1、 C++程序设计语言的设计理念。

C++ was designed to provide Simula’s facilities for program organization together with C’s efficiency and flexibility for systems programming.
追求高效、灵活。(为每种应该支持的风格提供全面的支持。)
允许一个有用的特征比防止各种错误使用更重要。Programmer needs to be trusted.

2、 C++程序设计语言的演化历程（主要贡献者)。

1) Ole-Johan Dahl（奥利·约翰·达尔）提出 OOP;
2) Kristen Nygaard（克利斯登·奈加特） 发明 Simula;
3) 丹尼斯·里奇（Dennis Ritchie）和 Ken Thompson（肯·汤普逊）设计 C 语言;
4) 1980 年，贝尔实验室的 Bjarne Stroustrup 对 C 语言进行改进和扩充，形成了”C with class”;
5) 1983 年，Rick Mascitti 将其正式命名为 C++;
6) 1994 年，制定 ANSI C++标准草案。

3、 C 和 C++的关系。

1) C++完全包含 C 语言成分;
2) C++ 支持 C 所支持的全部编程技巧;
3) 任何 C 程序都能被 C++ 用基本相同的方法编写，并具备相同的时间和空间开销;
4) C++还引入了重载、内联函数、异常处理等功能，对 C 中过程化控制及其功能进行了扩充;
5) 此外，C++还添加了 OOP 的完全支持。

4、 C 和 C++混合编程应该注意的问题。

1) 名变换：用 extern "C"修饰变量或函数，按照 C 语言方式编译和连接，限制 C++编译器做 name mangling（名变换)，确保 C++和 C 编译器产生兼容的 obj 文件;
2) 静态初始化：C++静态的类对象和定义在全局的、命名空间中的或文件体中的类对象的构造函数通常在 main 被执行前就被调用，只要可能，用 C++写 main()，即使要用 C 写 Main 也用 C++写;
3) 内存动态分配：new/delete 调用 C++的函数库，malloc/free 调用 C 的函数库，二者要匹配，防止内存泄露;
4) 数据结构兼容：将在两种语言间传递的东西限制在用 C 编译的数据结构的范围内；这些结构的 C++版本可以包含非虚成员函数，不能有虚函数。

5、 影响表达式求值的因素。

优先级、结合性、求值次序、类型转换。

6、 多态的含义。

1) 某一论域中的一个元素可以有多种解释;
2) 相同的语言结构可以代表不同类型的实体;

3) 相同的语言结构可以对不同类型的实体进行操作；
4) 一个公共的消息集可以发送到不同种类的对象，从而得到不同的处理。

7、 C++多态的几种表现形式。

虚函数、函数重载、操作符重载、模板、继承、函数指针。
可分为四类：重载多态、强制多态、包含多态和参数多态。

8、 引用类型的含义、作用、用途（何时将返回值定为引用）和滥用的危害。

含义：为一块已有的内存空间取一个别名。
作用：避免耗费资源，提高访问效率。主要用于函数参数传递和动态变量命名。
用途：返回对象本身。
危害：不应该把局部量或局部量的地址作为返回值。

9、 内联函数的作用、滥用的危害和使用建议。

作用：
1) 提高程序的可读性；
2) 提高程序的运行效率。
危害：
1) 增大目标代码；
2) 病态的换页；
3) 降低指令快取装置的命中率。
使用建议：
用于使用频率高的小段代码。

10、 如何利用析构函数防止类存泄露。

对象消亡时，在系统收回他所占的存储空间之前，系统将自动调用析构函数。一般情况下不需要定义析构函数，但是如果对象在创建后申请了一些资源并且没有归还这些资源，则应定义析构函数来在对象消亡时归还对象申请的资源在对象创建时，系统会为对象分配一块存储空间来存储对象的数据成员，但对于指针类型的数据成员来说，系统只分配了存储该指针所需要的空间，而没有分配指针所指向的空间，对象自己需要申请（作为资源）。同样，在对象消亡时，系统收回的只是指针成员本身的存储空间，而指针所指向的空间需要对象自己归还（作为资源）。

