编程通用入门及其原理

1. 术语与名词
2. 专有名词/术语
3. 关键字
4. 命名规则
5. 个人推荐的IDE
6. 实现与结构
7. 通用逻辑结构
8. 算法简介
9. 一些小知识
10. 高效编译/运行原则
11. 优化与可读性
12. 预先处理
13. 后期处理

Last Right 231021

并非专业编写，可能有遗漏以及个人主观意见

# 专有名词其一

* **类型：**根据对象定义的内存属性不同，将不同对象所属称作类型
* **语言：**将一种行为（非空集）转换为另一种行为（非空集）的映射，称作语言
* **语素：**在编译器分析代码时会将代码分解为独立的词语，这些词语被称作语素
* **关键字：由**编译器与语言标准默认保留以用作语法语素的词语
* **编译：**以c为代表的编译型语言需要转换为可执行文件后才可运行，这一转换行为称作编译；以java为代表的解释型语言在代码文本更新后在托管目标（java为java虚拟机，c#为.NET框架）上生成DLL或可执行文件，这一行为称作编译
* **ub(undefine behavior)：**未定义行为，它们是语言标准未定义的产物（通常是因为硬件原因），这是一种很值得关注的行为，未定义行为并不是代表着这一行为是随机的，而是指在不同环境或编译器下工作或生成的程序会发生不同的行为甚至无法工作的现象
* **时间界：略**
* **算法：略**
* **基本类型：**由编译器和语言标准定义的类型称为基本类型
* **类（Class）与结构（Struct）：**一些高级语言用于声明或定义类型时使用的关键字，在不同的语言中可能有着不同的特性，如c/c++中Class定义的类型中的成员默认私有而Struct默认公开
* **声明（变量）：**声明是形同**Type arg;**形式的语句，它表示存在一个Type类型arg名称的变量，但是如果这不是基础类型，则声明不会分配真实的对象和内存，随后不初始化该变量便访问它将会报错
* **定义（变量）：**定义是形同**Type arg=0;**形式的语句，它等同于声明并立刻初始化该变量
* **声明（函数或类）：**声明是形同**void foo();或class base;**的语句，它表示存在这样一条函数或是类型，你可以使用这样一条函数或定义该类型指针而不需要考虑其具体实现，它将在其他位置被定义
* **定义（函数或类）：**略
* **指针：**指针是一种指向内存地址的标识符，一般而言语言中拥有指针则意味着允许程序员直接操作可访问的内存地址，但是也带来了内存泄漏及越界访问等隐患。我们一般使用这样的表述来形容一个指针的行为

**Type\* arg; //arg是一个指向Type类型的指针**

注意，指针本身也是一种类型，是某一长度的整形，因此指针可以被强制或隐式转换为整型并进行数值运算。其具体长度根据系统及编译器而定，其指向的内存地址通常用0x开头的十六进制表示。

