C++程序运行机制

1 对象占用内存空间大小，等于所有成员变量大小之和。
2 调用成员函数：对象名.函数名 or 指针->函数名 or 引用名.函数名
3 引用：可以看做 const 指针(不允许更新)，必须初始化，只能引用变量，不能引用常量 or 表达式(不是变量,没有指针)。
4 函数参数声明为引用：避免参数复制　返回值声明为引用：(要求是全局变量、引用类型的参数、static 型局部变量、对象属性 or 对象自身)，同样避免返回值复制。
5 成员函数内可以访问当前对象和其他对象的全部属性，调用全部函数。
6 成员函数以外，只能访问 public 成员。
7 函数同名，但是参数列表不同->重载(对返回值没有要求)
8 可以有参数缺省值(默认从前到后)(但是要注意二义性)。

类与对象的特殊成员

静态成员

1 printf 从右到左计算表达式的值，从左到右输出。
2 静态成员整个程序只有一份，所有对象共享 需要以全局变量的形式声明 or 初始化 如果是 public 属性的话，可以在没有对象生成的情况下直接访问。
3 sizeof 不会计算静态成员的大小
4 静态成员函数中不能访问非静态的成员函数&变量 也不能使用 this 指针。
5 调用：类名::成员名 or 指针->成员名 or 引用.成员名。

构造函数

1 不能通过对象&指针&引用访问　(某些编译器可以 tmp = A();)
2 初始化对象&动态分配对象&传参&返回值时调用(后两者产生临时对象)
3 不能返回值(void 也不行)
4 参数为其他类->转换构造函数
参数为自己类的 const**&**(否则递归)->复制构造函数
5 定义了构造函数，编译器不生成无参数的构造函数
定义了拷贝构造函数，编译器不生成拷贝构造函数
6 最好是 public；private 构造函数只能在成员函数&友元中用于对象初始化
7 转换构造函数：实现类型的自动转换 只有一个参数(不能是自己类的引用) 需要时，编译器调用它生成一个临时变量 eg. A1 = 5 ==> A1 = (myclass)5;
8 复制构造函数：参数为自己类对象的 const& eg. A a2 = a1; & A a2(a1);==> 调用拷贝构造函数(只在初始化时才调用) 传入参数，返回值时调用拷贝构造函数
9 可以把参数声明成 const& 来减少函数调用
10 对象间用等号赋值，按位拷贝，并不调用拷贝构造函数(调用 operator=)

析构函数

1 最多只有一个析构函数 编译器默认生成一个什么也不做的析构函数
2 对象(数组)消亡，delete 运算，返回对象 都会调用析构函数
3 传入参数(不是&的话) 返回对象的消亡都会调用析构函数

```
eg. Fun(Obj obj){
        Return obj;(参数消亡)(准确来说，是先拷贝一个，之后函数结束，参数消亡，之后拷贝的临时变量在语句结束时消亡)
    } (in main) {
    c1 = Fun(obj);(拷贝的临时对象消亡)
    return 0;(c1 消亡)
```

}

关于构造函数，析构函数调用时间==>ppt_3_1_p47 51 61 62

Ps.静态本地变量，在函数初次调用时才构造，程序结束时消亡

必须在定义类的文件中说明/初始化所有静态成员变量，否则链接不通过

封闭类&成员对象

1 由成员对象的类->封闭类

2 构造封闭类对象时，先调用所有对象成员的构造函数，之后执行封闭类的构造函数

3 成员对象构造函数调用顺序只和类声明时顺序有关，与初始化列表中顺序无关

4 消亡时，先调用封闭类的析构函数，之后执行成员对象的析构函数(与构造函数顺序相反)

5 初始化列表中的参数可以是表达式(函数，变量(只要有定义))

6 必须用初始化列表初始化的:基类的构造函数,成员对象的构造函数(只要是没有无参构造函数的成员),const 成员的初始化(因为不能赋值),引用成员的赋值表达式(因为无法重新赋值)

引用成员&const 成员&const 对象

1 const A* p = &something;则不可以通过指针修改 something 的值

2 const 指针不可以赋值给 noncconst 指针(这样的话就可以修改值了,与本意相违)

3 函数参数前加 const 保证不修改参数值

4 const 引用不可被赋值(但原来的变量是可以被赋值的)

5 const 成员，引用成员必须在初始化列表中被赋值 eg. A(int n_, int& r_) :n(n_),r(r_){};

