末尾加 L 代表长精度，末尾加 f 代表单精度；

实型常量比较大小：由于有误差，不可以使用“==”，可以两数相减小于 1E-10 之类的方式；

字符型常量：单个字符——

单撇号表示法：对一般字符，一对’’括起来一个字符（但不能是单撇好或者反斜杠\）；

转义字符表示法：对特殊字符，\[字符或三位数]——

[三位数]为三位八进制数——八进制数代表的 ASCII 码对应的字符；

[三位数]为 x+二位十六进制数——十六进制数代表的 ASCII 码对应的字符；

[字符]对应的字符如下：

形式	含义	ASCII 码	形式	含义	ASCII 码
\n	换行	10	\t	横向跳格（制表符）	9
\r	回车（回行首）	13	\v	竖向跳格	11
\b	退格	8	\\	反斜杠字符	92
\f	走纸换行	12	\’	单引号字符	39

2.4 符号常量

本质：无参宏；

定义方式：预编译指令中#define[宏名][值]；（*没有分号！*）

编码方式：补码（所以只能是整型/字符型）；

优点：增强可读性，方便修改；

2.5 常变量

特点：定义后后续程序不可将其值更改；

定义方式：定义时 const[变量类型]-[值];；

优点：比符号常量类型更加多样，比一般变量增强了编辑合作时的安全性；

2.6 类别转换

原因：运算需要编码方式相同才能正确进行，函数实参和形参的数据类型产生了差异；

强制类别转换：手动按自己的意愿进行，格式——([转换到的类型关键字])[被转换的量];;

注：malloc 的返回值必须强制类型转换为所需的指针类型！

自动类别转换：计算机默认的转换规则，可能不合自己的意愿，规则如下逐步进行：

char/short→int→unsigned int→long→**double**（最万不得已的结果）←float

注 1：强制转换/计算自动转换整型⇔实型时补码和偏移码会互相转换，但是输入输出不会！

注 2：与指针进行运算的整型值会自动转换为对应指针的类型对应长度增值；

三、常见运算符

3.1 多个运算符时的使用原则

用括号控制运算顺序可以更直观、减少出错；

在无法确定优先级的时候可以拆分表达式；

3.2 一些相关概念

结合律：同一个运算符连续出现多次时，**左结合**是从左往右依次做，**右结合**是从右往左依次做；

优先级：不同运算符在一个语句中出现时，先执行优先级高的部分，同一优先级参考结合律；

注：括号可以调整运算顺序，使运算不按照默认的结合律和优先级进行，优先括号内；

3.3 常见简单运算符列表

注：表中优先级对应的数越小说明优先级越高；

运算符	含义	目数	结合律	优先级	备注
算术运算符（5 种+自增自减各 1 种）					
+	加法	双目	左结合	5	
	正值	单目	右结合	2	一般可以省略
-	减法	双目	左结合	5	
	负值	单目	右结合	2	

*	乘法	双目	左结合	4	
/	除法（求商）	双目	左结合	4	整型结果向下取整
%	除法（求余数）	双目	左结合	4	
++(i)	前置自增（i=i+1，使用 i）	单目	右结合	2	
(i)++	后置自增（使用 i，i=i+1）	单目	右结合	2	
--(i)	前置自减（i=i-1，使用 i）	单目	右结合	2	
(i)--	后置自减（使用 i，i=i-1）	单目	右结合	2	
赋值运算符（1 种+10 种复合赋值运算符[减少计算次数]）					
=	计算右边值赋给左边变量	双目	右结合	15	
(y)+=(x)	y=y+x	双目	右结合	15	
(y)-=(x)	y=y-x	双目	右结合	15	
(y)*=(x)	y=y*x	双目	右结合	15	
(y)/=(x)	y=y/x	双目	右结合	15	
(y)%=(x)	y=y%x	双目	右结合	15	
(y)<<=(x)	y=y<<x	双目	右结合	15	
(y)>>=(x)	y=y>>x	双目	右结合	15	
(y)&=(x)	y=y&x	双目	右结合	15	
(y)|=(x)	y=y|x	双目	右结合	15	
(y)^=(x)	y=y^x	双目	右结合	15	
关系运算符（6 种[真返回 1，假返回 0]）					
<	小于	双目	左结合	7	
<=	小于等于	双目	左结合	7	
>	大于	双目	左结合	7	
>=	大于等于	双目	左结合	7	
==	等于	双目	左结合	8	不要只写一个=
!=	不等于	双目	左结合	8	
逻辑运算符（3 种[只要能出确定结果就不往右做了]）					
!	非	单目	右结合	2	
&&	与	双目	左结合	12	
||	或	双目	左结合	13	
位运算符（6 种[处理二进制位的问题，现在用不上]）					
<<	左移	双目	左结合	6	
>>	右移	双目	左结合	6	
&	位逻辑与	双目	左结合	9	
|	位逻辑或	双目	左结合	11	
^	位逻辑异或	双目	左结合	10	
~	位逻辑非	单目	右结合	2	
逗号运算符（1 种）					
,	连接多个表达式返回末式值	双目	左结合	16	常在 for()中用
求字节数运算符（1 种）					
sizeof(x)	求 x 在内存中占的字节数	单目	右结合	2	x 为类型或运算值
条件运算符（1 种）					
?:	一个表达式完成选择结构	三目	右结合	14	详见“选择结构”

指针运算符（2种）					
&	取地址	单目	右结合	2	详见“指针”
*	取指针对应值、定义指针	单目	右结合	2	详见“指针”
分量运算符（2种）					
.	取结构体分量	双目	左结合	1	详见“结构体”
->	取指针对应结构体的分量	双目	左结合	1	详见“结构体”
强制类型转换运算符（1种）					
(类型)	强制转换为该类型值	单目	右结合	2	
函数调用运算符（1种）					
()				1	
下标运算符（1种）					
[]				1	

