1. 基本类型的赋值，转换问题。[（1）见 p80；]

（1） c 中是不是也和 Java 一样，存在隐式转换和强制转换？有区别吗？

有区别。

C 中的隐式转换就是“整型提升”。C 中的“整型提升”仅指：表达式中的操作数类型<=int 的情况下（short 和 char），提升到 int 型。

（注意：是表达式中的 char 和 short 在使用之前被转换为 int。）

>=int 型的类型的提升，被称为寻常算术转换：

寻常转换应该遵循：int – unsigned int – long int – unsigned long int – float – double – long double 由小到大的转换规则：一个操作数类型相对另一个操作数类型排名较低（较小），则自动转换为相对较大的类型。（数据转换按数据存储长度增长的方向进行。）

寻常算术转换和强制转换均为“算术转换”。寻常算术转换侧重于自动转换到相对较高的类型（如 int - float），强制转换则可人为将其转换到其他类型（如 long – int；int - double）。

（记住：是先转换，再执行操作。）

（记住：若某个操作符的各个操作数属于不同的类型，则除非其中一个操作数转换为另一个操作数类型，否则操作无法进行。问：该规则是否也针对赋值操作符“=”？不过等号右边的操作数（右值）自动转换成了左值的类型。）

（注意：以上概念是针对算术操作符而言的。算术操作符就是+ - * / %。

问：那对于其它操作符呢？尤其是关系操作符？）

（2） 不同类型操作数互相赋值时的精度变化情况：

从 int 到 float：精度可能会降低（有点例外）。Float 只能保证 6 位有效数字的精度（浮点型以指数方式存储），虽然长度为 4 个字节。

在 32 位平台上，通常 int 是 4 字节长度，最多表示到 21 亿多，而 int 型是“绝对精确”的，换句话说，就是 int 行最多可以保证 10 位十进制有效数字的精确度。而 float 只能保证 6 位有效数字的精确度，因此 int 到 float 的转换是可能丢失精度的。

比如整数“1234567899”转换成 float 后，大约是：1.23457936 乘 10 的 9 次方，也就是从第 7 位有效数字开始已经不准确了。double 可以保证 15 位 10 进制有效数字的精度，所以从 int 到 double 不会有这个警告。

（以上摘录自网络。）

（3） 左值与右值的注意点？

左值可以是变量或表达式，但必须能标识一个可以存储结果值的地点。

2. 位运算问题：

（1） 无符号数与有符号数左移，右移（<<,>>）的异同？

它们的左移均为逻辑移位（补 0）；无符号数的右移也是逻辑移位(因为无符号数无须担心符号位问题)；但有符号数的右移方式（逻辑移位还是算术移位）取决于编译器的类型（因而是不可移植的）。

（2） 逻辑移位与算术移位的区别？

逻辑移位补 0；算术移位（似乎只用于右移）：左边移入的位根据有符号数符号位值决定是 0 还是 1。

3. 各种数据类型的长度及注意点

（1） 整型：（长整型至少应该和整型一样长，整型至少应该和短整型一样长。）

char， signed char， unsigned char， short int， unsigned short int，int， unsigned int， long int， unsigned long int

其中，short int 至少 16 位，

limits.h 中说明了各种不同整型类型的特点： p30

	signed		unsigned
类型	最小值	最大值	最大值
字符	SCHAR_MIN (-128)	SCHAR_MAX(127)	UCHAR_MAX(0xff)
短整型	SHRT_MIN(-32768)	SHRT_MAX(32767)	USHRT_MAX(0xffff)
整型	INT_MIN (-2147483647 - 1)	INT_MAX (2147483647)	UINT_MAX (0xffffffff)
长整型	LONG_MIN (-2147483647L - 1)	LONG_MAX (2147483647L)	ULONG_MAX (0xffffffffUL)

整型字面值（整型字面值常量）：可以是 9 种整型中的任何一种。

（2） 浮点类型：long double 至少和 double 一样长，而 double 至少和 float 一样长。

头文件 float.h 中有记录：

	MAX	MIN
float	FLT_MAX (3.402823466e+38F)	FLT_MIN(1.175494351e-38F)
double	DBL_MAX (1.7976931348623158e+308)	DBL_MIN (2.2250738585072014e-308)
long double	LDBL_MAX (DBL_MAX) 或 (1.189731495357231765e+4932L)	LDBL_MIN (DBL_MIN)或 (3.362103143112093506 3e-4932L)