* **句柄：**句柄的本质是指针或整形，是一种用于标识某个资源或对象的唯一标识符
* **栈：**程序运行时自动分配和释放的一块内存区域，用于存储局部变量、方法参数、返回地址和临时数据等。栈内存的管理是由程序运行时系统自动完成的，因此无需手动申请或释放。每个线程都有一个独立的栈，栈的大小是固定的
* **堆：**程序运行时动态分配和释放的一块内存区域，用于存储对象和数组等。堆内存的管理是由程序员手动申请和释放的，需要使用 new 运算符来创建对象。堆内存的大小是不固定的，当堆空间不足时，程序会自动进行垃圾回收（GC）
* **垃圾回收：**不再使用的内存空间称为垃圾，垃圾回收是一种内存管理机制
* **线程：**程序流工作时占用CPU的一条路线，起始程序打开的线程称为主线程
* **引用：**在java等语言中引用是一种变量，它储存了对象在堆上的内存地址；在c/c++中引用是指针的别名，可以当作指针的一种表达式
* **解引用/取址：**在c/c++中使用“&”来获取变量的地址
* **生命周期：**变量存在的程序部分称为变量的生命周期，在此之外，变量的内存地址上不存在该变量
* **优先级：**在以后的学习中你们将多次遇到这个词，这个词将代表着程序的执行顺序
* **虚拟：**将函数或字段虚拟化，可以使得子类能够重载
* **抽象：**一指函数或字段的完全虚拟化，其本身将不进行任何实现而是仅仅声明；二指提取某些事物或对象的公共特征
* **重载：**重新实现重载的目标，在以父类类型情况下调用子类对象时实际调用或获取的是重载函数或重载字段
* **父类/基类：**被其他类继承
* **子类/派生类：**继承自其他类
* **协变：**将子类引用或指针赋给父类引用或指针（内存上对象不变）
* **逆变：**将父类引用或指针赋给子类引用或指针（不安全，安全性依赖于程序设计）
* **转换：**某些类型可以通过某种方法转变为另一类型（生成新对象），这种行为称作转换，并且以不同的转换形式分为强制转换与隐式转换
* **模板：**略
* **静态：**静态类是一种无法实例化的类，如java主类；静态函数是一种实际上不依赖于类实例的函数，可以通过类型直接调用；静态成员是一种实际上不依赖于类实例的成员，可以通过类型直接获取
* **虚函数表：**略
* **副作用：**代码的唯一作用是给人看，其余作用都是副作用（既生成程序是代码的副作用）
* **表达式：**表达式代表着一种操作，例如a+=10一类最为常见；不太容易被提起的还有逗号表达式等
* **截断：**如浮点型向整形转换时会丢弃小数，这种丢弃现象称为截断
* **按值传递/按引用传递：**这是两种传递参数的方式，按值传递会产生新对象，按引用传递会将实例的内存地址转进
* **函数帧栈：**进入函数范围时会创建函数帧栈以保存函数中的变量及函数环境，这一过程是函数的常量性能损耗，当函数中语句较少时应考虑编译器优化或inline
* **异常处理：**当程序出现不应该出现的情况时会抛出异常或运行错误，能够处理异常的程序称为拥有异常处理机制
* **IO：**输入输出
* **左值：**起初指代能放在赋值运算符左侧的变量，随着语法发展出现了一些变化，左值的本质是能够被操作的变量，通过参数传递可以使得一个右值转变为一个左值。注意：可以将**有名称变量**都认作左值
* **右值：**起初指代能放在赋值运算符右侧的变量，一般为匿名变量，右值变量会在使用结束后（离开生存周期）被立刻释放，因此取址右值是非法行为，通过某些手段可以强制将左值转换为右值，那么在使用该变量后原变量的内存地址上将变为一块非法内存
* **void（空）：**虽然是空类型，但是依然携带有一定的信息，这取决于语言标准
* **构造函数：**负责对象创建时的初始化
* **构析函数：**负责对象销毁时最后的资源释放工作
* **深拷贝：**将一个对象完整的拷贝到另一个对象上，改变其中一个变量不会影响到另一个变量
* **浅拷贝：**将一个对象不完整的拷贝到另一个对象上，并将剩余的成员以右值形式移动到目标对象，这通常发生在移动构造函数与移动赋值函数，目的是减少生成新的对象，因为被拷贝对象可能是一个右值或即将离开生命周期
* **程序入口：**最简单的情况下，在java中是主类的公开静态main函数，在c/c++中为第一个声明或定义为main名称的变量或函数
* **重定向：**通过某种方法将一个变量所指变更为一个新的对象，以实现新的功能，通常发生在实现了多态的环境下父类指针指向不同的子类对象并调用其重载函数，少数情况下是对void\*或语言基类型（c#为object）进行记号符检测并调用或反射
* **转发：**将资源或操作从一处位置转移到另一处
* **委托：**通过封装函数或使用函数指针，或使用语言特性等手段声明一个可以重定向具体函数的类型，这个类型称为委托；将一个具体函数赋值给一个委托类型，这个过程称为委托
* **托管：**将具体实现交付另一实现过程
* **函数(function)：**编程中的函数不完全是数学意义上的函数，特征是能够输入输出，拥有特定的生命周期
* **方法(method)：**这是一个java原创的术语，和函数没有任何差别（毫无意义的重命名）
* **字面量：**如1，“name”，这样在代码中声明的临时变量，它们只能是右值
* **向下转换/向上转换：**对应协变/逆变操作，或者基础类型的转换，如int64向int32赋值为向上转换，float向double赋值为向下转换，取决于表示数的模型和内存分配哪个更“大”更“宽”
* **断言：**编译期调试手段，略
* **溢出：**假设我们有int2（占两个bit的整形）和一个int4（占四个bit的整形），将int4变量赋给int2变量将会发生溢出，这很可能导致信息损失（2bit无法容纳4bit表示的信息，具体的情况是ub）

