```

//C语言
1、 long 、 unsigned long 和 pointer指针类型 在32位中是4个字节，在64位中是8个字节
2、不使用 sizeof 求数据类型的字节大小：#define mysizeof(value) (char*)(&value + 1) - (char*)(&value)
3、 位与    &       两个位都为1时，结果才为1
    位或    |       两个位都为0时，结果才为0
    位异或  ^       两个位相同为0，相异为1
    位非    ~       0变1，1变0
4、交换a与b的值
    a = a ^ b;
    b = a ^ b;
    a = a ^ b;
    可以合并为：a ^= b ^= a ^= b;
5、如果数组不作初始化，那么各个元素的值就不是0了，所有元素都是垃圾值。
6、将数组进行传参时，数组将会退化成指针，指向数组的首地址。
7、
#include <stdarg.h>
int sum(int num, ...)           // 变参函数定义，第一个参数是后续参数的数量
{
    va_list args;               // 即char* args。 args是用于指向可变参数首地址的指针。
    va_start(args, num);        // 初始化args变量，将其指向可变参数地址列表的首地址，num是最后一个固定参数
    int sum = 0;
    for (int i = 0; i < num; i++)
        sum += va_arg(args, int); // 获取下一个参数（int类型）并加到sum上
    va_end(args);               //清空va_list可变参数列表
    return sum;
}
#define va_copy(destination, source)//拷贝va_list的内容从src到dst
8、
int* p(char);       指针函数，是一个返回值为 int 类型指针的函数，p是函数名
int (*p)(char);     函数指针，是一个指向参数为 char 类型、返回值为 int 类型的指针
int (*p[5])(char);  函数指针数组，是一个存放5个参数为 char 类型、返回值为 int 类型的指针的数组
9、p->a 等价于(*p).a
10、递归函数
int Factorial(int n)    //阶乘
{
    if (n == 1)
        return 1;
    return n * Factorial(n - 1);
}
int Fibonacci(int n)    //斐波那契
{
    if (n == 2 || n == 1)
        return 1;
    return Fibonacci(n - 1) + Fibonacci(n - 2);
}
11、格式控制符："%u"十进制无符号、"%x"十六进制无符号、"%ld(u)"16位十进制有(无)符号、"lld(u)"32位十进制有(无)符号
"%f"小数、"%p"指针地址、"%s"字符串（输入为指针，无需解引用）
12、字节对齐：(class)(struct)(union)按4或8个字节看CPU位数。在数据成员完成各自对齐之后，其本身也要进行对齐
13、 联合体判断大小端序
union {
    int i;
    char c;
} u;
u.i = 1;        // 在大多数系统中，int型为4字节，这里赋值为1（即0x00000001）
printf("%d", u.c);  // 输出：1（x86、x64、arm都是小端序）
                    // 小端序u.c存储int的最低有效字节，即0x01.大端序u.c存储int的最高有效字节，即0x00
14、不使用if,>,<比较两数大小（有符号 32 位整数）
```

```
#define MAX(a, b) ((a) - (((a) - (b)) & (((a) - (b)) >> 31)))   // 返回较大的数
#define MIN(a, b) ((b) + (((a) - (b)) & (((a) - (b)) >> 31)))   // 返回较小的数
#define IS_EQUAL(a, b) (!((a) ^ (b)))                           // 判断两数是否相等，相等返回1
```

15、""编译器会先在当前代码文件所在目录中查找指定头文件。若没有找到，会继续按照和 <> 相同的方式在系统目录中查找。

16、Linux的内存分布：栈区、映射区（存储动态链接库以及调用mmap函数的文件映射）、
堆区、全局(静态)区（未初始化.bss和已初始化.data）、常量区(.rodata)、代码区(.text)

17、堆区管理

```
malloc：申请size字节的堆空间                              //分配一块未初始化的内存，需要手动初始化
calloc：申请n * memb * size字节的堆空间，返回一个数组指针。  //会将分配的内存初始化为零
realloc：重新申请一块size字节的堆空间                       //需要手动初始化
free：释放堆空间                                          //需要手动指向NULL
```