四、输入函数的使用

一般规则：缓冲区非空先按要求提取缓冲区，若缓冲区空，要求操作者输入存入缓冲区，继续提取；

4.1 格式化输入 scanf

功能：一次性进行任意类型的输入，与 scanf_s 差异不大；

使用结构：scanf(“%[类型][分隔符]%[类型]…”,&[变量名],[变量地址],…);;

注：&指取地址，所以指针/字符数组名不能加&！并且不能直接读入整型或浮点型数组！

%类型：存储空间和编码方式**必须与对应变量类型一致**，顺序和后面的变量名顺序一致；

分隔变量：系统分隔变量的时机可能是——

情形 1：当遇到指定分隔符的时候；

情形 2：当变量对应空间存满了的时候（如%c）；

情形 3：如果没有指定分隔符，那么任何空白（空格、制表符等）都被认为是分隔符；

/*如果”%d”则是读取完全部非空白数字；”%s”遇到分隔符则停止读取并补’\0’*/

输入相关的提示信息：不要出现在 scanf 里，在 scanf 前加一个 printf 语句输出提示；

字符输入：%c 只能吃一个字符，后续的字符和\n\r 都移到了下一个输入的变量——

解决方法 1：紧跟一个%c，把缓冲区的内容存入另一个在之后不会再碰的字符变量中；

解决方法 2：在%c 之后结束本次输入，下一行写 getchar();，然后再继续 scanf；

注：scanf 完成整型或浮点型录入之后缓冲区里剩下的内容不定是什么，尽可能一次录入各变量！

4.2 字符输入 getchar

功能：从缓冲区中提取**一个字符**（字节），若缓冲区空，先存入缓冲区再提取；

使用结构：[变量名]=getchar();

注意：getchar 是无参函数，且只能输入一个字符，提示信息在前面用 printf；

回车换行：单独作为一个字符需要提取，vs2012 中提取出来为\n（ASCII：10）；

4.3 字符输入 getch

需要头文件：conio.h；

功能：从缓冲区中提取一个**字符**（字节），若缓冲区空，**直接读取键盘操作对应字符**；

使用结构：[变量名]=getch();

注意：getch 是无参函数，且只能输入一个字符，提示信息在前面用 printf；

回车换行：单独作为一个字符，vs2012 中提取出来为\r（ASCII：13）；

4.4 字符串输入 gets

功能：一次性读取整行字符串，除了回车不认其它分隔符，一律保存进入字符串；

使用结构：gets([字符数组名]);

五、输出函数的使用

5.1 格式化输出 printf

功能：一次性输入各类常量、变量；

使用结构：*printf*(“[字符串]%[类型][字符串]%[类型]…”,[变量名],*[变量地址],…);;

注 1:“[变量名]”可以改为直接写出的常量或表达式；

注 2:注意指针需要添加取值运算符*（字符串除外），不可以直接输出整型或浮点型数组！

字符串常量输出：原样输出，可以包含\n，\t 等转义字符实现输出的调整；

整型输出的位数调整：%[位数][类型]，位数表示至少输出多少位，不足的用空格补；

浮点型输出的位数调整：%[输出位数][类型][小数位数]——

输出位数：整个浮点数输出的最小占用位数，小数点算一位；

小数位数：保留小数的最少位数，保留位数太多会因为进制转换而产生误差；

输出字符的 ASCII 码：鉴于字符型以补码形式存储，以%d 或%h 的形式输出 ASCII 码；

指针输出：%p 与%X 结果相同，%x 输出小写字母的十六进制，%d 转换为十进制；

5.2 字符输出 putchar

功能：输出一个字符；

使用结构：*putchar*([字符]);;

注意：如果字符写的是数，那么转换成 ASCII 码；如果字符写的是字符串，会出问题；

5.3 字符串输出 puts

功能：输出字符串，把’\0’变为’\n’；

使用结构：*puts*([字符数组名]);

六、选择结构

6.1 if 语句

使用结构：*if*([表达式])[语句]*else if*([表达式])[语句]}… *else*[语句];

注：如果没有需要的话，else if 和 else 可以省略任意个，也可以加无数个 else if;

执行流程：按顺序判断，符合条件就执行——

第一步：判断 *if* 里的表达式——

情形 1：若为 1，则执行后面的语句，结束选择；

情形 2：若为 0，且选择结构截止，直接结束选择；

情形 3：若为 0，且之后就是 *else*，执行第三步；

情形 4：若为 0，且之后还有 *else if*，跳到最近的 *else if*，执行第二步；

第二步：判断 *else if* 里的表达式——

情形 1：若为 1，则执行后面的语句，结束选择；

情形 2：若为 0，且选择结构截止，直接结束选择；

情形 3：若为 0，且之后就是 *else*，执行第三步；

情形 4：若为 0，且之后还有 *else if*，跳到最近的 *else if*，执行第二步；

第三步：判断 *else* 里的表达式——

情形 1：若为 1，则执行后面的语句，结束选择；

情形 2：若为 0，且选择结构截止，直接结束选择；

情形 3：若为 0，且之后就是 *else*，执行第三步；

情形 4：若为 0，且之后还有 *else if*，跳到最近的 *else if*，执行第二步；

6.2 条件表达式

使用结构：[变量名]=[表达式 1]?[表达式 2]:[表达式 3];;

注：三个表达式都不能是复合语句；如果不需要赋值，可以省略前面的变量名；

执行流程：判断表达式 1 的值——

情形 1：若为 1，执行表达式 2 并为变量赋值；

情形 2：若为 0，执行表达式 3 并为变量赋值；

6.3 switch 语句

使用结构：switch([变量名]){case [值]: [语句];break; … default: [语句]};

注：case 可以有多个，中间的 break 可以不写，default 可以不写然后省略最后一个 case 的 break;

执行流程：从上到下比对变量的值和 case 的值直到相等，从上到下执行后续语句直到 break;