注：浮点数字面值总是写成 10 进制的形式，必须有一个小数点或一个指数。

（3）指针

4. static ,extern ,const

5. C 中的抽象数据类型（ADT：abstract data type）是什么？

也称“黑盒”。由接口和实现 2 部分组成。接口是公有的，一般定义在头文件中，说明用户如何使用 ADT 提供的功能；实现是私有的，是实际执行任务的部分（实现细节对客户不可见）。比如：可将常量或函数等的申明放在头文件中，函数的定义放在源文件中。而这些函数还可以调用更为细节的且被定义为 static 的处理函数（static 函数对外不可见）。具体可见例子 p125。

6. 关于提高程序效率的一些建议：

（1） 尽量使用编译时求值的表达式（如常量表达式），而减少使用运行时求值的表达式（代价更高）。

（2） 在 for 循环里减少计数器的使用（可能的话），如下：（p148）

```
#define SIZE    50
int    x[SIZE];
int    y[SIZE];
void try()
{
        register int    *p1, *p2;
        for( p1 = x, p2 = y; p1 < &x[SIZE]; )
                *p1++ = *p2++;
}
```

（3） 数组操作时：使用指针往往比使用下标更有效率（但不一定）；指针的效率永远不会低于下标（理论上）。

（4） 声明为寄存器变量的指针通常比位于静态内存和堆栈中的指针效率更高。

（5） 自动变量（尤其是数组）如果在函数或代码块中经常要被初始化，可以考虑将其设为 static，这样只需在程序开始前初始化一次。

（6） 结构中成员之间的边界对齐问题：p206

结构的起始位置必须是结构中边界要求最严格的数据类型所要求的位置，如下：

struct ALIGN{char a; int b; char c;};

sizeof(ALIGN) 则显示要占 12 字节。因为 int 型的存储位置必须能被 4 整除，结构起始位置则也一样。所以 3 个成员将各占 4 个字节。

如果改为这样：

struct ALIGN{int b; char a; char c;};

则只占 8 字节（其中 2 个字符紧挨在一起）。

（必要时需对结构中成员的排序进行优化。）

单个字节(char)能对齐到任意地址

2 字节(short)以 2 字节边界对齐

4 字节(int, long)以 4 字节边界对齐

sizeof 得到结构的整体长度，包括因边界对齐而跳过的字节。

宏 offset（定义在 stddef.h 中）确定结构中某个成员的实际位置。

如：offsetof（struct ALIGN， b）

7. 运算符优先级中一些注意点,一些比较重要的优先级：

（1） * 与 ++/-- 的结合

从高到低： . -> ++(后缀) --(后缀) ！++(前缀) --(前缀) *（间接访问） &（取址）

8. 指针，数组中的注意点（多重指针，指针数组，多维数组）

（1） 数组名是一个指向某某型的常量指针（是常量!），它的值是第一元素的地址。但有两个例外：被用作 sizeof 或&的操作数。前者返回整个数组的长度；后者产生一个指向数组的指针（不是指向某某型的常量指针）。P141。

（2） 以下 3 者等价：

array[2]; 2[array]; *(array + 2);

（3） int a[5]; 和 int *b; 的区别： p150

声明数组,则同时根据指定的元素数量分配了内存空间，并以作为常量指针的数组

名指向这段内存空间的起始位置。

声明指针变量，编译器只为指针本身保留内存空间，不为任何整型值分配内存空间。所以：此时*a 是合法的，*b 是非法的。但 b++能编译，a++却不能（常量）。

做测试：char *a = “abc”; char b[] = “”abc”; char *c = a; 各自的++。

（4） 数组声明时的初始化：

int a[5] = {2,5,1}; 则后 2 位自动设 0。

Int a[ ] = {1,2,3,4,5};则长度自动设为 5。

（5） char a[ ] = “”Hello”;

char *b = “Hello”;