6 声明成员对象为引用:表示对象之间的关系，不用另制造一个对象

7 常量方法：

函数声明后加 const:常量成员函数，内部不能修改属性，不能调用非常量成员函数

声明时和定义时都要在后面加 const

函数名相同，参数表相同，有无 const:重载

声明对象时前加 const:常量成员，只能应用构造函数，析构函数，常量成员函数如果强制类型转换的话，是可以应用非常量成员函数的 eg. A& r = (A&) constobj;

友元

1 另一个函数(另一个类的函数(包括构造，析构函数))可以声明成友元

2 友元函数可以访问这个类的私有成员

This 指针

1 并不是成员变量，不能显式赋值

2 返回自己的地址:return this;返回自己的引用,值:return *this;

运算符重载

1 被重载成普通函数时，如果要访问 private 成员，应被声明成友元 被重载成成员函数时，通常应为 public 成员

2 重载为普通函数，参数个数为运算符目数

3 重载为成员函数，参数个数为运算符目数-1，由左操作数执行运算符函数

4 运算符不能修改左操作数的类(int / iostream)，同时又要访问私有成员时，重载成友元函数

5 c++不允许定义新的运算符，重载不能改变运算符的优先级

6 不能重载'.' '.*' '::' '?:' 'sizeof'

7 重载() [] -> = 必须声明为成员函数

运算符重载实例

1 类型转换运算符

没有返回值类型，但是有返回值 eg. Operator int(){return obj.n;}

调用样例 ppt4_1 p21

2 函数调用运算符

成员函数中有函数调用运算符"()"的:函数对象 样例 ppt4_1 p 22,23

3 下标运算符

通常返回引用，来支持数组元素的读和写，否则成为只读数组

4 赋值运算符

返回 const&来允许 a = b = c;

包含指针的类，需要自己编写拷贝构造函数(深拷贝)，编写复制构造函数(要考虑赋值给自己的情况，要释放自己之前申请的空间)

```
eg. Const A& operator= (const A& obj){
        If(p == obj.p){}
        if(p) delete p;
        //do something;
    }
```

如果返回值不是 const，赋值表达式将只改变最左侧变量的值(应是(a = b) = c 时，此处存疑，devc++,eclipse 中 a = b = c 运行结果同不加 const 时)

5 流运算符

1 程序运行时不许复制 cout ==> ostream& operator<< (ostream& out, A obj);

2 iostream 中将<<重载成成员函数

3 istream: input.ignore()略过一个字符 input >> setw(n)只读取 n - 1 个字符

6 自增自减运算符

1 前置运算符作为一元运算符重载，后置运算符作为二元运算符重载，多写一个没用的参数

2 前置:自增，返回*this，允许++++obj;

3 后置:返回和自己相同的(临时)对象，自增，不允许 obj++++(因是临时对象)

7 总结

1 单目运算符通常重载成成员函数，左右操作数相同的双目运算符，只要重载成成员函数就好啦

2 左操作数如果有时是类的对象，有是不是类的对象，重载两次，一次成员函数，一次友元函数

继承

1 派生类自动拥有基类的全部成员，派生类中定义的新的成员覆盖基类中的同名成员(仍可以通过作用域运算符访问到，缺省访问派生类中定义的成员)，不同名则扩充了基类

2 派生类拥有基类的 private 成员，但是不能直接访问 private 成员(除了基类的成员函数和友元函数/类)

3 protected 成员：可以被基类的成员函数，友元函数访问(可以访问别的对象的保护成员)，派生类的成员函数只能访问当前对象的基类的保护成员 ppt5_1 p19 p20 p21

4 构造函数 总是先调用基类的构造函数(前置运算符/省略则自动调用默认构造函数)

5 先完成派生类的析构函数，自动调用基类的析构函数(所以可以不用写啦)

6 public 继承的赋值兼容(private 或 protected 继承就不可以啦)

base = derived;//OK

Base& br = derived;//OK

Base* bp = &derived;//OK

7 protected 继承：public&protected 成员成为派生类的 protected 成员

8 private 继承：public 成员成为 private 成员，protected 成员不可访问

9 base* bp = &derived;此时*bp 是一个 base 类的对象，只能访问基类的 public 成员，派生类的成员看不到啦(但是可以强制类型转换回来(但要保证指向一个 derived 对象))