18、不可访问区域：0x00000000，也即NULL指针指向的地址

19、内存溢出：指应用程序使用的内存过多，超出系统能提供的最大内存

20、内存泄漏：动态分配的内存使用后没有释放，或者指向了其它内存。处理：及时释放、智能指针管理。

21、预处理（#预处理指令处理）、编译（语法分析和优化）、汇编（汇编码>>机器码）、
链接（合并所有目标文件.o/.obj和库文件生成可执行文件 / 库文件）

22、 volatile 内存关键字，每次访问变量时都必须从内存读取最新的值，而不是使用寄存器中缓存的旧值

23、 register 寄存器关键字，将高频访问的变量存储在寄存器中（即CPU），以提高访问效率

24、 extern 关键字，声明全局变量和函数，说明该变量或函数是在其他文件中定义，可以使用。
extern "C":在c++中调用c语言代码，告诉编译器这部分代码要是有C编译，避免函数名被 C++ 修饰（编译器导入导出符号不一致）。

```
//C++
```

1、 new 底层实现：调用malloc、调用构造函数、返回对象指针

2、引用的本质就是一个自动解引用的常量指针

3、 static 修饰

局部变量：该变量只会初始化一次且函数结束不会被释放
全局变量或函数：该变量或函数只能在当前文件中使用
成员变量和函数：该成员为类所有对象共有的

4、 const 修饰

变量：变量值不能改变
成员函数：整个函数中不可发生数据修改。

```
const int*          //指针的指向不可修改
int* const          //指针指向的内存数据不可修改
const int* const//都不可修改
```

5、 explicit 用于修饰构造函数，防止隐式类型转换

6、 inline 修饰的函数在编译阶段展开，减少函数频繁调用的开销，提升性能

7、多态

静态多态：模版，函数和运算符重载。编译阶段确定对象。
动态多态：虚函数和纯虚函数重写。运行阶段确定对象。

8、 virtual 虚继承用于解决菱形继承问题。

虚继承时，编译器会在派生类的内存布局中添加一个指向虚基表的指针，即虚基指针，指向虚基表。
虚基表中存放的是虚基类指针以及虚基类与本类的偏移地址，以及虚基类到共有基类元素之间的偏移量。通过偏移地址可以找到虚基类成员。
也就是说，他们的虚基指针最终指向同一个虚基类，当他们进行派生时，虚基指针会被继承，因此也是指向那个虚基类。

9、 virtual 修饰函数（成为虚函数），用于基类指针访问派生类的函数。

Base * obj = new Derived(); 被执行时，obj 的虚指针指向 Derived 类的虚表，而不是 Base 类的虚表。
当你调用 obj->func(); 时，程序会通过 obj 的虚指针查找实际对象的虚表，从而找到并调用 Derived::func()。

10、构造函数不能为虚函数，在对象构造的过程中，虚表并未完全初始化，虚函数在构造期间的调用可能会导致未定义的行为或程序崩溃。

11、 dynamic_cast 用于基类向派生类的强制转换，基类必须有虚函数，以支持 dynamic_cast 能够进行运行时类型信息(RTTI)检查

12、预处理器：

#define中 # 运算符会把参数转换为用引号引起来的字符串，## 运算符用于连接两个参数。
__LINE__ 当前行号， __FILE__ 当前文件名， __DATE__ 当前日期， __TIME__ 当前时间

13、strlen 是函数，只能计算字符串的大小，不包含'\0'。 sizeof 是关键字，可以计算任何数据的内存大小，包含'\0'。

14、定义一个常数，用以表面一年有多少秒。 `#define SECOND_PER_YEAR(60 * 60 * 24 * 365)UL`
UL表示长整型数

15、使用宏求结构体的内存偏移地址。 #define OffSet(type, field) ((size_t) & (((type*)0)->field))
16、enable_shared_from_this 在对象方法中创建指向自身的shared指针
17、NULL在C语言中是(void*)0， void 类型的指针都可以被转化为其他任何指针类型。而在C++中是0，c++不支持隐式类型转换，因而引入 nullptr 。
18、 override 关键字：表示该函数是一个重写函数。
19、 final 关键字：在虚函数后面加上 final 表明该虚函数不允许被重写。在类定义的类名后面加上 final 表明该类不允许被继承。
20、 default 关键字：用于使编译器自动生成相应的默认函数实现。（构造函数、析构函数、拷贝构造函数、拷贝赋值运算符和移动构造函数）
21、 delete 关键字，用于删除类的特殊成员函数或禁用某个函数。
22、 decltype 关键字，可以用于求表达式的类型。例如：decltype(10.8) x = 5.5; //x 被推导成了 double
23、C++17主要新特性： if 和 switch 的初始化、字符串视图（只读且避免不必要的字符串复制）、文件系统（文件操作，fstream用于读写）

