C++(Primer & CaiNiao)

Run(Linux)

- 编译链接 g++ main.cpp -o main
- g++ main.cpp -std=c++11
- 运行可执行程序 ./main

Run(Windows)

- Visual studio

Basics Of language

- C++必须包含一个main函数
- **include<>表示编译器标准库 include"" 表示打开当前路径下的头文件**
- using namespace name表示使用name命名空间，而不需要使用name::fun 类似于from name import *
- 标准C++由三个部分组成：核心语言、C++标准库、C++标准模板库(STL)
- 命名空间namespace可以理解为定义不同的仓库namespace1::fun1()
  - 定义：namespace namespace1{}，可以是定义新的，也可以是给原有的命名空间增加元素
  - 使用：namespace1::fun1()或者using namespace namespace1 多个文件下都能知道
- 常量：
  - 数字字符串常量
  - #define A value1预定义常量，define和const区别
  - const type name = value1定义常量
- 分号表示语句结束，语句块不需要加分号
- C++注释，//或者/**/
- 数据类型：自定义、内置、衍生类型
  - 基础类型bool char int float double(1,2,4,8字节) void, 可以加修饰符signed unsigned long short
  - 定义type name1, name2 = value2
  - 重新定义typedef type name
  - 变量只能被定义一次，可以多次被声明(extern)
- 运算符：
  - 算数运算符：+ - * / %

    - 赋值运算符：++ += -= /=
    - 逻辑运算符：&& || !
    - 关系运算符：== != > < >=
    - 三元运算符：number = (num == 10)?value1:value2 类似于number = value1 if num == number1 else value2
- 条件判断，if ,else if,else {}
- 选择判断，switch () {case val1: break }
- 循环操作：
    - for(init condition increment){} or for(double x: vector){}
    - while(condition){sentence }
    - continue 和 break
- 数组：
    - 定义数组：type name[length] 或者 type name[] = {value1, value2}
    - 多维数组：type name[length] [length1] 或者type name[][] = {{},{}}
    - 作为函数的输入以及输出
    - 数组多数情况下可以认为是地址，除了在sizeof中表示数组的长度,或者是数组不能自加
- 函数包括自定义以及内置函数：
    - return_type function_name(parameter_list) {}
    - 函数声明和函数定义(声明尽量在头文件中)
    - 内联函数主要对于一些简短的函数直接替换，可以像普通函数一样被调用，但是在调用时并不通过函数调用的机制而是通过将函数体直接插入调用处来实现的，这样可以大大减少由函数调用带来的开销
    - 默认参数的使用fun(int number, int a = 10,int b = 20)
    - 传入数组，传出数组，尽量少用C的形式，多用标准库的vector
    - **输入参数是非const的时候只能接受非const参数，而const没有这个限制，尽量多用const**
    - **函数传入形参的时候会自动产生副本,尽量使用引用&**
    - 函数重载是指同一个函数名可以定义多个但是要保证输入不尽相同
    - 返回引用时不能返回临时变量
    - **函数返回时除了返回引用，都会重新生成临时副本返回**
- 字符串：可以创建字符列表或者字符对象
    - string name1定义对象
    - 尽量多使用C++ string库，里面许多功能
- 指针：本质上就是存储某一类数据地址的变量
    - 野指针是指其指向的对象被收回，但是指针没有指向Null，空指针是指没有初始化的指针
    - 定义 int *p,var,*q **d;
    - 空指针nullptr, 空值null
    - 赋值 p = &var

