**1 《C++基础语法》**

1常用语法

(1) namespace

① 命名空间定义: namespace SpaceA {......} 作用域 (防止函数, 变量, 类的名字冲突)

使用方法：SpaceA::函数或变量;

② 匿名空间namespace{......}只能在所在文件里使用, 如同static

可以结合匿名空间一起使用，内部程序不暴露，只暴露接口

命名空间里函数或变量如果是声明, 可以在外部定义:

<data\_type> SpaceA:: fun(arguments) {......}

(2) using

① 引用名称到所使用的作用域 (函数名, 变量名, 空间类名…)

using namespace SpaceA; 把整个命名空间引入所在作用域, 但一般不提倡, 由于可能不

同空间有同名成员

using SpaceA::member; 同样在继承类里可以引入基类的功能 (using Base::func;)

② 给类型起别名 (替代typedef功能)

using uint = unsigned int; 如同typedef unsigned int uint;

using cout = std::cout;

(3) auto

编译器会根据初始化表达式自动推导出变量类型 auto x = value;（可以结合引用）不能用于函

数参数

for (auto x : y) x为变量名首元素的拷贝, 遍历有begin() 和end() 的y, 用于标准容器、数组、

自己写的类。由于该方法一开始会记录这个数据的结束位置, 所以不能在遍历的时候修改数

据(添加或删除), 因为会导致结束位置混乱

(4) 类型转换

① 隐式类型转换: 系统自动转换(不安全, 通常当数据类型不一样时, 且“=”赋值时会出现隐式转换)

② 显示类型转换: 通过explicit避免系统隐式转换, 因此要显示转换 (下述方法)

|  |  |
| --- | --- |
| static\_cast | 用于大多数安全的转换，编译器在编译期检查合法性。 |
| dynamic\_cast | RIIT对象指针的转换, 转换要明确原本的类型避免无法转换, 基类里必须有虚函数即多态情况, 因为需要虚函数表指针 (如Base b; 里面没有子类信息所以无法转换成子类, 而Base\* ap = &c; ap可以动态转换为类C因为ap动态类型原本就是类C), 在虚继承且有虚函数表时要有该转换会更安全, (满足: 继承, 有虚函数表就用动态转换) |
| const\_cast | 用于去除 const 或 volatile 限定的类型 (原本不是常量才安全) |
| reinterpret\_cast | 几乎不进行任何类型检查，指针强转 (直接按内存解释) |

2 C++函数

① 函数的形参可以设置默认值, 默认值后面的参数也都要有默认值

② 函数的占位形参: func(int) 或 func(int=5), 要给占位参数实值除非有默认值

(注: 有些operator函数使用时可以不用给占位参数值)

③ 函数的重载:

满足条件: 同一个作用域 + 相同函数名 + 不同参数列表(参数类型，参数个数，参数顺序)

(注意不要出现二义性)

④ 尾返回形式：auto fun(arguments) -> return\_data\_type {......}

3 动态内存

int\* ptr = new int(5); 或 int\* ptr = new int; 没有初始化

int\* arr = new int[5]; 5个元素的数组, (此处arr不是数组名, 只是一个连续动态首地址,

没有初始化数据)

delete ptr; 如果ptr = nullptr可以释放多次, 如果ptr ≠ nullptr 则不能重复释放 (否则报错)

delete[] arr; 释放数组

4 引用

① 引用的使用方法

引用是给一个变量起别名, 从而使他们共同指向同一个内存空间 (无论左值还是右值引用) 引用绑

定后不能再令其引到另一个变量 (只能一直绑定该变量)

引用的好处: 减少传参(形参, return值)出现的临时变量拷贝的现象 (函数栈帧)

Tip: 作为函数的返回值 (可以让返回值作为左值使用) 或是函数形参 (代替指针, 从而避免临时拷贝)

② 左值和右值引用

左值引用：

* 可以修改的左值的引用, 从而减少临时拷贝
* 普通引用: int a = 10; int& b = a;
* 数组引用: int (&arr\_2)[7] = arr\_1;
* 函数指针引用: void (\*& fun\_2)(arguments) = fun\_1;

右值引用：

* 不可以int& b = 10; 但可以const int& b = 10; (此处建立tmp = 10; b是一个不可修改的左值即引用tmp）
* int&& b =10; 是一个右值引用，引用的是10这个右值。此时b是左值可以修改
* 右值一般用在临时变量 (移动语义) std::move()将变量转换为右值

**2《封装》**

1 构造函数

2