注：default 包括了所有 case 后没有的值的情况，如果没有 default，则未出现的值就不执行;

七、循环结构

注：为了防止死循环，一般循环内语句为复合语句;

7.1 循环的调控语句

break;：终止循环，开始执行循环结束后的语句；

continue;：结束本次循环过程，进行下一轮；

7.2 while 事件控制当型循环语句

使用结构：while ([表达式])[语句];

执行流程：每次循环开始时判断，若表达式值为 1，执行一次循环语句，否则跳出循环；

7.3 do-while 事件控制直到型循环语句

使用结构：do [语句] while ([表达式]);;

执行流程：先执行再判断——

第一步：执行一次语句，再执行第二步；

第二步：判断表达式的值，若为 1，执行第一步，否则跳出循环；

7.4 for 计数控制当型循环语句

使用结构：for([表达式 1];[表达式 2];[表达式 3])[语句];

注：表达式 1 一般是循环变量的赋初值，但是对循环变量的定义最好发生在循环之前；

执行流程：三部曲——

第一步：执行表达式 1，再执行第二步；

第二步：判断表达式 2 的值，若为 0，跳出循环，若为 1，执行第三步；

第三步：执行表达式 3，再执行第二步；

八、其它简单的常用库函数

8.1 数学函数

注：使用本部分函数前需要#include <math.h>;

函数名	函数与形参类型	函数返回值	备注
三角函数（无 cot、sec、csc，亦无 long double 形式）			
sin	double sin (double x)	sin(x)	x 单位为弧度
cos	double cos (double x)	cos(x)	x 单位为弧度
tan	double tan (double x)	tan(x)	x 单位为弧度
sinf	float sin (float x)	sin(x)	x 单位为弧度
cosf	float cos (float x)	cos(x)	x 单位为弧度
tanf	float tan (float x)	tan(x)	x 单位为弧度
反三角函数（无 arccot、arcsec、arccsc）			
asin	double asin (double x)	arcsin(x)	-1<=x<=1
acos	double acos (double x)	arccos(x)	-1<=x<=1
atan	double atan (double x)	arctan(x)	
asinf	float asin (float x)	arcsin(x)	-1<=x<=1
acosf	float acos (float x)	arccos(x)	-1<=x<=1

atanf	float atan (float x)	arctan(x)	
asinl	long double asin (long double x)	arcsin(x)	-1<=x<=1
acosl	long double acos (long double x)	arccos(x)	-1<=x<=1
atanl	long double atan (long double x)	arctan(x)	
双曲函数（无 coth、sech、csch，亦无反双曲函数）			
sinh	double sinh (double x)	sinh(x)	
cosh	double cosh (double x)	cosh(x)	
tanh	double tanh (double x)	tanh(x)	
sinhj	float sinh (float x)	sinh(x)	
coshf	float cosh (float x)	cosh(x)	
tanhf	float tanh (float x)	tan(x)	
sinhl	long double sinh (long double x)	sinh(x)	
coshl	long double cosh (long double x)	cosh(x)	
tanhl	long double tanh (long double x)	tanh(x)	
乘幂相关函数（C 语言没有直接进行乘幂的运算符）			
exp	double exp (double x)	e^x	
pow	double pow (double x, double y)	x^y	
sqrt	double sqrt (double x)	x^(1/2)	
expf	float exp (float x)	e^x	
powf	float pow (float x, float y)	x^y	
sqrtf	float sqrt (float x)	x^(1/2)	
expl	long double exp (long double x)	e^x	
powl	long double pow (long double x, long double y)	x^y	
sqrtl	long double sqrt (long double x)	x^(1/2)	
对数函数			
log	double log (double x)	ln(x)	x>0
log10	double log10 (double x)	$\log_{10}(x)$	x>0
logf	float log (float x)	ln(x)	x>0
log10f	float log10 (float x)	$\log_{10}(x)$	x>0
logl	long double log (long double x)	ln(x)	x>0
log10l	long double log10 (long double x)	$\log_{10}(x)$	x>0
绝对值函数			
abs	int abs (int x)	|x|	
fabs	double fabs (double x)	|x|	
fabsf	float fabsf (float x)	|x|	
fabsl	long double fabsl (long double x)	|x|	
labs	long int labs (long int x)	|x|	头文件为 stdlib.h
llabs	long long int llabs (long long int x)	|x|	头文件为 stdlib.h
其它常用数学函数			
floor	double floor (double x)	[x]	
fmod	double fmod (double x, double y)	x/y 的余数	

8.2 时间函数

注：使用本部分函数前需要#include <time.h>，计时标准点为 1970 年 1 月 1 日 0 点整；

函数名称	函数与形参类型	函数作用
clock	int clock(NULL)	从程序开始运行到现在的毫秒数
time	int time(int)（形参输入0或NULL，但不可空）	从计时标准点开始到现在的秒数

8.3 随机函数

注：使用本部分函数前不需要额外添加头文件；

函数名称	函数与形参类型	函数作用
srand	int srand(unsigned int)	修改系统默认的“种子”
rand	void rand()	根据种子做复杂递推得到一个无从得知的整数（0~32767）

8.4 字符函数

注：使用本部分函数前需要#include <string.h>;

函数名称	函数与形参类型	函数作用
isalnum	int isalnum(int ch)	ch是数字返回4，是字母返回1，否则返回0
isalpha	int isalpha(int ch)	ch是字母返回1，否则返回0
iscntrl	int iscntrl(int ch)	ch是控制字符返回1，否则返回0
isdigit	int isdigit(int ch)	ch是数字返回1，否则返回0
isgraph	int isgraph(int ch)	ch是可打印字符（不包括空格）返回1，否则返回0
islower	int islower(int ch)	ch是小写字母返回1，否则返回0
isprint	int isprint(int ch)	ch是可打印字符（包括空格）返回1，否则返回0
ispunct	int ispunct(int ch)	ch是标点字符（不包括空格）返回1，否则返回0
isspace	int isspace(int ch)	ch是空格、跳格符或换行符返回1，否则返回0
isupper	int isupper(int ch)	ch是大写字母返回1，否则返回0
isxdigit	int isxdigit(int ch)	ch是十六进制字符返回1，否则返回0
tolower	int tolower(char ch)	把ch转换为小写字母，返回该字母
toupper	int toupper(char ch)	把ch转换为大写字母，返回该字母