两者效果一样，但含义不同。 P153。

（6） 2 维数组一定要理解为数组中包含子数组的形式。

理解：p157 (想像一下指针数组，但并不一样)

int a[3][10]; 其中：

a （指向子数组的指针，好比指向指针的指针）

a + 1 （指向第 2 个子数组的指针）

*(a + 1) （指向子数组中第 1 个整型值的指针(仍是指针！)）

*(a + 1) + 5 （指向子数组第 6 个整型值的指针）

((a + 1) + 5) （子数组第 6 个整型量的值）

以下等价：

*(a + 1) 和 a[1]

*(a + 1) + 5 和 a[1][5]

（7） c 中若写 a[3, 4] ， 则等同于 a[3] 。 p158

9. C 中函数的参数都是按值传递的（拷贝形式，无须担心值被改，但数组会被改）。

包括指针！将实参的指针的拷贝赋给形参，形参可直接修改所指内存地址中的值，但指针是以拷贝的形式传递的，所以还是按值传递的。好处 是并未破坏 原来的指针。

所以：值传递易产生“切割问题”。

如下：

```
Class Window {
Public:
    Virtual void display() const;
};
Class WindowWithScrollBars: public Window{
```

```
Public:
    Virtual void display() const;
};
Void printNameAndDisplay(Window w){
    w.display();
}
WindowWithScrollBars wwsb;
printNameAndDisplay(wwsb);
```

因为参数仅仅是拷贝，所以不可能把 WindowWithScrollBars 中不属于 Window 的部分复制给 Window 对象。所以调用的是 WindowWithScrollBars 的 display(),哪怕它是虚函数。所以要采用 by reference-to-const 的方式，当然，这仅指对象，若是内置类型入 int，还是用 pass by value 较好（见《Effective c++》 p88）

10. C 中常用的字符，字符串处理函数：

（注：size_t 定义在 stddef.h 中，代表无符号整数。）

关于查找：

（1） size_t strspn(char const *str, char const *group); p182

返回 str 起始部分匹配 group 中任意字符的字符数（也可理解为 str 中第一个不在 group 中出现的字符的索引值）。

如下：int len1 = strspn(“1234567890”, “af123af”); //len1 = 3

（2） size_t strcspn(char const *str, char const *group);

返回 str 起始部分不匹配 group 中任意字符的字符数（str 中第一个在 group 中出现的字符的索引值）。

如下： int n = strcspn(“Hello, world”, “Welcome you”); //n = 1

（注： 若找不到，则返回 str 的尾部。）

（3） char *strpbrk(char const *str, char const *group);

返回 str 中第一个与 group 中任何一个字符相匹配的字符的地址（指针）。

（注：若没找到，返回 NULL。）

（注： 它与 strcspn 的区别是，前者返回指针，后者返回下标值。）

（4） char *strctr(char const *str, int ch);

（5） char *strrchr(char const *str, int ch);

前者返回在 str 中第一次出现的字符 ch（int 型）的位置，后者返回在 str 中最后一次出现的 ch 的位置。

（6） char *strstr (char const *s1, char const *s2);

在 s1 中查找整个子字符串 s2 第一次出现的起始位置。

（注： 若找不到，返回 NULL ，若 s2 是空字符串，返回 s1。）

关于长度：

（7） size_t strlen(char const *string);

返回 string 包含的字符的个数。

（注：不包括\0 的长度。写 int a = strlen(“ABC”);或 int a = strlen(“ABC\0”); 结果都为 3。）

关于复制与连接：

（8） char *strcpy(char *dst, char const *str);

将 src 字符串复制到 dst 中（连带 NUL 字节）。返回指向 dst 数组的指针。原来 dst 中的内容被覆盖。

（注： 若 src 和 dst 在内存中是重叠的，则结果是未定义的。）

（注： 不要让 str 的长度大于 dst 的长度。）

（9） char *strcat (char *dst, char const *src);

将 src 添加到 dst 的末尾，返回指向 dst 数组的指针。

（注： 若重叠，未定义。）

（注： 要保证 dst 剩余空间足够。）

关于比较：

（10） int strcmp (char const *s1, char const *s2);

字典比较。比较 2 个字符串中最先不匹配的字符的大小（依据字符集）。

若 s1 < s2, 返回小于 0 的值；若 s1 > s2, 返回大于 0 的值；

若 s1 = s2,返回 0。

长度受限的字符串函数：

（11） char *strncpy(char *dst, char const *str, size_t len);

