1. 字符型有两种char和wchar\_t。wchar\_t是扩展字符集。wchar\_t ch = L'c';
2. 调用默认构造函数初始化: string s;或者string s();都可以。
3. 直接初始化和复制初始化。复制初始化使用等号，直接初始化把值放到括号中。如int i=10; int i(10);
4. 如果没有提供默认的构造函数，系统会提供一个。
5. 变量只能定义一次，但可以声明多次。变量声明的语法： extern 变量类型 变量名; 变量声明说明变量定义在其它地方。
6. 定义常量 const int I = 100; const 常量默认为文件的局部变量，非const变量默认为extern.要使const变量能够在其它文件中访问，必须指定它为extern.
7. 引用是变量的别名，定义时必须初始化。 int &i = j;
8. Const引用指向const对象的引用。 const int i =10; const int &j = I;
9. typedef 1)定义变量名，2）前面加上typedef, 3)把变量名作为类型名。
10. 枚举 enum enumType{item1,item2};
11. Struct和class唯一的区别是struct默认是public , class默认是private.
12. 预防多次包含同一个文件

#ifndef SALESITEM\_H

#define SALESITEM\_H

// Definition class

#endif

1. Using声明 using namespace::name;
2. 标准库类型：string vector bitset
3. 在函数体外定义的内置类型、内置类型数组，其元素均初始化为0. 在函数体内定义的内置类型、内置类型数组，其元素均未初始化。 类类型及类类型数组不管定义在哪里，均自动调用该类的默认构造函数初始化。如果没有默认的构造函数，则必须为数组的元素提供显式初始化。
4. C与C++以null作为结尾描述字符串的结尾。
5. 数组名指向数组首元素的指针。不能定义指向引用的指针。void \* 可以保存任何类型对象的地址。void \*可以作为参数或者返回值。
6. 指针和const: const int \*p;//指向常量的指针：一个指针指向常量 int \* const p;//const指针：一个普通指针，一旦赋值不能改变。
7. 指针数组： int \*p[10]; 数组指针: int (\*p)[4];
8. C++使用new和delete来动态分配内存。动态分配数组时，如果数组元素具有类类型，将调用默认构造函数初始化；数组元素为内置类型时，无初始化。string \*pStr = new string[10]; delete[] 释放指针指向的数组空间。
9. 函数参数的传递，如果形参为引用类型，则它只是实参的别名。如果形参为非引用类型，则复制初始化实参的值（复制构造函数）。
10. int main(int argc, char \*\*argv) { ... } main函数声明。
11. 默认实参：一个参数有默认实参，则后面的所有参数都必须有默认实参。
12. 静态局部对象：一个变量位于函数的作用域中，但是生命周期跨越了该函数的多次调用。static int I = 0;
13. 内联函数：函数声明前加上inline, inline只是一个建议，编译器有权忽略它。
14. const 对象、指向const对象的指针或者引用只能调用const成员函数。
15. const函数在类内声明和类外定义都需要加上const.
16. [初始化](http://baike.baidu.com/view/872.htm)列表以冒号开头，后跟一系列以逗号分隔的初始化字段。初始化列表只能在定义时指定，不能在声明时指定。成员是按照他们在类中出现的顺序进行初始化的，而不是按照他们在初始化列表出现的顺序初始化的。
17. 顺序容器：vector、list、deque、stack、queue、priority\_queue.关联容器：set、map、multimap、multiset。
18. 所有成员必须在类的内部声明。
19. 除了定义数据成员和函数成员之外，类还可以定义自己的局部类型名字。
20. 类的定义以分号结束。
21. 类的成员可以修改（即使在const成员函数内），可以通过mutable声明。 mutable int I;
22. 初始化const成员的唯一机会是在构造函数初始化列表。
23. 友元机制允许一个类将其所有的成员访问权授予制定的函数或类。友元类： friend class ClassName; 友元函数
24. static类成员:static成员函数没有this形参，它只能直接访问类的static成员。static标识符只能出现在类声明中，不能出现在类定义中。static成员函数不能为const或者虚函数。Static成员变量在类内声明，在类外初始化，类外不能加static. Static const常量类外初始化const不能省略。
25. 复制构造函数、赋值操作符和析构函数总称为复制控制。如果没定义这三个函数，系统会提供默认版本。复制构造函数和赋值操作符进行浅复制。
26. 复制构造函数可以用于：1）根据另一个同类型的对象显式或隐式的初始化一个对象。2）复制一个对象，将它作为实参传给函数。3）从函数返回时复制一个对象。初始化列表初始化对象。
27. C++支持两种初始化，复制初始化=和直接初始化()。直接初始化调用相应的构造函数。复制构造函数首先使用指定构造函数来创建一个临时对象，然后调用复制构造函数来调用对象。经过实验这是错的，实际上两者都是调用相应的构造函数。
28. 赋值操作符重载：Sales\_item& operator=(const Sales\_item &);。
29. ++，--前置 没有参数，后置有一个int参数。运算符重载分为成员重载和普通函数重载。
30. 在C++中，多态性仅能用于通过继承而关联的类型的引用和指针上。
31. 基类使用virtual指定希望多态的函数，来在派生类中重新定义。Virtual在类内声明中指定，类外定义时不能加virtual.
32. 在C++中，通过基类的引用或者指针来调用虚函数时，发生动态绑定。
33. 成员函数默认是非虚函数，对非虚函数的调用在编译时确定。在函数声明中加virtual，成为虚函数。
34. 继承定义： class classname: access-label baseclass{}; access-label可以是public\protected\private. 类默认继承的方式是private.
35. 一旦在基类中声明了virtual，那么这个函数就永远是虚函数，在派生类中可以不再加virtual.
36. 派生类中虚函数的声明必须与基类中的定义方式完全匹配，但有一个例外：返回对基类型的引用（或指针）的虚函数。派生类中的虚函数可以返回基类函数所返回类型的派生类的引用（或指针）。
37. 在某些情况下，希望覆盖虚函数机制并强制函数调用使用虚函数的特定版