# 专有名词其二

1. **API**

应用程序接口（英语：Application Programming Interface，简称：API），又称为应用编程接口，就是软件系统不同组成部分衔接的约定。由于近年来软件的规模日益庞大，常常需要把复杂的系统划分成小的组成部分，编程接口的设计十分重要。程序设计的实践中，编程接口的设计首先要使软件系统的职责得到合理划分。良好的接口设计可以降低系统各部分的相互依赖，提高组成单元的内聚性，降低组成单元间的耦合程度，从而提高系统的维护性和扩展性。

1. **ACID**

ACID，是指数据库管理系统（DBMS）在写入或更新资料的过程中，为保证事务（transaction）是正确可靠的，所必须具备的四个特性：原子性（atomicity，或称不可分割性）、一致性（consistency）、隔离性（isolation，又称独立性）、持久性（durability）。

1. **AJAX**

AJAX即“Asynchronous JavaScript and XML”（异步的 JavaScript 与 XML 技术），指的是一套综合了多项技术的浏览器端网页开发技术。

1. **CAS**

比较并交换(compare and swap, CAS)，是原子操作的一种，可用于在多线程编程中实现不被打断的数据交换操作，从而避免多线程同时改写某一数据时由于执行顺序不确定性以及中断的不可预知性产生的数据不一致问题。 该操作通过将内存中的值与指定数据进行比较，当数值一样时将内存中的数据替换为新的值。

集中式认证服务（英语：Central Authentication Service，缩写CAS）是一种针对万维网的单点登录协议。它的目的是允许一个用户访问多个应用程序，而只需提供一次凭证（如用户名和密码）。它还允许web应用程序在没有获得用户的安全凭据（如密码）的情况下对用户进行身份验证。“CAS”也指实现了该协议的软件包。

1. **JPA**

JPA 是 Java Persistence API 的简称，中文名 Java 持久层 API，是 JDK 5.0 注解或 XML 描述对象－关系表的映射关系，并将运行期的实体对象持久化到数据库中。

1. **JSON**

JSON（JavaScript Object Notation）是一种轻量级的数据交换语言，以文字为基础，且易于让人阅读。尽管 JSON 是 Javascript 的一个子集，但JSON是独立于语言的文本格式，并且采用了类似于 C语言 家族的一些习惯。

1. **POJO**

POJO（Plain Ordinary Java Object）简单的 Java 对象，实际就是普通 Java Beans。使用 POJO 名称是为了避免和 EJB 混淆起来，而且简称比较直接。其中有一些属性及其 getter setter 方法的类，没有业务逻辑，有时可以作为VO(Value Object) 或 DTO(Data Transform Object) 来使用。当然，如果你有一个简单的运算属性也是可以的，但不允许有业务方法，也不能携带有 connection 之类的方法。

1. **DSL**

领域专用语言（Domain Specific Language/DSL），其基本思想是「求专不求全」，不像通用目的语言那样目标范围涵盖一切软件问题，而是专门针对某一特定问题的计算机语言。

1. **GC**

在计算机科学中，垃圾回收（英语：Garbage Collection，缩写为GC）是一种自动的内存管理机制。当一个电脑上的动态内存不再需要时，就应该予以释放，以让出内存，这种内存资源管理，称为垃圾回收。垃圾回收器可以让程序员减轻许多负担，也减少程序员犯错的机会。垃圾回收最早起源于LISP语言。目前许多语言如 Smalltalk、Java、C# 和 D 语言都支持垃圾回收器。

1. **DML**

数据操纵语言（Data Manipulation Language, DML）是 SQL 语言中，负责对数据库对象运行数据访问工作的指令集，以 INSERT、UPDATE、DELETE 三种指令为核心，分别代表插入、更新与删除，是开发以数据为中心的应用程序必定会使用到的指令，因此有很多开发人员都把加上SQL的SELECT语句的四大指令以“CRUD”来称呼。