10 内存模型 ppt5_1 p34-40

11 直接基类&间接基类 只需要声明直接基类， 自动向上继承间接基类

12 多继承 eg. Class d: access-specifier$_1$ base$_1$, access-specifier$_2$ base$_2$ {};

构造函数 d(): base$_1$(), base$_2$(){};

13 创建对象时，按继承顺序调用基类==>成员对象的构造函数==>派生类的构造函数

14 多继承时的二义性，多个基类中有重名函数/成员 二义性检查在访问权限检查之前进行，访问权限不会消除二义性==>用虚继承解决(ppt 中略)，慎重多继承

多态与虚函数

1 函数指针(eg. int (*p)(int a, int b))通过指向不同的函数体，访问不同函数(eg.qsort)

2 虚函数：只允许指向当前对象的成员函数，类定义时，提供实现代码，构造函数执行之后，返回之前自动初始化；执行析构函数之前释放函数指针

3 用 virtual 关键字声明虚函数，写函数体可以不加 virtual

4 虚函数表：任何一个有虚函数的类/派生类都有一个虚函数表，任何对象中也都有一个指向虚函数表的指针，三次跳转->this->(pointofvtable->pointerofvirtualfunction)->code

5 如果派生类写了一个名字相同，但是参数表不同的函数->覆盖掉基类的虚函数，如果重新(完全相同的)声明，并实现虚函数代码->重载掉基类的虚函数

6 调用函数：看对象类型(去找 vtable) 访问权限：*指针的类型

7 一般来说，虚函数的类的析构函数也应该是虚函数(否则析构函数不在 vtable 里)(派生类的析构函数可以不写 virtual，但也是虚函数)

8 构造函数不能作为虚函数，也不能调用虚函数(这时候还没有 vtable 呢)

9 (打脸第八条)构造函数、析构函数调用虚函数->会调用自己类/基类中的函数(不会动态联编)，而普通的成员函数调用虚函数会是多态

10 纯虚函数：没有函数体的虚函数(eg.virtual void print() = 0)，有纯虚函数的类->抽象类

11 抽象类不能创建对象，可以声明指针，它的指针可以指向派生类的对象

12 抽象类的成员函数中可以调用纯虚函数，但是构造/析构函数不可以

13 只有实现了所有纯虚函数的类才成为非抽象类

14 多态的两种机制：派生类的指针赋值给基类的指针，调用虚函数时看对象类型

派生类的对象赋值给基类的引用，调用虚函数时看对象类型

15 多态的三种形态(老师你做 PPT 能不能前后统一一点啊...)

1 派生类对象地址赋给基类指针

2 派生类对象赋给基类引用

3 派生类对象访问基类的成员函数，如果调用了虚函数，也会调用派生类的虚函数(因为相当于做 this->virtualfun();)

泛型程序设计 模板开发

1 string ==> basic_string<char>

2 编译器在可执行程序中删除模板，种下根据需要的类型生成的函数/类

3 函数模板&类模板 STL：容器模板&算法模板

4 编译器首先检查普通函数中是否有参数类型匹配的，之后检查已经生成的函数模板(模板函数)中是否有匹配的，最后检查是否有函数模板中匹配的；类模板同理，普通类->模板类->类模板

5 类型参数在<>里(template<class T>)

6 函数模板的参数类型==>可以用类型参数，也可以用基本数据类型

7 使用多个类型参数，可以避免二义性

8 函数模板的重载：函数同名，参数数量不同

9 (又开始墨迹了)先找参数完全匹配的函数，再找参数完全匹配的模板，最后找参数自动转换可以匹配的函数，最后报错

10 类模板定义：首部声明类模板的参数 template<class T> 实例化之后得到模板类

11 每个函数前都要写类型参数 Carray<T>::function(T arg){} 类名后要声明类型参数,否则成为不同模板共享的函数

12 同一个类模板的两个模板类不互相兼容

13 非类型参数也要放到 template<>中，实现函数时也要写到类名后

```
eg. Template<class T, int arg>
    void Carray<T, argval>::function(T arg, int intarg){};
```

14 参数有一个不一样就不能互相赋值

15 类模板&继承(不要求)

```
Eg.template<class TSon, class TParent, int size>
    Class son:public base<TParent, size>{//code here}

    template<class TSon, class TParent, int size>
    Void son<Tson, TParent, size>::function(TSon arg, in size){};

    //in main
    Son<int, string, 5> sonobj;//会生成 base<string, 5>作为基类
```