//网络协议
1、OSI七层模型：物理层、数据链路层、网络层、传输层、会话层、表示层、应用层
2、TCP / IP模型：网络接口层（物理层 + 链路层）、网络层、传输层、应用层
3、TCP、UDP区别：面向连接、自动补包、字节流、保证顺序、不支持组播广播
4、Socket
TCP服务器：socket、bind、listen、accept、read 客户端：socket、connect、write
UDP服务器：socket、bind、recvfrom 客户端：socket、sendto
5、发送广播：socket、getsockopt、setsockopt、sendto
6、接收组播：socket、setsockopt、bind、recvfrom

//Qt
1、信号与槽：依赖Q_Object宏。connect第五个参数（自动连接、直接连接、队列连接、阻塞队列连接）
2、事件：操作系统事件 >> 转换为Qt事件 >> 事件派发 >> 事件接收

//音视频
1、音频常用参数：采样频率、量化位数、通道数。
2、WAV格式 = PCM裸流 + 44字节文件头
3、FFmpeg音频：Packed格式为交错存储（AVFrame::data[0]: LRLRLRLR），Planar格式为分开存储（AVFrame::data[0] : LLLL; data[1] : RRRR）
4、音频参数推导
采样率48000：代表一秒钟48000个样本数据；双通道：代表每个通道都是以48000Hz采样的
【如果】采样精度(位深)16bit
【可得到】比特率(每秒数据量) = 采样率＊通道数＊采样精度 = 48000＊2＊16 = 1536000bit 即 192000 字节
【如果】编码格式为aac，则每帧样本数1024：由编码协议规定（mp3为1152）
【可得到】帧率 = 采样率 / 每帧样本数 = 48000 / 1024 = 46.875Hz ； 每帧时长 = 1000ms / 46.875 = 21.333ms
5、MP3是有损压缩，主要去除高频部分声音
6、1080p:每帧都是完整的 1080i:隔行扫描，第一帧奇数行，第二帧偶数行，第三帧奇数行...
7、YUV420帧数据大小 = RGB24帧数据大小 / 2 = 宽 * 高 * 1.5;
8、YUV420指：色度分量在水平和垂直方向上都进行2 : 1的下采样。每四个Y分量共用一个U分量和一个V分量
9、BMP图像 = 14字节位图文件头 + 40字节位图信息头 + RGB帧数据
10、H264编码原理：利用 空间冗余、时间冗余、编码冗余、视觉冗余、知识冗余、结构冗余 去除冗余数据，属于有损压缩
    先验知识：
    人眼对亮度分量的敏感度高于色度分量，轻微的色度变化对视觉体验影响很小
    人眼对低频分量的敏感度高于高频分量，轻微的高频变化对视觉体验影响很小
11、编码流程：预测、变换、量化、扫描、熵编码->码流
12、五层结构：图像组(GOP)、帧(Frame)、【NALU】、片(Slice)、宏块(Macroblock)、子块(subblock)、【像素】。
13、H264比特流 = 起始码 + NALU + 起始码 + NALU
0x000000 结束码
0x000001 起始码
0x000002 保留使用
0x000003 防止竞争
14、打包格式：Annex - B格式适合传输流，AVCC适合MP4、FLV、MKV格式
15、FLV = FLV Header（9字节） + FLV body（PreviousTagSize0 + Tag1 + PreviousTagSize1 + Tag2 + ...

```
       + PreviousTagSizeN - 1 + TagN）
165     TagType：Script Tag（18）、Video Tag（9）、Audio Tag（8）
166 16、RTSP方法：OPTIONS、DESCRIBE、SETUP、PLAY、TEARDOWN，默认端口554
167
168
169
```