1. **DDL**

数据定义语言（Data Definition Language，DDL）是 SQL 语言集中负责数据结构定义与数据库对象定义的语言，由 CREATE、ALTER 与 DROP 三个语法所组成，最早是由Codasyl（Conference on Data Systems Languages）数据模型开始，现在被纳入 SQL 指令中作为其中一个子集。

1. **DI**

Dependency Injection，依赖注入。在软件工程中，依赖注入是种实现控制反转用于解决依赖性设计模式。一个依赖关系指的是可被利用的一种对象（即服务提供端） 。依赖注入是将所依赖的传递给将使用的从属对象（即客户端）。该服务是将会变成客户端的状态的一部分。 传递服务给客户端，而非允许客户端来建立或寻找服务，是本设计模式的基本要求。

1. **DNS**

域名系统（英文：Domain Name System）是互联网的一项服务。它作为将域名和IP地址相互映射的一个分布式数据库，能够使人更方便地访问互联网。DNS使用TCP和UDP端口53。当前，对于每一级域名长度的限制是63个字符，域名总长度则不能超过253个字符。

1. **GUI**

图形用户界面（Graphical User Interface）是指采用图形方式显示的计算机操作用户界面。与早期计算机使用的命令行界面相比，图形界面对于用户来说在视觉上更易于接受。

1. **HTTP**

超文本传输协议（英文：HyperText Transfer ProtocolP）是一种用于分布式、协作式和超媒体信息系统的应用层协议。HTTP是万维网的数据通信的基础。

1. **IOC**

控制反转（Inversion of Control，缩写为IoC），是面向对象编程中的一种设计原则，可以用来减低计算机代码之间的耦合度。其中最常见的方式叫做依赖注入（Dependency Injection，简称DI），还有一种方式叫“依赖查找”（Dependency Lookup）。通过控制反转，对象在被创建的时候，由一个调控系统内所有对象的外界实体，将其所依赖的对象的引用传递给它。也可以说，依赖被注入到对象中。

1. **JWT**

JSON Web Token（JWT）是一个非常轻巧的规范。这个规范允许我们使用JWT在用户和服务器之间传递安全可靠的信息，特别适用于分布式站点的单点登录（SSO）场景。

1. **LDAP**

轻型目录存取协定（英文：Lightweight Directory Access Protocol）是一个开放的，中立的，工业标准的应用协议，通过IP协议提供访问控制和维护分布式信息的目录信息。

1. **MVC**

MVC模式（Model–view–controller）是软件工程中的一种软件架构模式，把软件系统分为三个基本部分：模型（Model）、视图（View）和控制器（Controller）。MVC 模式的目的是实现一种动态的程序设计，使后续对程序的修改和扩展简化，并且使程序某一部分的重复利用成为可能。除此之外，此模式通过对复杂度的简化，使程序结构更加直观。

PS：我个人编写的unity c#抽象框架([NINEMINEsigma/AlphaDiagram0.4: A Unity Project (github.com)](https://github.com/NINEMINEsigma/AlphaDiagram0.4) / [NINEMINEsigma/AlphaDiagram0.4.2: AlphaDiagram (github.com)](https://github.com/NINEMINEsigma/AlphaDiagram0.4.2))就是MVC+接口倒置的抽象架构

1. **MVP**

Model-view-presenter，简称MVP，是电脑软件设计工程中一种对针对MVC模式，再审议后所延伸提出的一种软件设计模式。被广范用于便捷自动化单元测试和在呈现逻辑中改良分离关注点（separation of concerns）。

1. **MVVM**

MVVM（Model–view–viewmodel）是一种软件架构模式，有助于将图形用户界面的开发与业务逻辑或后端逻辑（数据模型）的开发分离开来，这是通过置标语言或 GUI 代码实现的。

1. **OLAP**

联机分析处理（英语：On-Line Analytical Processing），是一套以多维度方式分析数据，而能弹性地提供积存（英语：Roll-up）、下钻（英语：Drill-down）、和透视分析（英语：pivot）等操作，呈现集成性决策信息的方法，多用于决策支持系统、商务智能或数据仓库。其主要的功能，在于方便大规模数据分析及统计计算，对决策提供参考和支持。与之相区别的是联机交易处理（OLTP）。