（12） char *strncat(char *dst, char const *str, size_t len);

（13） int strncmp(char const *s1, char const *s2, size_t len);

功能类似于前 3 者，但有一些区别和注意点。见 p179

11．无符号数运算的一些注意点：

（1） 比较大小时：

if (strlen(x) >= strlen(y)) … //合理

if (strlen(x) – strlen(y) >= 0) … //不合理

原因：strlen 返回 size_t,是无符号整型，相减结果还是无符号（表达式结果保持了无符号类型），所以永远>=0。除非把 strlen 的结果强制转换为 int。

12．指针，内存操作中的一些其它注意点：

（1） 如下例子：

int *a = 0; 表达式*a 等于多少？a 是一个 NULL 指针，对它解引用是错误的。但有些环境不会在运行时捕捉到这个错误，而是去访问内存位置 0 的内容，这是一个隐患。所以，在解引用之前必须先对指针进行有效性检查。

内存操作函数：

void *memcpy(void *dst, void const *src, size_t length);

void *memmove(void *dst, void const *src, size_t length);

void *memcmp(void const *a, void const *b, size_t length);

void *memchr(void const *a, int ch, size_t length);

void *memset(void *a, int ch, size_t length);

以上函数直接对内存操作，以字节数为单位。其中，memcpy 能处理任意字节的序列，比如将一段内存中的序列读入一个结构体中；而 strcpy()只处理字符序列，且遇 NUL 字节即结束。

memmove 与 memcpy 的唯一区别是：memmove 允许源和目标操作数重叠，memcpy 不允许。

memchr 从 a 的起始位置开始查找字符 ch 第一次出现的位置。

动态分配内存（stdlib.h 中）：

void *malloc(size_t size); 分配失败返回 NULL，所以要作判断。

void free(void *pointer); pointer 是 malloc,calloc,realloc 返回值，也可为 NULL。动态分配的内存必须整块一起释放。

void *calloc(sizes_t num_elements, size_t element_size); 与 malloc 区别在于：calloc 将分配的内存初始化为 0。

void realloc(void *ptr, size_t new_size); 修改一个原先已经分配好的内存块的大小（长则

不足，短则切割，若原先内存块无法改变大小，则分配另一块内存并返回指向它的指针，所以旧的指针就不能再用了)。若 ptr 为 NULL，则和 malloc 无异了。返回类型为 void*，可以赋给任意类型的指针。

13．#define 宏与函数之间的优劣：p283

宏的执行速度比函数快得多，函数需要调用、返回等操作。函数只能对特定的类型操作，而宏是类型无关的，宏还可以实现一些函数无法实现的操作。但宏需要将所有代码拷贝到调用程序中，增加了代码长度。

所以：宏比较适合执行简单的计算，如求 2 个值中的较大值。　如果宏代码比较长，并且频繁被调用，还是声明为函数比较好。

14．预定义符号（由预处理器定义的符号）　p279

有_FILE_，　_LINE_ , _DATE_ , _TIME_ , _STDC_

15．注意宏使用中的一些细节　　p283

（1）　如下:

```
#define PRINT(FORMAT, VALUE)                \
        printf("The value of " #VALUE);     \
        "is" FORMAT "\n", VALUE )

Print("%d", x + 3);
```

输出为：　The value of x + 3 is 25

printf() 函数的第一个参数必须为字符串，所以比必须在 VALUE 前面加上#。#VALUE 的目的就是将宏参数 VALUE 转换为字符串形式。

（2 ）　##的作用：

把两边的符号连接成一个符号。如下：

```
#define ADD_TO_SUM( sum_number, value )     \
        sum ## sum_number += value

ADD_TO_SUM( 5, 25 );                        // sum5 += 25;
```

（3）宏定义中的参数往往要打上不止一层的括号。

16. 一些宏命令：

（1） #undef name 将一个宏定义移除（比如一个现存的名字需要被重新定义，那旧定义必须首先被移除）。P285

（2） 条件编译

```
#if  constant-expression
        statements
#elif  constant-expression
        otherstatements
#else
        otherstatements
#endif
```

（3） 是否被定义： p287

```
#if     defined(symbol)
#ifdef  symbol

#if     !defined(symbol)
#ifndef  symbol
```