8.5 字符串函数

注：使用本部分函数前需要#include <string.h>;

函数名称	函数与形参类型	函数作用
strcat	char *strcat(char* str1, char* str2)	把str2接到str1（去掉’\0’后）后面，返回字符串
strchr	char *strchr(char *str, int ch)	str中找到第一次出现ch位置返回该指针，否则NULL
strcmp	int strcmp(char *str1, char *str2)	按字典顺序比较字符串，返回小于-1，等于0，大于1
strcpy	char *strcpy(char *str1,char *str2)	把str2指向的字符串复制到str1，返回str1指针
strlen	unsigned int strlen(char *str)	返回str中字符（不包括’\0’）的个数
strncpy	char *strncpy(char*str1,char*str2,int n)	把str2前n个字符复制到str1中，返回str1指针
strstr	char *strstr(char *str1,char *str2)	str1中str2第一次出现的位置返回指针，否则NULL

九、一些简单的算法

注：代码中的标记可能是需要替换掉的，只表示思路，请勿直接照搬。

9.1 求迭代多次的结果——递推法

单变量递推：若有a=f(a)，求fn(a)——

```
int a;
int i=0;
for (;i<n;++i){
    a=f(a);
}
```

```
    printf("%d",a);
```

n 个变量递推：若有 $a=g(f_1(a_1),\cdots,f_{n-1}(a_{n-1}))$，求 $g^{mn}(a)$——

```
    int a1,a2,...,an;
    int i=0;
    for (;i<n;++i){
        a1=g(f(a2),f(a3),...,f(an);
        a2=g(f(a3),...,f(an),f(a1);
        ...;
        an=g(f(a1),f(a2),f(a3),...);
    }
    printf("%d",an);
```

9.2 求符合条件的情况——穷举法

思想：把每种情况全部列举，挑出符合条件的情况进行输出；

实现：使用循环的嵌套，每轮循环解决一类的情况；

9.3 比大小——打擂台法

求输入 n 个数中的最大值并输出——

```
    int x,y,i;
    scanf("%d",&x);
    for (i=1;i<n;i++){
        scanf("%d",&y);
        (y>=x)?(x=y):(x=x);
    }
    printf("%d",x);
```

9.4 判断是否为质数

判断输入的整数 x 是否为素数（需要提前包含 math.h 以求平方根）——

```
#define prime 0
#define not_prime 1
    int x,i,iFlag=prime;
    scanf("%d",&x);
    for (i=2;i<=(int)sqrt((double)x);i++){
        if (x%i==0){iFlag=not_prime;break;}
    }
    (iFlag)?printf("%d 不是质数。",x):printf("%d 是质数。",x);
```

9.5 求最大公约数——辗转相除法

求 a 和 b 的最大公约数——

```
int a,b;
scanf("%d%d",&a,&b);
while (1){
    if (b>0){
        a%=b;
        if (a>0) b%=a;
        else break;
    }
    else break;
```

```
}
printf("最大公约数是%d。",(a==0)?b:a);
```

9.6 得到随机数

求 m 到 n 之间的伪随机数：每次启动程序输出是一样的——

```
int i=rand()%(n-m+1)+m;;
```

求 m 到 n 之间的比较真的随机数：引入运行时间，每次的输出就不一样了——

```
#include<time.h>
    int i;/*定义保存随机数的变量*/
    srand(time(0));/*利用当前时间生成随机种子*/
    i=rand()%(n-m+1)+1;/*根据种子生成一个大数，取模并平移到所需范围*/
```

9.7 交换位置

把 a 和 b 交换位置：增加一个暂时储存的变量——

```
#include <stdio.h>
#include <stdlib.h>
int a,b;
int iSwap(){
    int *pa=&a, *pb=&b,
        //暂时存储被转出的变量
        iTemp=*pa;
    *pa=*pb;
    *pb=iTemp;
    return 0;
}
int main(){
    iSwap();
    system("pause");
    return 0;
}
```

十、ASCII 码

10.1 ASCII 码分配规律

标点符号：0x20~0x2F 0x3A~0x40 0x5B~0x60 7B~7E，即 32~47 58~64 91~96 123~126；

数字：0x30~0x39，即 48~57，数字 N 对应 0x3N，即 48+N；

大写字母：0x41~0x5A，即 65~90，按 A~Z 顺序依次排；

小写字母：0x61~0x7A，即 97~122，按 a~z 顺序排，同字母小写码=大写码+32；

10.2 ASCII 码总表

Bin (二进制)	Oct (八进制)	Dec (十进制)	Hex (十六进制)	缩写/字符	解释
0000 0000	00	0	0x00	NUL(null)	空字符
0000 0001	01	1	0x01	SOH(start of headline)	标题开始
0000 0010	02	2	0x02	STX (start of text)	正文开始
0000 0011	03	3	0x03	ETX (end of text)	正文结束
0000 0100	04	4	0x04	EOT (end of transmission)	传输结束
0000 0101	05	5	0x05	ENQ (enquiry)	请求
0000 0110	06	6	0x06	ACK (acknowledge)	收到通知
0000 0111	07	7	0x07	BEL (bell)	响铃
0000 1000	010	8	0x08	BS (backspace)	退格