1. **SQL**

SQL（结构化查询语言）是一种特定目的程序语言，用于管理关系数据库管理系统（RDBMS），或在关系流数据管理系统（RDSMS）中进行流处理。

1. **SPA**

单页 Web 应用（single page web application），就是只有一张 Web 页面的应用，是加载单个 HTML 页面并在用户与应用程序交互时动态更新该页面的 Web 应用程序。

1. **SOA**

面向服务的体系结构（英语：service-oriented architecture）并不特指一种技术，而是一种分散式运算的软件设计方法。软件的部分组件(呼叫者)，可以透过网络上的通用协定呼叫另一个应用软件元件执行、运作，让呼叫者获得服务。SOA原则上采用开放标准、与软件资源进行交互并采用表示的标准方式。因此应能跨越厂商、产品与技术。一项服务应视为一个独立的功能单元，可以远端存取并独立执行与更新，例如在线上线查询信用卡账单。

1. **SOAP**

SOAP（原为Simple Object Access Protocol的首字母缩写，即简单对象访问协议）是交换数据的一种协议规范，使用在计算机网络Web服务（web service）中，交换带结构信息。SOAP为了简化网页服务器（Web Server）从XML数据库中提取数据时，节省去格式化页面时间，以及不同应用程序之间按照HTTP通信协议，遵从XML格式执行资料互换，使其抽象于语言实现、平台和硬件。

1. **NoSQL**

NoSQL 是对不同于传统的关系数据库的数据库管理系统的统称。

1. **XML**

可扩展标记语言（英语：Extensible Markup Language，简称：XML），是一种标记语言。标记指计算机所能理解的信息符号，通过此种标记，计算机之间可以处理包含各种信息的文章等。如何定义这些标记，既可以选择国际通用的标记语言，比如HTML，也可以使用像XML这样由相关人士自由决定的标记语言，这就是语言的可扩展性。XML是从标准通用标记语言（SGML）中简化修改出来的。它主要用到的有可扩展标记语言、可扩展样式语言（XSL）、XBRL和XPath等。

# 关键字（以c++与c#举例）

1. int 整形 一般为int16,int32,int64或其他具体整形的别名
2. float 单精度浮点数
3. double 双精度浮点数
4. unsigned 无符号整形前缀（非负数）
5. long 整形，至少比int长
6. long long 整形，至少比long长
7. short/char 整形，至多与int一样长
8. enum 枚举
9. class 类
10. struct 结构
11. interface 接口
12. try 异常处理-检测区域
13. catch 异常处理-处理区域
14. finally 异常处理-后处理区域
15. static 静态
16. public 公开
17. private 私有
18. protected 保护
19. typeid/typeof 类型获取
20. using（指令）引入命名空间
21. using（声明）声明类型别名或创建新类型
22. const/readonly 常量/只读
23. new 创建对象
24. delete 销毁指针所指对象
25. malloc （c语言）分配指定空间并返回void\*指针
26. null/nullptr 空指针
27. 还有更多关键字，此处不再列举

# 关于命名的准则

1. 单下划线与双下划线一般表示私有或保护，保留
2. 双驼峰（java/c#标准风格）NameForm 可读性好，美观
3. 单驼峰（c/c++标准风格） nameForm 也常被各种语言的临时变量用作命名
4. 缩进，如

