

C++

一. 常量: 程序运行过程中值不变.

整型、实型、字符型、字符串、布尔型.

变量: auto ~~在~~: 暂时性存储. (堆栈方式)

static: 在内存中以固定地址存放. 整个程序运行期间有效.

extern: 在所有函数和程序段中都可引用.

register: 通用寄存器.

符号常量: const 数据类型说明符 常量名 = 常量值.

[强制与态] 级别: char, short, int, unsigned, long, unsigned long,
float, double

二. 函数.

1. 参数传递

{ 值调用. (单向传递)

{ 引用调用 (有明引用时, 同时对它的数据使之指向已存在的对象).

2. 内联函数.

编译时嵌入调用处. 节省了参数传递、控制转移.

3. 默认形参

在定义时可预先声明默认的形参值. 从右向左声明.

不同作用域内允许说明不同的默认形参.

4. 函数重载:

- { 形参类型不同
- { 形参个数不同.

编译器实现最佳匹配

三. 类与对象

- { 抽象 (数据抽象) (行为抽象).
- { 封装: 可为可重用模块.
- { 继承.
- { 多态.

对象是类的实例化

1. 访问控制:

基类中的函数	基类	继承类
public	public protected private	public protected 不可直接访问
protected	public protected private	protected protected 不可直接访问
private	public protected private	private private 不可直接访问

2. 类的成员函数:

① 声明

返回值类型 类名 :: 函数成员名 (参数表) { --- }

② 带默认并参值的成员函数, (与前面所述相同)

③ 内联成员函数: 在编译时被插入到每一个调用它的地方。

{ 隐式声明: 放在类体内。

{ 显式声明: 加 inline 写在类体外。

④ 对象: 类名 对象名

访问对象的公有成员: 对象名. 公有成员函数名 (参数表)

① 构造函数:

1) 对数据成员的初始化. 在对象创建时自动调用

2) 若没有构造函数, 编译器会自动生成默认形式的构造函数 (不做任何事情)

3) 可以是内联函数, 可带参数表, 可带默认的并参值, 可重载

4) 对象所占的内存只放数据成员, 不存函数成员代码的副本。

② 拷贝构造函数:

把每个数据成员的值都复制到新对象中。 (同类的对象)

1) 当同类的的一个对象去初始化该类的一个对象时。

2) 调用非参的类的对象的函数, 进行形参传值时。

3) 函数返回值是类的对象的函数执行完, 返回给调用者时。

(调用拷贝构造函数将对象的数据成员值拷贝到临时对象中，临时对象的值再通过“=”赋给调用者，然后临时对象自动消失，无需析构函数)

(若没定义类的拷贝构造函数，则自动生成默认的拷贝构造函数，起复制数据成员的功能)

③析构函数

在对象生存期结束时刻自动被调用，内存释放。

可以是虚函数。

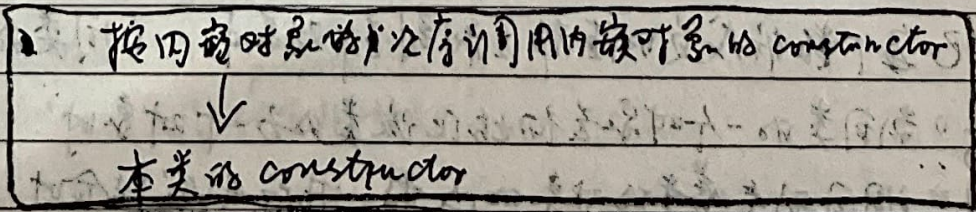
不显式声明则系统生成一个默认的空析构函数。

4. 类的组合. 即类的对象内嵌到另一个类中作为数据成员.

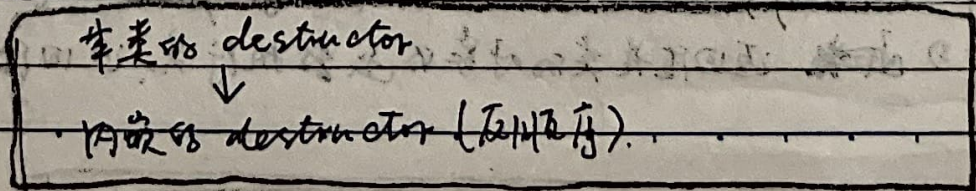
①构造函数: 类名::类名(开列表): 内嵌对象1(开列表), 内嵌对象2(闭表)
{ 类的初始化 } 称为初始化列表.