0000 1001	011	9	0x09	HT (horizontal tab)	水平制表符
0000 1010	012	10	0x0A	LF (NL line feed, new line)	换行键
0000 1011	013	11	0x0B	VT (vertical tab)	垂直制表符
0000 1100	014	12	0x0C	FF (NP form feed, new page)	换页键
0000 1101	015	13	0x0D	CR (carriage return)	回车键
0000 1110	016	14	0x0E	SO (shift out)	不用切换
0000 1111	017	15	0x0F	SI (shift in)	启用切换
0001 0000	020	16	0x10	DLE (data link escape)	数据链路转义
0001 0001	021	17	0x11	DC1 (device control 1)	设备控制 1
0001 0010	022	18	0x12	DC2 (device control 2)	设备控制 2
0001 0011	023	19	0x13	DC3 (device control 3)	设备控制 3
0001 0100	024	20	0x14	DC4 (device control 4)	设备控制 4
0001 0101	025	21	0x15	NAK (negative acknowledge)	拒绝接收
0001 0110	026	22	0x16	SYN (synchronous idle)	同步空闲
0001 0111	027	23	0x17	ETB (end of trans. block)	结束传输块
0001 1000	030	24	0x18	CAN (cancel)	取消
0001 1001	031	25	0x19	EM (end of medium)	媒介结束
0001 1010	032	26	0x1A	SUB (substitute)	代替
0001 1011	033	27	0x1B	ESC (escape)	换码(溢出)
0001 1100	034	28	0x1C	FS (file separator)	文件分隔符
0001 1101	035	29	0x1D	GS (group separator)	分组符
0001 1110	036	30	0x1E	RS (record separator)	记录分隔符
0001 1111	037	31	0x1F	US (unit separator)	单元分隔符
0010 0000	040	32	0x20	(space)	空格
0010 0001	041	33	0x21	!	叹号
0010 0010	042	34	0x22	"	双引号
0010 0011	043	35	0x23	#	井号
0010 0100	044	36	0x24	$	美元符
0010 0101	045	37	0x25	%	百分号
0010 0110	046	38	0x26	&	和号
0010 0111	047	39	0x27	'	闭单引号
0010 1000	050	40	0x28	(	开括号
0010 1001	051	41	0x29	)	闭括号
0010 1010	052	42	0x2A	*	星号
0010 1011	053	43	0x2B	+	加号
0010 1100	054	44	0x2C	,	逗号
0010 1101	055	45	0x2D	-	减号/破折号
0010 1110	056	46	0x2E	.	句号
0010 1111	057	47	0x2F	/	斜杠
0011 0000	060	48	0x30	0	字符 0
0011 0001	061	49	0x31	1	字符 1
0011 0010	062	50	0x32	2	字符 2
0011 0011	063	51	0x33	3	字符 3
0011 0100	064	52	0x34	4	字符 4
0011 0101	065	53	0x35	5	字符 5
0011 0110	066	54	0x36	6	字符 6
0011 0111	067	55	0x37	7	字符 7
0011 1000	070	56	0x38	8	字符 8
0011 1001	071	57	0x39	9	字符 9
0011 1010	072	58	0x3A	:	冒号
0011 1011	073	59	0x3B	;	分号
0011 1100	074	60	0x3C	<	小于

0011 1101	075	61	0x3D	=	等号
0011 1110	076	62	0x3E	>	大于
0011 1111	077	63	0x3F	?	问号
0100 0000	0100	64	0x40	@	电子邮件符号
0100 0001	0101	65	0x41	A	大写字母 A
0100 0010	0102	66	0x42	B	大写字母 B
0100 0011	0103	67	0x43	C	大写字母 C
0100 0100	0104	68	0x44	D	大写字母 D
0100 0101	0105	69	0x45	E	大写字母 E
0100 0110	0106	70	0x46	F	大写字母 F
0100 0111	0107	71	0x47	G	大写字母 G
0100 1000	0110	72	0x48	H	大写字母 H
0100 1001	0111	73	0x49	I	大写字母 I
01001010	0112	74	0x4A	J	大写字母 J
0100 1011	0113	75	0x4B	K	大写字母 K
0100 1100	0114	76	0x4C	L	大写字母 L
0100 1101	0115	77	0x4D	M	大写字母 M
0100 1110	0116	78	0x4E	N	大写字母 N
0100 1111	0117	79	0x4F	O	大写字母 O
0101 0000	0120	80	0x50	P	大写字母 P
0101 0001	0121	81	0x51	Q	大写字母 Q
0101 0010	0122	82	0x52	R	大写字母 R
0101 0011	0123	83	0x53	S	大写字母 S
0101 0100	0124	84	0x54	T	大写字母 T
0101 0101	0125	85	0x55	U	大写字母 U
0101 0110	0126	86	0x56	V	大写字母 V
0101 0111	0127	87	0x57	W	大写字母 W
0101 1000	0130	88	0x58	X	大写字母 X
0101 1001	0131	89	0x59	Y	大写字母 Y
0101 1010	0132	90	0x5A	Z	大写字母 Z
0101 1011	0133	91	0x5B	[	开方括号
0101 1100	0134	92	0x5C	\	反斜杠
0101 1101	0135	93	0x5D	]	闭方括号
0101 1110	0136	94	0x5E	^	脱字符
0101 1111	0137	95	0x5F	_	下划线
0110 0000	0140	96	0x60	`	开单引号
0110 0001	0141	97	0x61	a	小写字母 a
0110 0010	0142	98	0x62	b	小写字母 b
0110 0011	0143	99	0x63	c	小写字母 c
0110 0100	0144	100	0x64	d	小写字母 d
0110 0101	0145	101	0x65	e	小写字母 e
0110 0110	0146	102	0x66	f	小写字母 f
0110 0111	0147	103	0x67	g	小写字母 g
0110 1000	0150	104	0x68	h	小写字母 h
0110 1001	0151	105	0x69	i	小写字母 i
0110 1010	0152	106	0x6A	j	小写字母 j
0110 1011	0153	107	0x6B	k	小写字母 k
0110 1100	0154	108	0x6C	l	小写字母 l
0110 1101	0155	109	0x6D	m	小写字母 m
0110 1110	0156	110	0x6E	n	小写字母 n
0110 1111	0157	111	0x6F	o	小写字母 o
0111 0000	0160	112	0x70	p	小写字母 p