11、 纯虚函数、虚函数和非虚函数的定义原则。

纯虚函数：只给出函数声明而没有给出函数实现。只有函数接口会被继承，子类必须继承函数接口，提供实现代码。要求子类定义函数时使用。
虚函数：函数的接口及缺省实现代码都会被继承，子类必须继承函数接口，可以继承缺省实现代码。允许函数重定义时使用。
非虚函数：函数的接口和其实现代码都会被继承，必须同时继承接口和实现代码。

12、 C++程序设计应该遵守的原则。

例：
1) Use const whenever possible;

2) Guard against potential ambiguity;
3) Strive for exception-safe code;
4) Use destructor to prevent resource leaks。

13、 Const 的用法。

1) 对数据：表示为常量，不可修改。
2) 对指针：
   a) 常量指针，常量只能被常量指针指向；
      （ps：常量指针可以指向非常量，用于消除函数副作用）
   b) 指针常量，指针指向的内容一经确定后不能改变；
   c) 常量指针常量。
3) 对函数：
   a) const 参数，参数值一经确定后不能改变，可消除函数副作用；
   b) const 返回值，返回值为常量类型，不能改变；
4) 对类：
   a) const 成员变量，在成员初始化表中初始化；
   b) const 成员函数，不能改变成员变量的值。

14、 引入构造函数的原因。

成员初始化的需要。

1) 由于访问权限控制，通常不能直接赋值；
2) 若使用普通成员函数，一方面需要显式地调用，使用不便，另一方面可能导致未调用初始化函数就使用对象，不安全。

15、 面向对象程序设计的主要特点。

封装、继承和多态。通过消息传递来实现程序的运转。

16、 静态绑定和动态绑定的概念。

静态绑定发生在编译时刻，依据对象的静态类型进行，效率高，但灵活性差。
动态绑定发生在运行时刻,依据对象的实际类型动态进行,灵活性高,但效率低。
C++中默认为前期绑定，后期绑定需显式地指出。

17、 代码重用的方法。

1) 内联函数；
2) 继承；
3) 模板。

18、 引用类型与指针类型相比的优势。

1) 引用是采用直接访问形式，指针则采用间接访问形式——效率高；
2) 引用与被引用变量共享内存，而指针有自己的内存空间——内存占用少；
3) 在作为函数参数类型时，引用类型参数的实参是一个变量，而指针类型参数的实参是一个变量的地址——代码可读性好；
4) 引用类型一旦定义后不能改变，而指针变量定义后可以指向其他同类型的变量——安全。

以下为课件中内容，自行补充：

1、 结构化程序设计的缺点。

1） 数据与操作分离，代码可读性差，难以理解；
2） 代码重用性差。

2、 函数副作用的危害以及如何消除。

危害：
1） 破坏了程序的可移植性；
2） 降低了程序的可读性。

消除：
1） 常量指针。

3、 使用内联函数的限制。

1） 非递归；
2） 由编译系统控制。

4、 静态全局变量的作用。

1） 防止名冲突；
2） 值可靠。

5、 静态全局函数的作用。

1） 限制文件外部代码对函数的使用；
2） 对函数定义的补充。

6、 面向对象编程的优点。

提高开发效率和软件质量：
1） 更高层次的抽象；
2） 数据封装；
3） 更好地模块化支持；
4） 软件复用；
5） 对需求变更有更好的适应性。

7、 后期绑定的实现方法。

创建一个虚函数表，记录所有虚函数入口的地址。对象的内存空间中含有一个指针指向其虚函数表。

8、 多态的作用。

1） 提高语言的灵活性；
2） 实现高层软件的复用。

9、 new/delete 重载的意义。

频繁调用系统的存储管理，影响效率。程序自身管理内存，提高效率。

10、 引入模板函数的原因。

1) 宏实现的缺陷：
   a) 只能实现简单的功能；
   b) 没有类型检查；
   c) 重复计算。
2) 函数重载的缺陷：
   a) 需要定义的重载函数太多
   b) 定义不全。
3) 函数指针的缺陷：
   a) 需要定义额外参数；
   b) 大量指针运算；
   c) 实现复杂；
   d) 可读性差。

11、 引入异常处理机制的原因。

发现异常之处与处理异常之处不一致。使用函数参数程序结构不清楚。