**void foo()**

**{**

**}**

的写法属于c风格，如

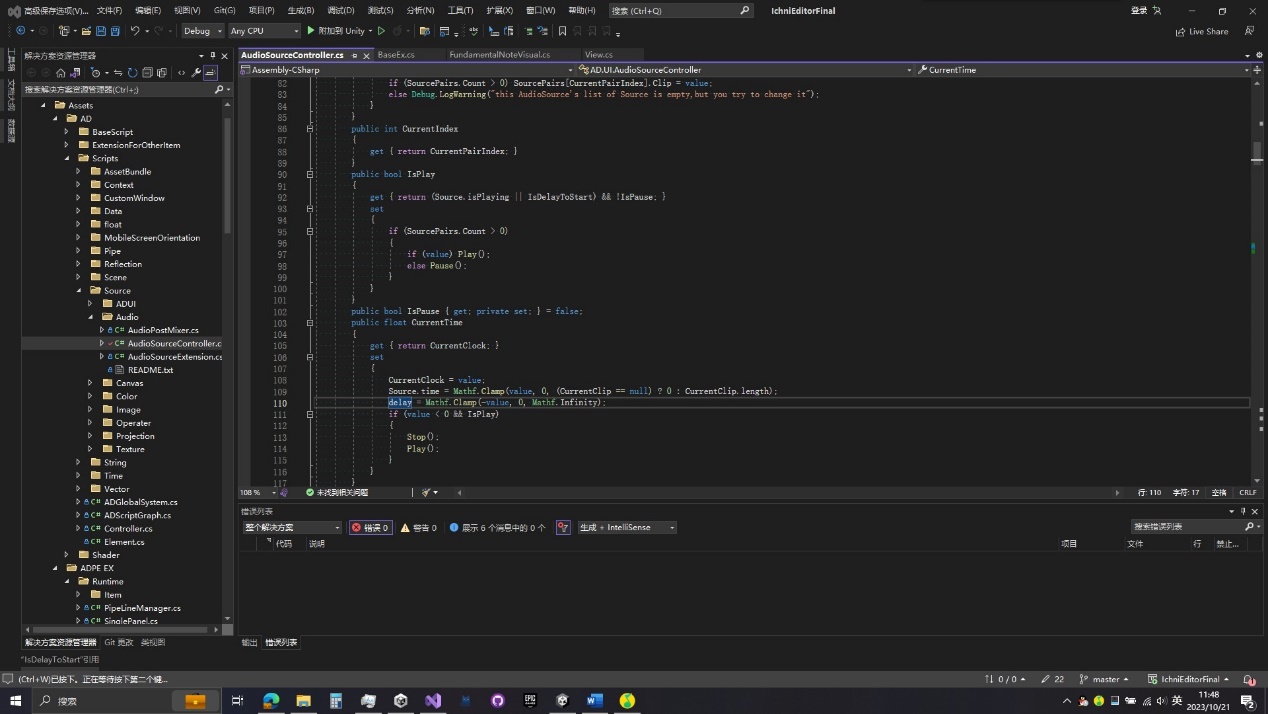
**void foo(){**

**}**

属于java风格

1. 临时变量命名，通常使用temp（temporary）作为一种公知的命名方法，每个人通常也有自己常用的其他的临时变量名称
2. 迭代器命名，在当前生命周期内的最外层迭代器命名为i（iterator），其次为j,k…以此类推

# Microsoft Visual Studio (VS)

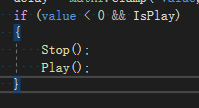
这是一款重工业级的大型集成开发环境，支持所有主流语言及主流开发工具的标准模块（需要下载对应的模块）

傻瓜式下载，傻瓜式安装（别装系统盘上就行，不然就真傻瓜了）。强大且通用，不需要下载一堆插件就比很多编译器（如spring4s）下了一堆插件还好用（PS：缺点很明显，VS很笨重，我的VS2022自身就占80G了）

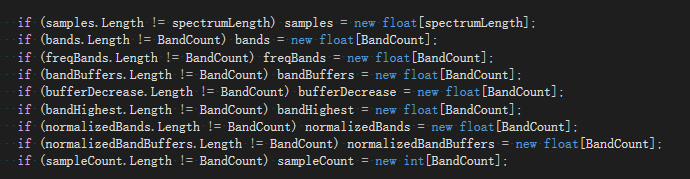
# Microsoft Visual Studio Code (VC)

VS的轻量化版本，本身已经脱离IDE的层级，单纯且简洁的文本编辑器，依赖于插件来实现编译等功能，便于携带（装进U盘）

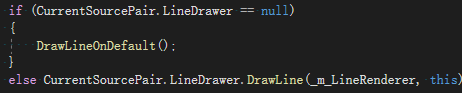
# 逻辑语句与结构



if逻辑语句，分为**布尔表达式**与**逻辑块**。如图中value<0&&IsPlay为布尔表达式，当布尔表达式为真时执行逻辑块；如图中{}中的内容为逻辑块，逻辑块内拥有一层新的生命周期，其中创建的栈上的变量会在离开逻辑块时销毁