    - 指针和数组，数组名字本质上就是一个指针，指向了数组的最开始地址，但是不能对数组++，也就是不可以改变数组指针，可以对指针++来循环数组里面的元素，也可以使用指针[n]
- 引用：引用是某个变量的一个别名,必须在创建的时候初始化
    - 定义 int &name = varible1
    - 引用相当于原始变量的别名，不占内存地址，与原变量是同一地址
    - 应用：引用作为参数，如果有大块的数据需要传递的话，采用指针，在Cpp中也可以采用引用。指针可读性差，用引用更方便。
    - 应用：如果需要提高引用的效率，又要保护不被改变，就应该使用常引用。const type &name = varible1 这样引用就不能修改，但是原本的变量可以被修改，尽量定义为const
    - **应用：引用作为返回值，在内存中不产生返回值的副本，type &fun(){} ,但是要求变量的引用不能是局部变量，因为局部变量在函数退出之后就会自动销毁。**
- 输入输出：cin cout 必须在std里面
- 数据结构：
    - struct type_name {} varible1定义,可以像类一样定义构造函数，直接写在结构内部
    - 调用varible1.name = N
    - 在定义和函数里面使用的时候type_name varible1
    - 在声明的时候加上;类似于类声明后面加上;
    - 列表初始化struct：Points p = {1,2,3}
- const
    - const主要是表示一旦定义完之后就不能改变，const type name = value，必须赋初值
    - const修饰指针变量及引用：
        - 指向常量的指针，表示变量不可改变const int* a = &b 表示a指向的变量不可改变
        - 常量指针，表示指针不可改变int const* a = &b int *const a = &b表示指针a不可改变只能指向x，x可以改变
    - **作为函数参数的const与变量定义相同 类中的成员函数后面加上const表示这个函数不会修改任何数据成员(对类来说)**
    - 函数前面加上const表示返回不可改变，返回引用变量的时候，最好加上const 不然会出现fun(a,b) = c能通过编译的情况
- new & delete
    - new来分配内存：主要表示是构造一个空间并且返回地址例如type* pointer = new type，可以用来动态使用内存
    - delete来释放内存：不然在程序的剩余部分可能会出现内存耗尽的问题
    - 对于数组来说，type* pointer = new type [n] ,delete [] pointer;
    - new对于对象来说会自动调用构造函数，而delete会调用析构函数
- new & malloc
    - new和delete配对使用，效率较高，对类初始化的时候会自动调用构造函数，delete的时候会自动析构
    - new不需要显式指定多大内存，而malloc需要显式指定多大内存
    - new直接返回对象所需要指针，而malloc返回指针需要进行强制类型转换

    - new从自由存储区分配内存，malloc从堆上分配内存
- static
    - 局部变量：保存在静态存储区，在离开函数之后仍然保存直到程序结束后被销毁
    - 全局变量：保存在静态存储区（普通变量保存在栈中）

Object Oriented Programming

- 特点：多态、继承、封装、接口
    - **接口，简单说就是public的方法，供外部使用，通过这些public的方法，可以操作内部数据，所以称之为接口。**
    - **封装，一个类是由数据与方法组成的，将数据和方法放在一起，就是封装，只需要关注外面功能，不需要关注细节.**
- 声明：class Name{public: private:} 定义函数type Name::fun(){}
- 对成员变量的访问控制：
    - public: 所有类
    - protected: 基类、派生类和友元
    - private: 基类和友元
    - 友元函数：可以访问类里面的所有变量，例如friend void disp(const &XXX obj) 需要在类里面声明
- 类自动定义：构造函数、复制构造函数、析构函数、赋值运算符、地址运算符
- 派生类不能继承：构造函数，析构函数，友元函数和赋值运算符
- 列表初始化type Name(para1,para2..)等价于 type Name = {para1,para2},这调用了相应的构造函数，而type Name = type(para1,para2)是先调用构造函数，在调用复制构造函数
- **构造函数**，类创建的时候会自动执行的函数，派生类创建的时候会自动调用默认基类构造函数
    - 不带参数的构造函数Line:Line(void)
    - 带参数的构造函数Line:Line(double len)，体现函数重载
    - 复制构造函数：（与赋值运算符区别，赋值运算符是已经有了这个变量之后进行的赋值）
        - 当一个类初始化等于同一个类的时候自动调用
        - 把对象传入函数或者从函数返回的时候会调用
    - this指针可以使用表示当前的对象，this->member,可以返回本身*this类似于self
    - 构造函数不能为虚函数
        - 虚函数对应一个虚函数表，可是这个vtable其实是存储在对象的内存空间的。问题出来了，如果构造函数是虚的，就需要通过vtable来调用，可是对象还没有实例化，也就是内存空间还没有，无法找到vtable，所以构造函数不能是虚函数。
    - 成员初始化列表
        - Name::Name(int a_, int b_): a(a_), b(b_){}
        - 只能用于构造函数
        - 必须用这种格式来初始化引用数据成员（因为引用与常量类似，必须初始化）