16 类模板与友元:普通函数被声明成类模板/某一模板类的友元函数/友元类(此时每一个成员函数都成为友元函数)

17 不允许以函数参数(任何编译时不确定的值)作为模板非类型参数值

18 (非模板的)友元函数成为该模板所有模板类的友元,而带有类型参数的只成为那一个类型的友元函数

19 静态成员:每一个模板类有自己的静态成员

标准模板库

1 代码重用的方式:结构化程序设计/函数指针/抽象与封装/继承与多态/泛型程序设计

2 标准模板库:容器类模板/算法类模板

3 容器:每一个容器存储的元素必须是同类型的/可以存储基本数据类型/类的对象

4 迭代器:用于依次访问容器中各个元素,作用与指针类似

5 算法:可以适用于任何数据结构

6 第一类容器:顺序容器(vector deque list)&关联容器(set map multi-)

7 对象被插入容器时,插入的是复制品(重载拷贝构造函数/operator ==/operator <)

8 vector:尾部增删元素较快/deque:两端增删较快/list:任何位置都较快

9 set:集合 map:映射 multi:允许相同元素/键值的元素 都以平衡二叉树实现 O(logn)

10 容器适配器:stack:后进先出 queue:先进先出 priority_queue:最高优先级的最先出

11 所有容器共有: = < <= > >= == !=(按字典序比较)

empty() max_size() size() swap()

12 仅第一类有的:begin() end() rbegin() rend() erase() clear()

13 迭代器:用于指向第一类容器的元素 通过非 const 迭代器可以修改指向元素

14 迭代器由弱到强:输入->输出->正向->双向->随机访问(向下兼容)

15 所有迭代器 ++p p++

输入迭代器 *p p = p1 p == p1 p != p1

输出迭代器 *p p = p1

正向迭代器 以上全部

双向迭代器 --p p--

随机访问迭代器 p += i p -= i p + i p - i p[i] p </<=/>/>= p1

16 vector deque 支持随机访问迭代器 list map set 双向迭代器

17 vector<T>::value_type i <=等价=> T i

18 算法示例:find(iter first, iter last, const T& val)区间左闭右开

容器

1 顺序容器:元素位置与存储空间顺序一致

关联容器:顺序与值大小关系有关(二叉树)

适配器容器:顺序与进入容器时间一致

2 顺序容器支持 front() back()(最后一个元素的引用) push_back() pop_back()

3 template<class T> class A {typedef T datatype} A<int>::datatype n (就是 int n)

4 vector .at()会做下标越界检查(抛异常) []越界的话会扩张(未定义?)size .insert(place, val)

5 list:自己有 sort() remove() unique() merge()(会清空被合并的) reverse() splice()

6 成员函数中用 typename 说明是一个类型,没有就生成一个

7 unique()之前先 sort()

8 lst1.splice(p1, lst2, p2, p3)把[p2, p3)插入 p1 之前,在 lst2 中删除[p2, p3)

9 deque 还支持 push_front() pop_front()

10 copy(init first, init last, outit x)把[first, last)的内容拷贝到 x 后

11 accumulate(init first, init last, T val, pred pr)执行 val = pr(val, *iter)返回 val

12 lst.sort(greater<T>())按降序排序

13 关联容器支持 find() lower_bound() upper_bound() count() insert()

14 multiset<key, less<key>, allocator<key> >要求支持<运算符/自己写函数对象

15 重载< = 运算符最好用友元函数,否则似乎会有问题

16 equal_range(n)返回 pair(lower_bound(), upper_bound())

17 set<key, less<key>, allocator<key> >

18 set.insert()返回 pair<set<T>::iterator, bool> multiset.insert()返回 multiset<T>::iterator

19 map[key] = val 有则改之无则加勉(返回的是 val 的引用)

20 map[c] = map[d] <==> map.find(c).second = map.find(d).second

21 multimap<key, T, less<key>, allocator<T> > typedef pair<const key, T> value_type

22 stack<T, deque<T> >只能访问插入删除栈顶的元素 push pop top

23 queue<T, deque<T> >先进先出 push pop top

24 priority_queue pop 删除最大元素 top 返回最大元素的引用 用 less<T>比较

算法

统计类算法

1 count(init first, init last, const T& val);