本，这里可以使用作用域操作符：

Item\_base \*baseP = &derived;

double d = baseP->Item\_base::net\_price(42);

只有成员函数中的代码才应该使用作用域操作符覆盖虚函数机制。

派生类虚函数调用基类版本时，必须显式使用作用域操作符。

如果派生类函数忽略了这样做，则函数调用会在运行时确定并且将是一个自身调用，从而导致无穷递归。

1. 通过基类的引用或指针调用虚函数时，默认实参为在基类虚函数声明中指定的值，如果通过派生类的指针或引用调用虚函数，则默认实参是在派生类的版本中声明的值。
2. 派生类不能访问基类的 private 成员，也不能使自己的用户能够访问那些成员.

• 如果是公用继承，基类成员保持自己的访问级别：基类的 public 成员为

派生类的 public 成员，基类的 protected 成员为派生类的 protected

成员。

• 如果是受保护继承，基类的 public 和 protected 成员在派生类中为

protected 成员。

• 如果是私有继承，基类的的所有成员在派生类中为 private 成员。

1. 使用 class 保留字定义的派生默认具有 private 继承，而用 struct 保留字定义的类默认具有 public 继承：
2. 友元关系不能继承。基类的友元对派生类的成员没有特殊访问权限。如果基类被授予友元关系，则只有基类具有特殊访问权限，该基类的派生类不能访问授予友元关系的类。
3. 派生类初始化列表只能初始化派生类的成员，不能直接初始化继承成员。派生类构造函数通过将基类包含在构造函数初始化列表来间接初始化继承成员。多继承时，基类的初始化顺序与继承的顺序相同，与初始化列表的顺序无关。一个类只能初始化直接基类。
4. 派生类的复制控制:如果在派生类中显示的定义了自己的复制构造函数或赋值操作符，则定义将完全覆盖默认定义。这是需要你负责对基类成分的复制或赋值。
5. 复制构造函数： derived(const derived& d):base(d)
6. 如果派生类定义了自己的赋值操作符，则该操作符必须对基类部分进行显式赋值。

Derived &Derived::operator=(const Derived &rhs)

{

If(this == &rhs){

return \*this;

}

Base::operator=(rhs);

Return \*this;

}

1. 析构函数不负责撤销基类对象成员，编译器总是显式调用派生类对象基类部分的析构函数。但析构函数通常应该定义为虚析构函数。构造函数和赋值操作符不是虚函数。如果在构造函数或析构函数中调用虚函数，则运行的是为构造函数或析构函数自身类型定义的版本。
2. 与基类成员同名的派生类成员将屏蔽对基类成员的直接访问。可以使用作用域操作符访问被屏蔽的基类成员。
3. virtual <类型><函数名>(<参数表>)=0; 包含纯虚函数的基类叫做抽象基类。
4. 虚基类的原则是基类在内存中只有一份对象。

class x1:virtual public x

{

//……

};

class x2:virtual public x

{

//……

};

1. template<class T>  //模板声明,其中T为类型参数  
   T max(T a,T b,T c) //定义一个通用函数,用T作虚拟的类型名  
   {  
      if(b>a) a=b;  
      if(c>a) a=c;  
      return a;  
   }

template <class T, int i> // template<模板参数表>

class TempClass //class 类名TempClass

{public:

TempClass(void);

~TempClass( void );

int MemberSet( T a, int b );

private:

T Tarray;

int arraysize;

};

#define min(i, j) (((i) < (j)) ? (i) : (j))