        - 必须用这种格式来初始化基类构造函数（或者使用默认基类构造函数）
        - **初始化顺序只与变量声明顺序有关**
- **析构函数**会自动创建，在删除类的自动调用，先调用派生类析构再调用基类析构
    - 定义Name::~Name(){}
    - 基类析构函数必须加上virtual，如果不加上的话就只会执行指针对应的基类析构函数，如果加上的话会先调用派生类析构函数再自动调用基类析构函数
- 继承基类：
    - 构造函数首先创建基类函数
    - 应该通过列表初始化来传递信息
    - 析构函数相反，先析构派生类在析构基类
    - 基类与派生类之间的关系：
        - 派生类可以使用基类的函数
        - 基类指针和引用可以直接指向派生类，但是只能调用基类方法
        - 不可以将派生类指针指向基类
    - **多态**：
        - 静态多态：函数重载和运算符的重载，定义函数表来绑定
        - 动态多态：虚函数+继承，使用指针或者引用的时候会自动根据对象类型来决定调用哪个函数，必须使用虚函数才会动态绑定，定义虚函数表
    - 调用基类方法可以使用基类::fun1()
    - 虚函数：来实现多态，在不同的子类里面来自动判断输出的类型
        - 基类和派生类前都加上virtual
        - 纯虚函数=0，表明这个类是抽象类不能单独实例化
        - 如果派生类函数与基类函数同名，但是参数不同。此时不论有没有virtual基类函数都被隐藏
        - 如果同名参数相同，没有virtual的时候基类函数被隐藏
        - 虚函数实现原理：在实例化的时候生成虚函数表：存储基类和派生类所有的函数，如果符合虚函数的标准就会将基类的函数地址替换成派生类的函数地址，从而实现多态
    - 派生类型：
        - public：基类的公共成员和保护成员都变成派生类的相应成员
        - protected：基类的公共成员和保护成员都变成派生类的保护成员
        - private：基类的公共成员和保护成员都变成派生类的私有成员
- 重载运算符：
    - 可以选择成员函数或者非成员函数（一般是友元函数）来定义
- 函数返回值的四种情况：
    - 返回const对象的引用：使用引用可以提高效率，因为它不会再创建副本，所以如果返回传递给它的对象可以使用引用。输入是const的时候输出也必须是const
    - 返回非const对象的引用：一般是在复制重载运算符或者<<运算符中使用
    - 返回对象：如果返回的对象是被调用函数中的局部变量，那必须直接返回对象
    - 返回const对象：防止出现fun(a,b) = c的情况

- 静态成员：
  - 表示无论创建多少个对象的时候静态成员都只有一个副本
  - 静态函数：跟类里面的成员没有关系，可以直接调用，class::fun()，但也不能使用非静态变量
  - 在类之外进行初始化，如果是const静态成员或者枚举类型可以在类里面初始化
- C++11的特性，auto类型推导-范围for循环
- 智能指针采用引用计数的智能指针，主要用于要将一个对象分配给多个指针，会自动删除

Standard Template Library (STL)

- STL抽象了4个部分，算法，容器，函数和迭代器
- 基本上所有都包含：
  - size()
  - begin()
  - end()
  - random_shuffle(a.begin(), a.end())
  - sort(a.begin(),a.end())

String

- 初始化，不同方式构造string，#include
- 方法：
  - find(s) rfind(s)
  - capacity() reserve()
  - 字符串拼接s1+s2或者s1.append(s2)
- 内存是分配一定内存，到了一定内存后自动增大2倍

Shared_ptr

- 用来解决new之后，忘记回收内存的问题
- shared_ptr name = new type(para1,para2);
- 使用引用计数，当所有的指针技术都为零的时候，将对象销毁

Vector

- 比数组方便很多的类似数组的对象,#include
- vector name ,vector name(n)
- auto ib = name.begin();可以认为是指向开始的指针
- push_back() 插入元素到最后
- 迭代for(auto iter = vec.begin() iter != vec.end() iter++){}