0111 0001	0161	113	0x71	q	小写字母 q
0111 0010	0162	114	0x72	r	小写字母 r
0111 0011	0163	115	0x73	s	小写字母 s
0111 0100	0164	116	0x74	t	小写字母 t
0111 0101	0165	117	0x75	u	小写字母 u
0111 0110	0166	118	0x76	v	小写字母 v
0111 0111	0167	119	0x77	w	小写字母 w
0111 1000	0170	120	0x78	x	小写字母 x
0111 1001	0171	121	0x79	y	小写字母 y
0111 1010	0172	122	0x7A	z	小写字母 z
0111 1011	0173	123	0x7B	{	开花括号
0111 1100	0174	124	0x7C	|	垂线
0111 1101	0175	125	0x7D	}	闭花括号
0111 1110	0176	126	0x7E	~	波浪号
0111 1111	0177	127	0x7F	DEL (delete)	删除

10.3 ASCII 码查询程序

十进制 ASCII 码查询：printf(“%d”,’[字符]’);

十六进制 ASCII 码查询：printf(“%x”,’[字符]’);

八进制 ASCII 码查询：printf(“%o”,’[字符]’);

十一、变量存储类型

11.1 动态区（栈区）变量

定义方式：大括号内定义的变量自动为动态区，大括号外定义需要前置 auto;

初值定义：默认初值是无法使用的 CC，动态刷新改变值;

生命周期：只在大括号内存在，出大括号自动释放空间;

***注**：在大括号外但是在生命周期内时需要指针间接调用;*

11.2 动态区（堆区）变量

定义方式：malloc([字节数]);申请堆区空间，返回 void*指针指向第一个字节;

***注 1**：使用 malloc 后需要检查返回值是否为 NULL（无连续空间）;*

***注 2**：使用完空间一定要记得在程序里用 free([指针]);释放！但是程序爆掉会自行释放;*

初值定义：默认值是无法使用的 FE，动态刷新改变值;

生命周期：存在直到空间释放;

11.3 静态区变量

定义方式：程序开始时大括号外的全局变量自动为静态区，大括号内定义需前置 static;

初值定义：默认初始值 0，只会被初始化一次;

生命周期：程序运行时始终存在，但是在一些区域无法直接调用，需要使用指针;

十二、函数

12.1 函数的定义

有参函数：[类型名][函数名]([类型名][形参名],…){[语句]; return [返回值];}

无参函数：[类型名][函数名](void){[语句]; return [返回值];}

无类型函数（不推荐）：void [函数名]([类型名][形参名],…){[语句];}

***注 1**：函数的类型等同于其返回值的类型;*

***注 2**：函数形参名可以与函数外变量（甚至是实参）相同，函数体内部优先表示形参;*

12.2 函数的声明

场合：在程序中函数定义的代码之前调用函数，需要在相关程序的开头声明函数;

格式：[类型名][函数名]([类型名],[类型名],…);

***注**：数组形参声明时可以写全数组形参名和所有参数（如：int a[4][2]）;*

12.3 函数的调用

格式：[函数名]([实参名],…);

场合：赋值、表达式、输入输出语句；

注：函数每次调用都会完整执行里面的代码，所以无论返回值有没有用，函数的代码都执行了；

12.4 函数参数的传递

调用开始：调用函数时断点保护压栈，形参分配存储空间，实参赋给形参；

调用过程：实参无法直接调用，形参是函数处理的变量，全局变量可以直接或间接调用；

调用结束：函数返回一个值，形参空间被释放，里面保存的信息消失；

参数双向传递的情形：

地址结合：无法直接调用的在生命期里的变量可在函数中通过指针进行修改，调用结束不恢复；

数组形参：数组名在形参中即相当于一个指针，二维数组的长度说明了对应行指针变量的长度；

12.5 函数的返回值

一般情况：函数只能返回一个值；

返回多个值的方法：

返回一组类型相同的值：在函数内定义数组保存这些值，返回这个数组对应的行指针或首地址；

返回一组类型不同的值：定义结构体，使函数返回保存这些值的结构体；

12.6 函数存储类型

外部函数：可被项目中其它文件调用的函数，为一般默认情形，在其它文件调用前需要声明；

外部函数的声明：*extern*+一般的声明方式；

内部函数：定义时前缀 *static*，只能在本代码文件中调用；

12.7 函数指针

函数的地址：函数名即代表函数首条语句在代码区中的地址；

函数指针定义：[函数类型](*[指针变量名])([形参类型]…);

函数指针赋值：[指针变量名]=[函数名];

函数指针调用：[指针变量名]([实参名]…);

函数指针意义：在一个函数需要根据不同需求调用不同函数时使用统一形参表示这些函数；

12.8 递归调用

含义：函数直接或间接的自我调用；

两个要素：边界条件（使递归为有限次数）、递归关系（实现自我调用）；

使用原因：实现函数自我调用的需求，或者使比较复杂的循环简单化；

12.9 排序算法

插入排序：*n* 个数的数组 *a* 内从大到小进行排序——

```
#include <stdio.h>
#include <stdlib.h>
int a[];
int iArrange(int iIndex){
    if (iIndex==0) return 0;//边界条件
    else{
        int i=0;
        iArrange(iIndex-1);//先对前面的数排序
        for (;i<iIndex;i++){
            //把更大的数换到前面
            if (*(a+i)<*(a+iIndex)){
                int iTemp=*(a+iIndex);
```

```
                *(a+iIndex)=*(a+i);
                *(a+i)=iTemp;
            }
        }
        return 0;
    }
}
int main(){
    iArrange(数组长度);
    system("pause");
    return 0;
}
```