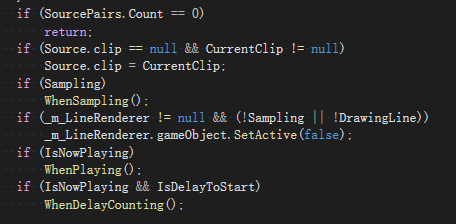


连续if，满足条件便会执行逻辑块，与其他if无关



if-else布尔表达式为真则执行if紧接着的逻辑块，否则执行else紧接着的逻辑块

if语句解析



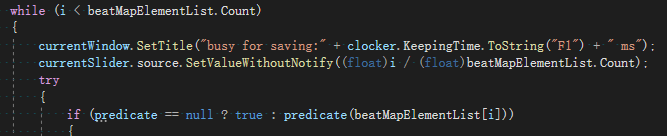
当SourcePairs容器为空（容纳对象数量为0）时立刻退出函数

当Source的成员clip为空且CurrentClip不为空时将CurrentClip赋值给Source.clip

当Sampling为真时执行WhenSampling函数

根据优先级（当前绝大多数语言优先级继承自c语言，参考[C语言运算符优先级（超详细）-CSDN博客](https://blog.csdn.net/yuliying/article/details/72898132)），我们判断\_m\_LineRenderer非空判断，为真才尝试继续判断，在此之后根据括号表达式的优先级，!Sampling||!DrawingLine被计算，其中!优先级比||高，因此先对Sampling与DrawingLine取非，然后逻辑或得到右侧表达式结果，其结果与左侧表达式进行逻辑和

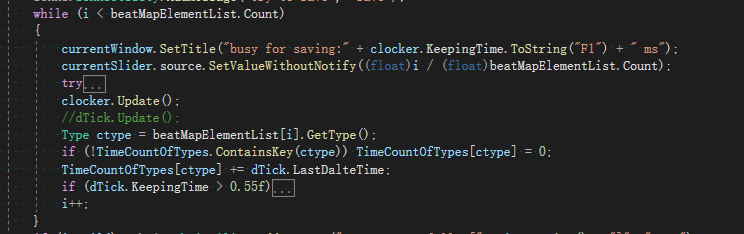
以下两句略



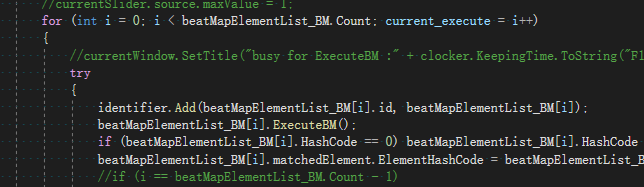
while循环（入口），分为**布尔表达式**和**逻辑块**，此处布尔表达式也被称作**循环不变式**，当其为真时，才会进入逻辑块，持续为真时会不断的重复运行逻辑块内的内容，只要有一次为假则退出循环

do-while循环（出口），与入口while循环相似，但是保证循环至少被运行一次

while循环解析



当变量i始终小于容器的Count成员变量时，逻辑块将持续运行，并在末尾将变量i增加



for循环，分为**初始化**，**布尔表达式**，**结束**，**逻辑块**，使用分号间隔，除了布尔表达式以外都可以使用分号表达式空置，进入for循环时会进行初始化，在循环开始前判断布尔表达式，执行与while相同的内容后，在单次循环结束后执行结束语句，注意，结束语句可以是一个表达式，一种非常经典的写法是

**for(int i=0,e=length;i<e;function(i++));**

这样一条语句展示的是初始化i与e变量后，以i<e作为循环不变式，以分号表达式作为循环的内容（空置），以function作为结束语句并且自增i，其效果与

**for(int i=0,e=length;i<e;i++)**

**Function(i);**

或是

**for(int i=0,e=length;i<e;)**

**Function(i++);**

等同

复合语句详解（java）

**System.out.print(new Scanner(System.in).nextInt()>0?”+”:”-”)**

通过不同的颜色，表示出了该语句中的解析层级，首先引用了System.in作为初始化Scanner类型对象构造函数的参数，在对象实例化后调用了nextInt函数，该函数返回一个int右值，并通过三元运算符得到一个字符串字面量，并且作为System.out.print函数参数传入，最终调用System.out.print函数打印结果，该语句等同于以下多个语句