1) 对基本类型的数据成员也可通过初始化列表进行初始化, 比赋值语句效率高.

2) 调用构造函数的顺序:



②析构函数:



④ 拷贝构造函数:

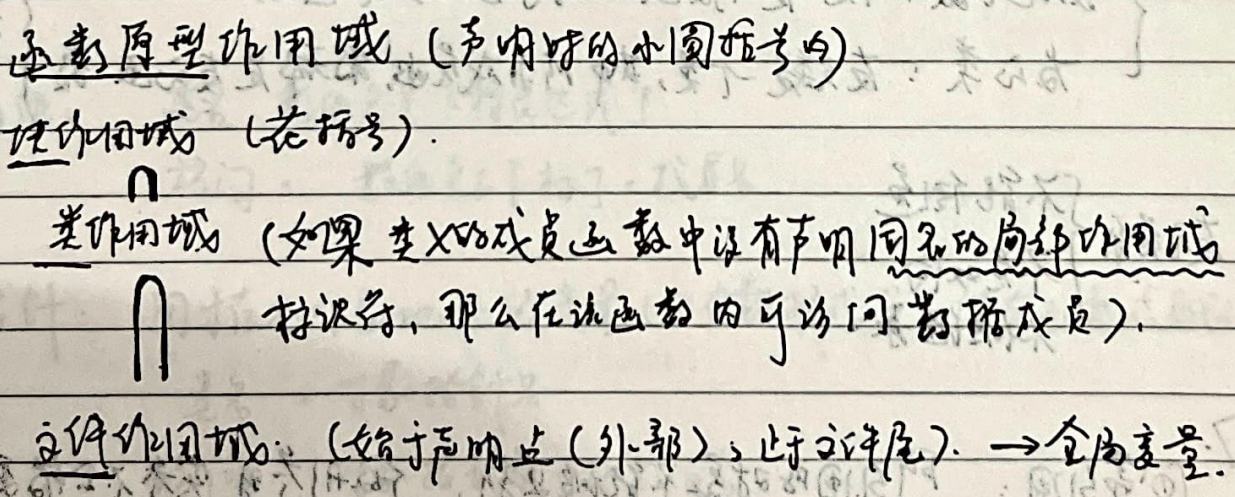
形式: $C::C(C \&c1) = b(c1.b)$

C类包含B类的对象b作为成员
c1为C类的一个对象.

注: 循环依赖时要 前向引用声明.

四. C++ 程序结构

1. 作用域: 标识符的有效范围



2. 可见性: 标识符是否可以被引用

先声明再引用

同名隐藏规则 (内层优先)

3. 生存期

- 静态生存期: static 对象类型 对象名 (地址固定)
- 动态生存期: 局部产生 (暂时性生存)

三. 类与对象 (补)

通过非内联函数访问

- 5. (类的静态成员: static 声明成员, 有静态生存期, 通过类名: 标识符访问)
- 类的非静态成员: 在每个对象中都有一个 copy

类的静态函数成员: 访问该类的静态数据和函数成员.

维护对象之间共享的数据, 不依赖于任何对象.

6. 类的友元.

- 友元函数: 友元是 ^非一般函数或类的成员函数.
 - 友元类: 友元是一类, 其中所有成员函数都是友元函数.
- 访问私有和保护成员.

友元关系 {

- 不能传递
- 不是双向
- 不被继承

- 7. ① 常引用: 所引用的对象不能被更新, 做形参时实参不会被更改.

const 类型说明符及白用名.

所引用的对象

- ② 常对象: 必须初始化, 不能被更新 (整个对象的生存期).