选择排序：*n* 个数的数组 *a* 内从大到小进行排序——

```
#include <stdio.h>
#include <stdlib.h>
int a[];
int iArrange(int iLen){
    int i,j;
    for (i=0;i<iLen-1;i++){
        //选择数组中第n大的放到第n个
        for (j=i;j<iLen;j++){
            if (*(a+j)>*(a+i)){
                //遇到更大的就换到第n个
                int iTemp=*(a+j);
                *(a+j)=*(a+i);
                *(a+i)=iTemp;
            }
        }
    }
    return 0;
}
int main(){
    iArrange(数组长度);
    system("pause");
    return 0;
}
```

冒泡排序：*n* 个数的数组 *a* 内从大到小进行排序——

```
#include <stdio.h>
#include <stdlib.h>
int a[];
int iArrange(int iLen){
    int i=0,j;
    for (;i<iLen-1;i++){
        //使第i位变成其后所有位的最大值
```

```
            for (j=iLen-2;j>=i;j--){
                //相邻数让大的在前
                if (*(a+j)<*(a+j+1)){
                    int iTemp=*(a+j+1);
                    *(a+j+1)=*(a+j);
                    *(a+j)=iTemp;
                }
            }
        }
        return 0;
    }
    int main(){
        iArrange(数组长度);
        system("pause");
        return 0;
    }
```

十三、指针和数组

13.1 指针变量概念

指针：变量（或代码）存储的（首）地址；

指针变量：存储变量（或代码）指针的变量；

变量长度：64 位为 8B，32 位为 4B；

指针变量两个属性：保存的指针指向的变量、指向的变量类型（基类型）；

指针变量数据类型：基类型*的类型（或 void*）；

空类型：void*，空类型指针可以指向任何一类变量，取具体类型需要强制类型转换；

注 1：指针变量的数据类型既不是基类型，也不是一般化的“指针”类型；

注 2：空类型指针不确定长度无法进行加减整数的偏移运算，但是可以强制类型转换；

13.2 指针变量操作

定义：[基类型]*[变量名];

注：高级指针即连着打多个*，其变量类型也是多个*的高级指针类型；

初始化：定义时可以同时初始化，也可以不初始化，但是不能不赋值直接使用野指针——

方法 1：[基类型]*[变量名]=[现有指针变量];

方法 2：[基类型]*[变量名]=&[所指变量名];

方法 3：[基类型]*[变量名]=([基类型]*)malloc([所需字节数]); ***（注：相当于数组，用完需要 free）***

方法 4：[基类型]*[变量名]=([基类型]*)[具体地址];

赋值和操作：与初始化类似——

方法 1：[变量名]=[现有指针];

注 1：后面可以进行加减法使指针指向偏移，加减法整数自动转换为基类型对应长度；

注 2：现有指针可为指针变量、数组/字符串名（首地址）、函数名（入口地址）等；

方法 2：[变量名]=&[所指变量名];

方法 3：[变量名]=([基类型]*)[具体地址];

方法 4：[变量名]+=[向后移动基类型变量个数整数];

注：+=可以换成-=；

注 1：函数操作指针与操作其它变量无异，包括形参的单向传递等；

注 2：指针形参可以间接访问作用范围外的变量，对这些变量的操作可不因函数结束恢复；

13.3 变量访问方式

直接访问：变量名代表存储空间，在变量作用范围内利用变量名直接访问；

间接访问：&[变量名]取变量地址，*[指针]获得变量存储空间，在变量生命周期内即可间接访问；

注：访问变量都是访问存储空间，变量的值在存储空间中以一定编码方式储存，二者时常混用；

13.4 数组基本操作

数组定义：数组名即数组首个元素的指针——

方法 1：[元素类型][数组名][[数组长度]];

注 1：高维数组定义[类型][数组名][[最高维长度]]…[[最低维长度]];

注 2：数组长度的表示只能是常量，常变量也不行！

注 3：指针类型可以定义数组——[基类型]*[数组名][[数组长度]];

注 4：紧跟在变量名后面有[]，就意味着这个变量名指的是数组；

方法 2：[元素类型]*[数组名]=([元素类型]*)malloc([长度*元素类型对应存储空间大小]);

注 1：这与指针定义无异，用完需要 free，需要检查返回值是否为 NULL；

注 2：本定义左右可以拆开，数组长度不受指针类型和长度的制约；

注 3：malloc 申请的空间大小可以是变量，但是记得乘以元素类型长度；

一维数组初始化：[元素类型][数组名][[数组长度]]={[元素值],[元素值],…};

一维字符数组初始化另一种方式：char[字符串名][[字符串长度]]="[字符串内容]"；

注 1：字符串内容长度应当比字符串长度至少小 1，以便末尾自动补'\0'；

注 2：运用字符串相关的函数和功能均需要末尾存在'\0'作为终止符；

数组不完全初始化时的自动初始化：数组空间中的填充内容为——

定义在静态区：紧着前面的元素初始化，后面所有元素自动补 0（对字符串就是'\0'）；

定义在动态区：紧着前面的元素初始化，后面所有字节补 CC（字符串缺少终止符）；

高维数组初始化：不完全初始化时紧着前面的元素初始化——

按各个维定义：[元素类型][数组名][[数组长度]]={{[元素值],…},{[元素值],…},…};

按一维数组定义：[元素类型][数组名][[数组长度]]={[元素值],[元素值],…};