2 count_if(init first, init last, pred pr);返回 pr(iter) == true 的元素个数

3 min_element(iter first, iter last);返回最小元素的迭代器(用<比较,即没有人比它小)

4 max_element(iter first, iter last);返回最大元素的迭代器(不小于任何元素)

5 for_each(init first, init last, funptr pf);对每一个元素执行 pf(e)

查找类算法

6 find(init first, init last, const T& val);

7 find_if(init first, init last, pred pr);返回 pr(*i) == true 的 i

8 binary_search(iter first, iter last, const T& val, pred pr);没有第四个参数时用<比较,否则认为 pr(x, y) == true 时 x 小于 y,要求有序

9 lower_bound(iter first, iter last, const T& val)要求有序,返回使[first,FwdIt) 中所有的元素都比 val 小的最大位置,upper_bound()同理

10 equal_range(iter first, iter last, const T& val)同上文

排序类算法

11 需要容器支持随机存储迭代器,故只可用于 vector deque

12 sort(ranit first, ranit last, pred pt);pr(x, y)为 true/x < y 则 x 应比 y 靠前(可用 greater<T>)

13 时间复杂度 n * logn,保证最坏情况下性能可以使用 stable_sort 保持相等元素次序

14 堆排序算法需要支持随机访问迭代器

15 push_heap()(n * logn) 可以通过 push_back()再 push_heap()向堆中添加元素

16 pop_heap(ranit first, ranit last, pred pr)取出最大元素,放到最后,将前面再做成堆

变换类算法

17 merge:要求原容器已排序 unique:也要求排序 transform:依次变换每一个元素

18 random_shuffle(ranit first, ranit last)要求随机访问

19 next_permutation(init first, init last)求下一个排列,双向访问迭代器

20 reverse(bidit first, bidit last)颠倒顺序

21 merge(init first1, init last1, init first2, init last2, outit x, pred pr)把两个序列升序合并到 x 中,以 pr 为比较器(默认以<为比较器)

22 unique(fwdit first, fwdit last, pred pr)去除升序序列的重复元素,返回删除后的 last()

23 transform(init first, init last, outit x, unop uop)对[first,last)中的迭代器 I ,执行 uop(*I) ; 并将结果依次放入从 x 开始的地方,返回 x + last - first

集合类算法

24 作用于升序容器

25 includes(init first, init last1, init first2, init last2, pred pr)判断 2 中元素是否都在 1 中,默认用==做比较器,可用 pr(x, y) == true 判等

26 set_difference(init first1, init last1, init first2, init last2, outit x, pred pr)将 1 中不属于 2 的元素放到 x 中,可用 pr(x, y)判等

27 set_intersection(参数表同上)将 1 2 共有的元素放到 x 开始的地方(共有的元素出现 min(n, m)次)

28 set_symmetric_difference(参数表同上)将互不在另一区间的元素放入 x 开始的地方

29 set_union(参数表同上)求两个区间的并,共有的元素出现 max(n, m)次

位操作算法

30 成员函数见 ppt_10_1_p49 50 51

输入输出流

1 cout.put()

2 while(cin >> x)因为重载了强制类型转换运算符

3 freopen(“name.txt”, “r”, stdin)将 cin 重定向为 name.txt

4 cin.get()返回读到的一个字符(可以是空白字符)
5 cin.get(char* buffer, int size, char delim = '\n')读 size - 1 个字符/遇到 delim,将其留在流里,buffer 最后加'\0'
6 cin.getline(参数表同上)但是 delim 从流中去掉
7 cin.eof()返回是否遇到结束符
8 cin.peek()偷看一眼,放回流中
9 cin.putback()放回一个字符
10 cinn.ignore(int ncount = 1, int delim = EOF)删掉 ncount 个字符,遇到 delim 中止
11 流操纵算子#include <iomanip>
12 整数流的基数 dec oct hex 居然是函数
13 cout.precision() cout << setprecision() 指定有效位数 cout << setiosflags(ios::fixed) << setprecision() 控制小数点后的位数 resetiosflags(ios::fixed)取消固定小数点位置
14 setw:流操作算子 width:成员函数 cin.width() cin >> setw() 每次都要设置宽度的
15 自定义流操纵算子 ostream& operator<< (ostream& (*p)(ostream&))可以调用函数
16 文件操作:略
漏题:
Int a[10] = {};
Muitiset<int> b(a, a + 10);