类名 const 对象名;

改变方法与常外通过对象不访问其私有数据

- ③ 常成员函数: 返回类型 函数名 (括号) const;

不能更新对象的数据成员。

不能调用类集中非成员函数。

常对象只能调用常成员函数，也是常对象唯一的对外接口。

常成员函数是同名非常成员函数的重载。

④ 常数据成员：(常量 or 常引用)

任何函数都不能对其赋值。

构造函数只能通过初始化列表 ("冒号"加"括号") 初始化。

8. 讨论

对象数组：类名 数组名 [下标表达式]

访问：数组名下标 · 成员名。

对象指针：用存放对象地址的变量 (只有数据成员，没有函数成员)。

类名 * 对象指针名

访问：对象指针名 -> 成员名

this 指针：指向当前被成员函数操作的对象

五. 动态内存分配

new 类型名 (初始化列表)

delete 指针名。

当然，类型名包括类名，(类类型) 此时指针就是对象指针。

多维数组 (动态): `new 类型名 [下标1] [下标2] ...`

`new 类型名 (x, y) [下标2] ...`

同样指向 C 类型的多维数组。

`delete [] p`

[动态数组]:

流拷贝: 两个数组占用同一内存区, 导致 `delete` 时在同一内存上释放了两次

深拷贝: 创建一块新内存区, 两次释放为不同内存区, 要自己写!

六. 继承与派生.

1. 声明: `class 派生类名: 继承方式 基类名1, 继承方式 基类名2 ...`

{ ... };

{ 多继承
单继承 }

2. 规则

{ 继承除了构造函数和析构函数以外的所有成员 (包括 copy constructor)
派生的析构函数的同名重载规则 (内层优先)
需加入新的构造函数和析构函数.

类型兼容规则:

凡是基类能解决的问题, 公有派生类都可解决

- 派生类的对象 可赋值给基类对象
- 派生类的对象 可初始化的基类的引用。 → 在构造(派生类)起作用。
- 派生类的对象 也可 赋值给基类的指针。

3. 派生类的构造和析构函数.

派生类的默认构造函数只对派生类 新增数据成员 初始化。
若欲对从基类继承下的成员也初始化, 则需再编写 构造函数。
析构函数 也如此。

① 构造顺序: 调用 基类 的构造函数. (左→右)

↓
调用 内嵌对象 的构造函数 (左→右). 只是基类的 数据成员

↓
派生类新成员 的构造.

派生类名: 派生类名 (参数表长) = 基类名 (参数表1) ...

基类名 (参数表n), 内嵌对象名 1 (内嵌对象参数表1), ...,

内嵌对象名 m (内嵌对象参数表m).

{ 派生类新增成员的初始化语句 }

② 排日构造函数: $C::C(C \& C1) = B(C1)$

基类的引用 ~~基类~~ 派生类的引用. (类型兼容性规则).

1) 派生类的虚函数覆盖了基类的虚函数, 隐藏基类中同名函数的所有其他重载形式.

2) 虚函数是动态联编的, 但默认形参值是静态联编的.

3) 虚析构函数 `virtual ~类名()`.

派生类虚函数 (修改基类的行为)

派生类非虚函数 (不希望被派生类改变功能)

通过使用基类类型指针可访问到该指针正在指向的派生类的同名函数.

3) 函数模板:

```
template <typename T>
```

类型名 函数名 (参数表)

{ ... }

T 表示一种抽象的类型 在调用时要具体化.

2) 类模板:

```
template <typename T>
```

class 类名

{ ... }

1) 在类外定义其成员函数:

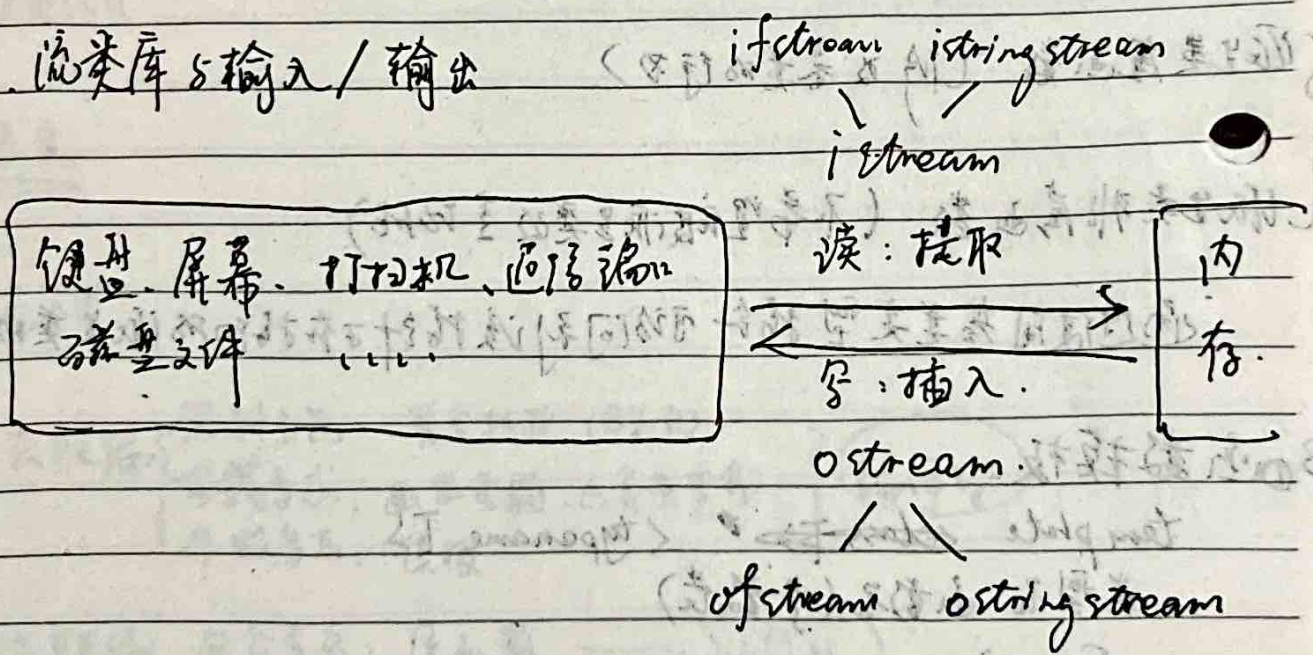
```
template <typename T>
```

类型名 类名 <T> :: 函数名 (参数表)

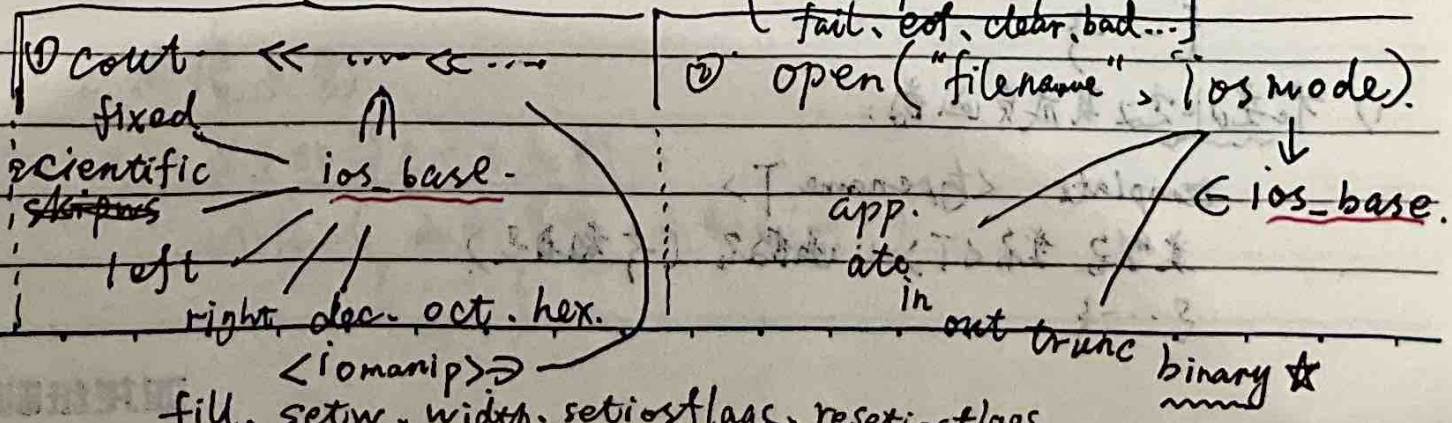
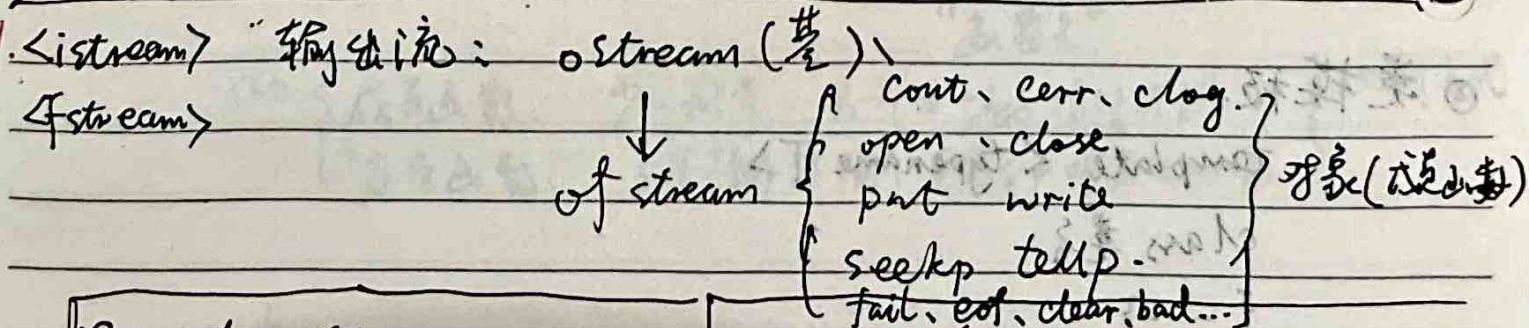
{ ... }