注：此时紧着靠前的数组元素初始化；

注 1：高维数组存储时即为按最小级别元素顺序排列成的一维数组的顺序存储；

注 2：高维数组和一维数组的区别在于调用的方式和相关指针的类型；

数组的使用：除了初始化之外均只能分元素而不能整体使用（字符串有特殊使用方式）；

数组作为函数参数：当作指针常量理解；

13.5 数组退化为指针

一维数组：数组名为指针常量，其值为首元素地址，其类型即基类型为其元素类型的指针类型；

高维数组：看作元素为低一维数组的一维数组，x 维数组名需要 x 个*才能得到首元素的值；

字符串常量：存储在代码区，首位的地址即其指针，其内容不可以被修改；

注 1：二维数组名的指针类型是行指针，取值一次得到"行"，即该行首元素的地址；

注 2：[]运算符可以看作指针加了内部的整数之后进行了一次取值；

13.6 一维数组元素值的表示

注：数组的位次序号都是从 0 开始的！

下标法：[数组名][[位次序号]];

指针法：*([数组名]+[位次序号]); ***注：此时位次序号自动转换为了数组名对应的指针变量类型；***

13.7 二维数组元素值的表示

下标法：[数组名][[行序号]][[行内位次序号]];

指针法：*(*([数组名]+[行序号])+[位次序号]);

混合法：*([数组名][[行序号]]+[位次序号]);

13.8 空指针

概念：有类型但不指向任何地址的指针；

定义：[基类型]*[指针名]=NULL; ***注：头文件已有#define NULL 0;***

注：空指针不是野指针！

13.9 指针变量定义的相关模糊表述一览

注1：紧跟[]的名称就是数组名（即变成加*的类型）;

注2：末尾为([形参])的代表函数;

注3：*裸露在外面即指针类型或其组成的数组;

[基类型]*p：基类型指针变量 p；

[基类型]p**：基类型二级指针变量 p；

[基类型]*p[n]：基类型指针变量组成的指针数组 p，长度为 n；

[基类型](*p)[n]：基类型行指针变量 p，+1 代表前进 n*基类型长度个字节，二维数组名为此类型；

[基类型]*(*p)[n]：基类型长 n 的行指针变量的指针类型 p（一种二级指针）；

[基类型]*p[n][m]：+1 代表前进 n*m*基类型长度个字节，三维基类型数组名即为此类型；

[基类型](*p[n])[m]：基类型长 n 的行指针变量数组；

[基类型]*p([形参])：返回值为指针的函数 p；

[基类型](*p)([形参])：函数指针 p；

[基类型](*p[n])([形参])：长度为 n 的函数指针数组 p；

[基类型]*(*p)([形参])：返回基类型指针变量类型的函数指针 p；

[基类型]p([形参])**：返回值为二级指针的函数 p；

十四、结构体

注：结构体变量用在函数、数组中与一般的基本类型变量区别不大;

14.1 结构体类型的申明、变量定义和初始化方法

注：结构体类型申明的末尾无论是什么都有分号！

做法1：最标准但最繁琐的做法（可以不进行初始化）——

struct [结构体类型名]{[分量类型][分量名];…;};

struct [结构体类型名][结构体变量名]={[分量值],[分量值],…};

做法2：联合申明和变量定义（变量定义同时可以初始化）——

struct [结构体类型名]{[分量类型][分量名];…;}[变量名],[变量名],…;

做法3：只使用一次的结构体类型（变量定义同时可以初始化）——

struct{[分量类型][分量名];…;}[变量名],[变量名],…;

做法4：可以在申明完成后使用时省略 struct（但是申明过程中用结构体指针仍然需要）——

typedef struct [结构体类型名]{[分量类型][分量名];…;}[结构体类型名];

[结构体类型名][结构体变量名]={[分量值],[分量值],…};

14.2 结构体类型变量存储规则

注：规则比较抽象，反正比所有分量总和一般来说是大的;

规则一：第一个成员首地址为结构体变量首地址；

规则二：其它成员的首地址都较结构体首地址向后偏移了对齐数的整数倍个字节；

注：对齐数=min{成员类型存储空间大小,系统默认值（vs2012 为 8）};

规则三：结构体总存储空间大小要求为最大对齐数的整数倍（不行就扩充）；

规则四：嵌套结构体时，嵌套的结构体对齐到自己最大对齐数的整数倍；

14.3 结构体变量的引用

作为整体引用：仅限于取地址、为同类型结构体变量赋值等少数情况；

变量名分量引用：[结构体变量名].[成员名];

指针指向分量引用：[指向结构体变量的指针]->[成员名];

14.4 结构体的意义

函数返回多维参数：定义返回值为结构体的函数可以等效于返回多个长度不一定相同的值；

方便排序：方便根据某类量的某个性质的顺序进行对这类量的所有性质的排序；

性质紧密相关：可以使不可分割的不同性质真正地“不分割”了；

14.5 链表

概念：一个结构体，其分量包含指向同类另一个/一些结构体的指针；

含义：通过结构体中的指针将不同的结构体串成一条链，较数组处理起来更加自由；

操作：通过分配和释放空间、改变结构体中指针的指向进行链表操作，视情况具体而定；

建立：辅助定义三个指针（首指针、尾指针、新变量指针）；

十五、其它自定义变量类型

15.1 枚举类型

类型说明：*enum*[类型标识符]{[元素]=[整数],[元素],…};

注：每个元素后都可以赋一个整数值，如果没有赋值自动定义为上一个元素的整数值加一；

变量定义和初始化：可以不同时进行初始化——

方法 1：*enum*[类型说明符][变量名]=[元素];

方法 2：*enum*[类型说明符][变量名]=(*enum*[类型说明符])[整数];

注：enum 类型的元素都是实打实的值，不能够说明后再对其赋值！

枚举型变量可以进行的操作：计算、赋值、自增自减（不推荐）、判断比较；

枚举型变量不可以进行的操作：用枚举元素输入输出；

意义：增强可读性；

15.2 定义类型别名

定义方式：*typedef*[已有类型名][自定义类型名];

使用：在作用范围（大括号内或全局）可以直接用自定义类型名替换已有类型名；

意义：增强可读性、区分自己定义的变量类型和已有类型；