（4） #error 用来生成错误指令。 P291

（5） #line

（6） #progma

17. 命令行定义 p285

在命令行中定义符号，用来启动编译过程。如根据同一个源文件编译一个程序的不同版本。如：编译时决定数组的长度。

定义数组如下：int array[ARRAY_SIZE];

UNIX 环境中用-D 选项如下： cc -D ARRAY_SIZE = 100 prog.c；

18. 本地形式#include 和函数库形式#include 的区别： p289

函数库形式的#include 直接在由编译器定义的“一系列标准位置”查找函数库头文件。

本地形式#include 现在源文件所在当前目录查找，若未找到，就到标准位置去查找。

19. 防止一个头文件在一个源文件中被重复包含，可如下组织头文件： p291

```
#ifndef  AAA
#define  AAA  1
/*
**
*/
#endif
```

（在比较复杂的系统中，每一个头文件都要这么写。）

20. 函数指针 p258

```
int  *f ( );          f 是一个函数，返回整型指针。
int  (*f)( );         f 是一个函数指针，它所指向的函数返回一个整数值。
int  *( *f )( );      f 是一个函数指针，它所指向的函数返回一个整型指针。
int  f( )[ ];  （错误！）f 是一个函数，它返回一个整型数组，但 c 的函数返回值只能是标
                      量，不能是数组！
int  f[ ]( );  （错误！）f 是一个数组，其中的元素是“返回值为整型的函数”，但数组中的
                      元素的长度必须是 相同的，而各个函数的长度未必相同。
int  ( *f[ ] )( );    f 是一个数组，数组元素的类型是函数指针，它所指向的函数的返
                      回值是一个整型值。
int  *( *f[ ] )( );   f 是一个数组，数组元素的类型是函数指针，它所指向的函数的返
                      回值是一个整型指针。
```

（以上为旧式风格的函数声明，应该使用完整的函数说明，如下：）

```
int  ( *f ) ( int, float );
```

21. C 与 c++中的 const

第一，C 和 C++标准都规定了 const 描述的变量为只读，对 const 变量进行更改的行为是未定义的，任何结果都是可能的。

对于 const 变量行为，各个编译器都有自己的理解，在优化的时候极有可能被优化为常量。这也是为什么很多人推荐使用 const 变量代替#define 常量－－既可以被优化成常量，同时还能让编译器进行类型检查。

第二，对于函数参数而言，压栈都是从右往左的。这应该在标准里提到，我一时没找到。但各个参数的计算顺序是未定义的，由各编译器自己发挥。可能会是一边计算一边入栈，也可能是全部计算完再入栈。

常量肯定是只读的，例如 5， “abc”，等，肯定是只读的，因为程序中根本没有地方存放它的值，当然也就不能够去修改它。而“只读变量”则是在内存中开辟一个地方来存放它

的值，只不过这个值由编译器限定不允许被修改。
（以上摘自网络）
但 c 和 c++中的 const 也有不同。

21(b). C 与 C++中的 const 的区别
首先，const 是由 C++采用并加进标准 C 中的。
区别一：在 C 中，编译器对待 const 如同变量一样（在内存中分配空间），只不过带有一个特殊的标记，意思是“不要改变我”。所以，在 C 中不可以用 const 变量作为数组维数。而在 C++中，const 变量在编译时刻就常量化了。
区别二：在 C 中，const 变量的连接属性是外部的，而 C++中，其属性是内部的。
（以上摘自网络）0。
将一个函数声明为 const，则函数不能改变成员变量。

22. 变量初始化（是初始化）的两种方式：1.在声明时直接初始化；2.在构造函数中初始化阶段初始化。

二：关于程序结构的建议：
1. extern 类型的（默认的，全部可见）的变量或函数，可以在一个.h 文件中用 extern 声明它在其它文件中已定义过了，在其他需要用到它的地方再把这个.h 文件导入进去就可以直接引用了。

2. 通常将类声明放在.hpp 文件中（方便客户使用），将方法定义放在.cpp 文件中。

调查重点：1 函数指针 2 数据类型 3 符号优先级 4auto/static/const 等 5internai/external 6enum 常量的引用 7c/c++字符串操作注意点 8 类型转换注意点 9c/c++区别