- 2) 模板类的成员函数必须是函数模板。
- 3) 建立对象的声明: 类模板 <模板参数表(具体)> 对象名1, ... ;

八. 流类库与输入/输出



类模板: `basic_<xxx>` ∈ <iosfwd>



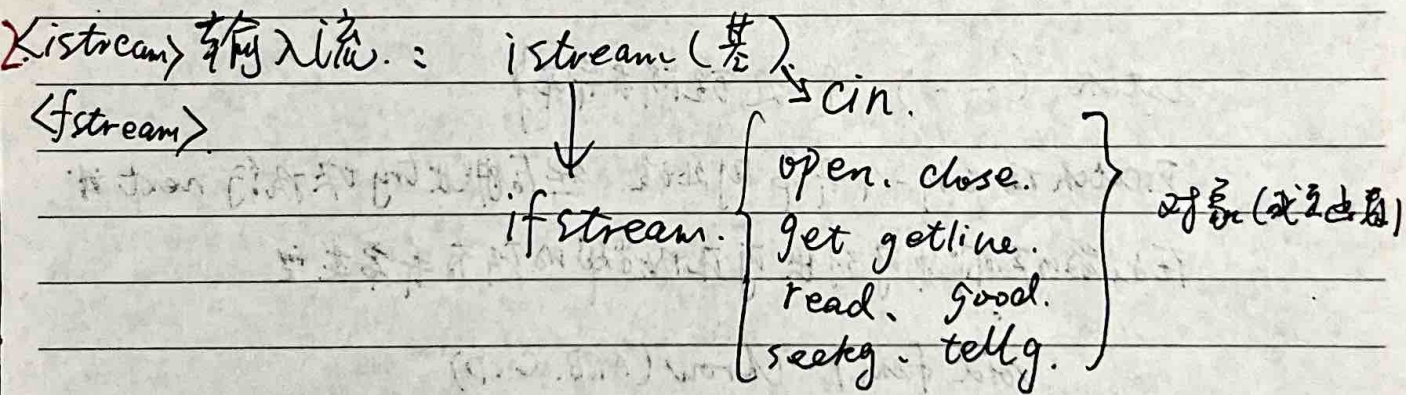
③ put ('A')

④ close () 是成员函数, 常用对象访问.

⑤ write ((char *) & x, int size.)
地址 字节数.

⑥ seekp (int).

⑦ eof. (返回非0) 未读字节返回0.
(内存已读完)



① cin >> ... >>

{ ios_base::in (默认)

ios_base::binary

② get 也读空格符.

③ getline (地址, 读的数目, 终止符)

(读完会删)

④ close()

⑤ read ((char *) 地址, *字节数)

⑥ seekg (int). (文件的终止符)

⑦ good () (返回非0) 未读到字节返回0.

九. 异常处理:

```
try { ..... throw 表达式 ..... }
```

```
    ..... throw 表达式 ..... }
```

```
catch (异常类型声明) { 处理办法 }
```

```
catch (异常类型声明) { 处理办法 }
```

```
    ;
```

```
catch (.....) { 处理所有异常 }
```

一旦 catch 接收到一个异常则处理. 然后跳出 try 块执行 next 块.

在函数的声明中可列出可抛出异常的所有异常类型

```
void fun() throw (A, B, C, D)
```

若不标, 则可抛出任何类型异常

```
void fun()
```

若如此 void fun() throw (); 则不抛任何类型