1、C++三大特性：

封装，继承，多态

封装：将属性和方法封装一起，隐藏内部实现，向外暴露接口

继承：子类继承基类的方法与属性，可以重写或扩展基类的功能

多态：允许同一接口呈现不同行为

2、C++中struct和class的区别：

struct 继承与访问都是默认public，存储一些结构简单的数据结构，与C语言兼容

Class继承和访问都是默认private，鼓励了更加严格的封装与抽象，与C语言不兼容

struct与union的区别：

struct每个成员都有自己的内存空间，大小为各成员之和，存储不同但是相关联的数据，可以同时访问多个数据

union所有成员共享一块内存空间，大小为最大成员大小，存储多个类型的数据，但一次只能使用一种，只能访问当前存储的成员

指针和引用的区别

指针存储另一个内存变量的地址，初始化的时候可以为空，存在多级指针，sizeof得到的是指针本身的大小，灵活但是容易出现野指针、内存泄漏

引用是对象的别名，在定义时需要初始化，不能为nullptr，sizeof引用得到的是对象本身的大小，不存在多级引用，一旦绑定某个变量就不可改变

空类可以实例化吗？

可以，示例会占用最小内存，大小为1字节，以确保每个对象有一个唯一的地址

当一个类中没有任何成员变量和成员函数，这是sizeof(A)的A值为多少？

其实就是空类，空类会占用最小内存，大小为1字节，编译器必须为每个对象分配一个独特的地址，以便在程序中区分不同的对象示例，因此会占用1字节的空间

子类不能继承父类的函数有哪些？

构造函数

析构函数

拷贝构造函数

赋值构造函数

私有成员函数

C++构造函数初始化列表执行顺序：

先调用基类的构造函数，如果有多个基类则按照它们在列表中的顺序构造

按照成员变量在类中声明的顺序进行初始化，而不是按照它们在初始化列表中的顺序初始化

再调用派生类构造函数的主体部分

构造函数的顺序，析构函数的顺序

构造函数：

先调用基类的构造函数，如果有多个基类则按照它们在列表中的顺序构造

按照成员变量在类中声明的顺序构造

最后调用构造函数的主体部分

析构函数：

先调用析构函数的主体部分

在根据成员变量在类中声明的顺序，反向销毁成员变量

最后调用基类的析构函数，如果有多个基类，也是按照它们在列表中的顺序反向调用

10、C++有哪些构造函数？

默认构造函数

参数化构造函数

委托构造函数：在构造函数中调用其它的构造函数

移动构造函数：转移构造所有权，以&&作为参数，销毁传进来的对象，调用的话临时对象作为右值，直接返回类对象，或者显式调用std::move

拷贝构造函数

显示构造函数

11、成员变量初始化顺序：

成员在类中声明的顺序

12、局部变量与全局变量的区别：

局部变量的声明周期就是所在代码块的运行周期，通常存储在栈上，作用范围为代码块内，初始化时为未定义值

全局变量存储在数据段，初始化默认为零值，生命周期从程序开始到结束，作用范围在整个程序范围内有效

13、常量指针与指针常量的区别

const int \* ptr 常量指针 指针指向的对象的值不可改变

int \* const ptr 指针常量 指针指向的对象不可更改

14、智能指针有什么缺点？

指针指针能极大的简化内存管理，但是也会带来性能开销、循环引用导致的内存泄露和代码复杂性增加的问题，要避免循环引用的情况。

15、shared\_ptr是不是线程安全的？

引用计数是线程安全的，被管理的对象和读写机制需要我们自己添加同步机制来保证线程安全。

指向同一对象（浅拷贝）共享同一实体，任何修改对所有指针可见

指向深拷贝后的资源，初始相同，修改后可能不同，初始内容复制自身资源，然后续修改仅影响当前指针指向的资源。

16、什么时候使用智能指针，什么时候使用裸指针？

智能指针：

适用于需要自动管理声明周期的场景

适用于需要防止内存泄露的场景

适用于需要管理复杂对象的所有权和生命周期的场景

适用于需要保证异常情况下资源能被正确释放的场景

裸指针：

适用于简单的、临时的、不涉及所有权的对象

对性能开销要求高的场景

使用与不需要管理资源声明周期的场景

17、share\_ptr和weak\_ptr的区别与联系？

共享指针共享资源的所有权，增加对象会增加引用计数，控制资源的生命周期，可能导致循环引用导致内存泄露

弱指针不拥有资源的所有权，不增加引用技术，不控制资源的声明周期，可以打破循环引用，防止内存泄露

18、引用作为函数参数在C++中有一下几个好处：

直接使用对象，避免了不必要的拷贝操作

可以通过引用直接修改函数参数的值

引用对象和直接使用对象在语法上保持一致

对于大禹像，使用引用的传递的效率更高

避免指针失效问题，不会控制到不可靠的数据

19、模板是在什么时候进行实例化？

模板在编译器碰到模板的具体使用时，进行实例化

开发者显示指定模板实例化

模板在编译时实例化，当编译器碰到模板的具体使用时，例如调用模板函数或创建模板类实例时，编译器会生成相应的代码

20、为什么模板类一般都是放在一个h文件中？

因为模板类的实例化发生在编译期，编译器需要在实例化模板时看到模板的完整定义与实现。

将模版类的定义与实现都放在头文件中，确保编译器在实例化模板时能够访问到完整的信息，避免链接错误。

如果将模板类的实现放在源文件中，可能导致链接错误，因为编译器在链接阶段无法找到模板示例化的具体实现

21、define宏定义和const、static的区别：

define 无类型检查，全局，编译时直接替换文本，无法设置断点，定义简单常量或者表达式

const 有类型检查，作用域内，编译时常量，无法设置断点，定义常量通常表示为只读

static 有类型检查，文件内或函数内的局部作用域内，静态存储，生命周期贯穿整个程序，无法设置断点，定义静态变量或者静态成员变量

22、new和malloc申请的是哪里的内存？如何减少内存碎片？

new和malloc都是申请的堆区的内存，但new会调用构造函数进行初始化，并在分配失败的时候抛出异常，malloc不调用构造函数，需手动初始化，并在分配失败的时候返回NULL。

减少内存碎片的策略：

使用内存池、按需分配大块内存、避免频繁的分配和释放操作、合并相邻的空闲块

23、实际情况中想要new分配内存但又不确定大小怎么做？

总结：

确定最大可能的大小，然后用new分配内存；

使用std：：vector或std：：unique\_ptr等动态容器，根据动态调整内存；

使用基类指针指向派生类对象，动态确定对象实际类型和大小。

24、堆和栈的区别：