- 删除和插入时间在不同位置不同

Python

- generator-生成器
  - 是指生成了一个迭代对象，并不需要全部立即生成，生成迭代器
  - (i for i in range(10))
  - 包含yeild的函数是指这个函数是一个生成器，在遇到yeild的时候会返回，下次调用的时候从此处开始
- iterator-迭代器是一个更抽象的概念，对任何对象，如果它的类有next方法和iter方法返回自己本身既可以认为是一个迭代器
- 高级函数
  - **传递值还是传递引用**
    - 对于可变对象类似于字典和数组，传递引用也就是在函数里面可以直接更改原始值
    - 对于不可变对象类似于元祖字符串，复制新的值
  - zip：将多个可迭代对象组成一个元祖的列表
  - **装饰器**：在代码运行期间动态增加功能的方式叫做装饰器，本质上是一个返回函数的高阶函数
    - 定义装饰器函数：至少两层函数，返回装饰器
    - 使用装饰器：@log fun表示fun = log(fun)
    - 用于解决一类问题，实现代码复用，如增加日志，运行时间计数
- **闭包**：在一些语言中，在函数中可以（嵌套）定义另一个函数时，如果内部的函数引用了外部的函数的变量，则可能产生闭包。闭包可以用来在一个函数与一组“私有”变量之间创建关联关系。在给定函数被多次调用的过程中，这些私有变量能够保持其持久性。
- 垃圾回收机制（引用计数机制为主，分代回收和标记清除为辅的方法）
  - **引用计数原理**: 当数据的引用数变成0的时候,python解释器就认为这个数据是垃圾,进行垃圾回收,释放空间
    - 优点：简单实时
    - 缺点：保留对象引用会占用一点点空间，不能处理交叉引用的问题
  - 标记清除和分代回收不太了解
- 进程与线程
  - 线程是进程调度的最小单位，同一进程可以有多个线程
  - 进程：进程是互相独立的，而同一进程中的不同线程是共享空间
  - 进程之间的不同线程会互相影响，也可以直接通信，启动速度快，不同进程之间通信需要中间代理实现
  - 正常情况下多核可以多个线程同时处理，互相独立，但在python中因为有全局锁的存在，没办法实现真正意义的多线程，每个进程有独立的GIL，而在多进程中因为每个都有一个GIL可以实现真正的并行
- python2 和 python3区别

    - print,range()返回列表还是返回迭代器
    - ascii编码和utf-8
    - input和raw_input
- Python类
    - 访问限制：变量名之前加上两个下划线就表示变成了私有变量self.__name
    - 也可以有多态，不过是自动覆盖的，没有函数重载，对于不同类型的参数函数可以接受，对于不同参数数量的函数也可以接受
    - 既想直接使用属性，也可以自动设置属性的可以使用@property
    - *call*,将类定义为一个函数，可以直接调用f = Fool(); f(a,b,c)
    - ***new***,在实例化的时候先于_init_使用
        - _new_是控制是否生成新实例
        - _init_是决定了创建新实例之后如何创建的过程
        - 可以用于创建单例模式，_new_会执行多次，_init_执行多次
    - 创建类的两种方法：直接创建和利用元类也就是利用type直接创建
    - 也有析构函数_del_
    - 深拷贝和浅拷贝
    - **类方法，类实例方法，静态方法，类属性，类实例属性**
        - 实例方法：不能被类直接调用，可以被实例调用
        - 静态方法：加上staticmethod，不需要传入self，可以被实例和类直接调用
        - 类方法：加上classmethod，传入cls参数，可以被实例和类直接调用
        - 类属性：全部类共享
        - 实例属性：不能被类直接调用，如果在类外直接引用会产生一个同名的实例属性

知识点

- python 装饰器

```
def errorDetect(fun):
    @functools.wraps(fun)
    def wrapper(*args, **kwargs):
        print(fun.__name__+" is called.")
        return fun(*args, **kwargs)
    return wrapper
```

- python 单例模式

```python
class A(object):
    __instance = None
    def __new__(cls,*args,**kwargs):
        if cls.__instance is None:
            cls.__instance = object.__new__(cls)
        return cls.__instance
```

Niuke

- static关键字的作用
- C语言与C++区别
- C++有几种类型转换(const_cast, static_cast, dynamic_cast，reinterpret_cast)
  - static_cast用于确定类型的基本类型转换，发生在编译阶段，也可用于父类指针与子类指针之间的转换，向上转换时即子类到父类安全，向下转换不一定安全
  - dynamic_cast在运行时动态转换指针和引用(前提是类里面必须包含虚函数)，向上转换一定是成功的，向下转换只有在原本的基类指针指向派生类对象的时候才会转换成功
  - reinterpret_cast用于指针类型之间的转换
- 指针、引用和数组的区别
- 智能指针shared_ptr
- 析构函数为什么需要是虚函数？默认不是？
- 析构函数和构造函数的作用？
- 重载，隐藏，覆盖的区别？
- 虚函数和多态，虚函数原理？
- new/delete和malloc/free的区别
- auto关键字，for_each循环，初始化列表，智能指针
- STL：vector扩容原理：以原大小的两倍配置一份新空间，将原空间数据拷贝过来
- STL：vector、list底层-不连续的双端链表、map和set基于红黑树、无序map和set基于哈希表
- resize和reserve的区别，resize是变化vector的size，reserve是改变最大容量
- move不需要拷贝，直接修改对象的所有权到vector，emplace直接传入所需参数，而不需要构造一个实例之后再调用复制构造函数，效率较高