**Scanner input=new Scanner(System.in);**

**int a=input.nextInt();**

**string result=”-”;**

**if(a>0) result=”+”;**

**System.out.print(result);**

复合语句更加简洁，并且减少了变量的定义与声明，隐藏了无用的细节，聪明点的已经可以意识到10月20日的6个作业其实有5个只用寥寥几行就可以完成，有兴趣的可以回去尝试重新编写判断奇偶的任务

# 算法简介（见附件）

# 在物理层面深入浅出

类的对象在物理内存上并不一定是连续的，若非特殊指定，这样一个对象中的成员和隐藏的数据很可能分散的储存在各个地方，因此在内存层面进行的强转是极其不安全的行为。

类型包含着此类对象应占有多少内存空间，函数表应如何构造，进行怎样的内存对齐方式等类型信息，

# 在设计层面深入浅出

直接声明一串变量与函数，如

**int a=0;**

**int b=0;**

**int add(int a,int b)**

**{**

**return a+b;**

**}**

与声明一个类，如

**class base**

**{**

**public:int a=0;**

**public:int b=0;**

**public int add(int a,int b)**

**{**

**return a+b;**

**}**

**}**

其本质的差异不大，类对象一样只是一串数据，唯一多的就是一张成员函数表，这张表上存放着指向类成员函数的指针，在调用对象的成员函数时，函数生命周期内会自动拥有访问此对象成员（包括私有）的权利，同时自动声明一个名称为this的变量指向/引用此对象

类的静态成员和静态成员函数并不简单的存在类内，它们实际的存在形式更接近全局变量和全局下定义的函数，拥有着静态的生命周期（直到程序退出才离开它们的生命周期）

# 关于重载运算符/操作符（operator）

能够重载多少运算符取决于语言，基本都能归属于以下几类

* 一般运算符
* 复合运算符
* 类外运算符（类外定义是一种扩展，此处略）

一般运算符总是包含了最基本几个的二元运算符：+-\*/%

对于类型**Type**它们通常声明如下（此处举例c#，这里不举例java的原因是java的重载受限过大）

**Type operator+(Type Right);**

**Type operator-(Type Right);**

**Type operator\*(Type Right);**

**Type operator/(Type Right);**

**Type operator%(Type Right);**

复合运算符一般（最好的情况下应该描述为**总是**，但是不免不了有些人喜欢写迷惑行为）是一般运算符与其他运算的复合版本，如

**Type operator+=(Type Right)**

的实现通常为

**Type operator+=(Type Right)**

**{**

**this.data=this.data+Right.data;**

**return this;**

**}**

如此以保证语句不存在意外的歧义同时保证函数的耦合性。

到此为止各位应该已经能理解两种自增与两种自减运算符的具体实现了，先增/先减仅仅只是增加/减少自己后直接返回，后增/后减则是在增加/减少自己前创建一个临时变量并以自身赋值给它，最后返回这个临时变量

# 编译器语素的一些默认解析

1. 如**“string“**这样的一个字符串字面量通常被翻译为一个char类型的数组，根据不同语言也会有不同的情况，如c#的基础类型包含了字符串类型**System.String**，那么将会被直接构造为字符串类型，同时使用池化技术减少相同字符串字面量可能需要的额外空间
2. 如**11**，**55**这样的整形字面量的类型通常为int，所以当书写一个过于巨大的数，如1145141919191919000时，会发生**溢出**，应当根据语言标准特别声明更大类型的整形字面量
3. 如**0.5**这样的浮点型字面量的类型通常为double，double通常也是计算机浮点数运算最快的类型（但是如c++中同样提供了fastfloat，fastdouble等更加明确的类型），也会存在溢出问题
4. 从以上已经可以察觉，1/2并不是0.5，因此如何声明一个字面量应该慎重，减少不必要的字面量，将其转为只读常量可以极大的减少这类问题

# 如何高效的编写程序，同时让程序高效的运行

**非常简单，相信编译器的优化，你能考虑的细枝末节问题顶多都是上个世纪就玩剩下的，要么就是你写的逻辑和语法本来就有问题**

很多情况下这都能从编译器生成的汇编码中得到答案和证明，如某些临时变量的内联等，记住，代码是人看的，关键在于简洁和明确

随着IDE越来越方便，我们已经不像以前的人一样使用极其简短的被戏称为竞赛写法的字母命名，能通过名称表明它的意义和行为是一种优雅的艺术