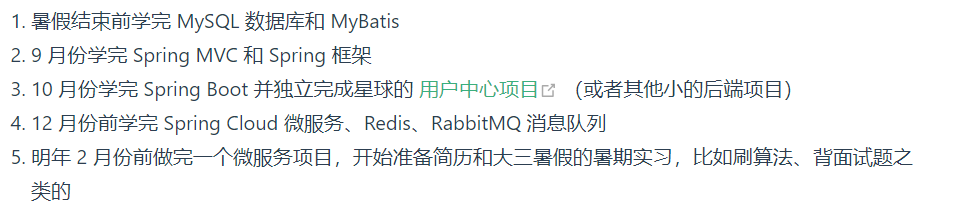
# 一．Plan hou

1. **核心竞争力**
   1. **硬实力：竞赛奖项（算法，软件杯），项目经验**
   2. **软实力：学习能力，沟通能力，团队合作能力，领导力，英语口语**
2. **学习原则：目的导向，主次分明，制作大纲，逐个击破**
3. **项目经验**
   1. **课程作业项目**
   2. **学校研究项目（了解，获取相关的信息）**
   3. **学生组织项目**
   4. **自我发起项目**
      1. **写文档**
      2. **托管到github上**
      3. **让合作伙伴写推荐信**
   5. **竞赛项目**
   6. **实习项目**
   7. **用github跟踪和学习项目**
4. **算法**
   1. **刷板子，转平台，**
   2. **找题库，好的题解**
5. **一些自己无法完成的项目，可以学习相关的东西**
   1. **学会使用GitHub，StackOverflow，慢慢了解相关的博客，开始写博客**
   2. **github上找项目开始学习**
6. Git,Github学习
7. **操作系统，计算机网络**
   1. **操作系统： 重装系统**
      1. **Windows系统，软件生态多，易于玩游戏，喜欢搞升级，升级毛病多，安装编程环境麻烦，可能在关键时刻死机**
      2. **Macos系统：中毒，死机少，mac软件很多不能和win比，软件问题，太贵了，前端开发，苹果客户开发**
      3. **Linux系统：开源，安全，可定制，底层设计，原理可以**
      4. **不知项目，租云服务器(便宜)，xshell运行linux,**
   2. **计算机网络，学习网路知识**
8. **学习前端，后端知识，学会维护网站**
9. **AIGC行业：未来发展路线**
   1. **短视频脚本**
10. **政策相关**
    1. **了解国家政策，以及规划**
    2. **了解行业政策**
    3. **了解学校的一切信息**
11. **了解一些国产AI，和国外的对比，还有一个openAI工作者，手写chatgpt的视频对应源码学习** 
    1. **小模型**
       1. **微软开始制作小模型**
12. **玩转coze，是一个开源平台，可以搭建自己的AI机器人模型**
13. **玩通电脑：**
    1. **解决电脑升级问题 finish**
    2. **解决edge浏览器和chrome浏览器更新问题**
    3. **了解清理电脑的软件**
    4. **重装系统**
    5. **坚持学习ytb对应内容**
14. **self summary;将自己任何时候总结的观点写入记录下来，需要时间形成自己的方法论，不断内化**
15. **iphone与安卓**
    1. **应用生态，软件更好**
16. **Bat种子解决，迅雷下载使用，以后看东西直接下载种子,bat种子下载犯法的**
17. **一些流氓软件，对应解决方案，**
    1. **各种网盘，百度网盘，夸克网盘下载速度缓慢**
18. **获取关于找工作的所有信息，**
    1. **职员内推**
19. **自媒体营销策略**
    1. **自媒体分享渠道，分享一个产品或平台，可以增加在搜索引擎中的排名（SEO）搜索引擎优化**
    2. **增加曝光度，积累信任**
20. **计算机领域的相关信息**

****

# **二.5.31~8.31**

1. **C++深度学习**
   1. **语法**
   2. **进阶：c++ primer ，Effective c++;**
   3. **计算机基础知识：** C++ 语言本身相对更底层，常用于系统开发、嵌入式、高性能计算等领域。因此 C++ 开发者相对于学 Java 或其他方向的同学来说，需要具备更扎实、更深入的计算机基础能力。
      1. **计算机导论（基本概念）**
      2. **数据结构和算法 （每天2小时左右）**
      3. **操作系统 （每天2小时左右）**
      4. **计算机网络（网课）**
      5. **计算机组成原理**
   4. **STL源码剖析**
   5. **Linux服务器编程：**对 C++ 开发来说，Linux 系统是重中之重。掌握 Linux 可以帮你更好地理解系统底层的运行机制，比如进程管理、内存管理、文件系统、网络传输等。
   6. **UNIX网络编程**
   7. **软件开发通用：**除了 Linux 之外，软件开发的其他通用技能都可以融入到日常学习中，而不是花一大段连续时间去专攻。比如企业项目研发流程，读一遍有个印象就好；Git & GitHub 需要在平时做项目、用开源项目时多多使用；23 种主流的设计模式可以每天学 2 - 3 个；软件工程的专业知识大学科班一般都会教，学校不教的话作为一个课外知识去补充即可。
      1. **企业项目开发流程**
      2. **Git & github**
      3. **Linux系统：**
      4. **设计模式：23种设计模式（面向对象，已经或多或少涉及一些）**
      5. **软件工程专业知识：**软件工程强调在软件开发过程中，按照一定的 流程和方法 来组织、规划、管理和控制软件开发过程，以保证开发出高质量、可靠、易维护的软件项目
      6. **软件工程专业学习的好处**
         1. 提高开发效率：软件工程涉及了设计模式、项目框架、工具的知识，学好后可以提高开发效率，缩短软件开发周期。
         2. 提高软件质量：通过一定的流程、方法和工具来规范软件开发过程，可以有效地提高软件质量，降低软件维护成本
         3. 提高团队协作效率：软件工程介绍了一些团队协作工具和方法，可以帮助团队更好的协作，避免开发中的问题
         4. 提高软件安全性：通过学习规范的流程以及软件风险知识，可以培养我们对系统安全性、可靠性和可维护性的意识，从而有效地避免像安全漏洞和程序崩溃等问题。
   8. 后端开发通用：SQL数据库也是一种语言
   9. C++项目实践，求职备战
   10. **涉及JavaWeb的知识**
       1. ****

# **三. 6.31前**

## 编程指北的书单，正好有涉及Linux内核的书籍，

1. 计算机网络和操做系统已经开始网课了，计算机组成原理也开始了，（）
2. 数据结构和算法（坚持练习）（听那哥们说他们的算法题很难，多了解了解，假期加强对这块的学习）
3. 对电子书籍进行借阅
4. Linux（免费服务器三个月，也有了，正好学习，（b站网课，kwc,免费服务器，sgg,网课，但是涉及内核不懂，慢慢来吧，网课会涉及到linux内核相关知识））
5. 选择牛客网刷题，计算机各种题型都有，写blog
6. 编程指北和鱼皮，卡尔的路线结合学习
7. 看那份时间规划图，积累资源，为假期的学习做准备
8. github加速方法， 掌握gitee,等等
9. 放假前15天开始买票
10. 手机分身，将一些软件放到一些地方
11. 前端涉及的语言：html、css、javascript、前端易上手见效快

## 上面是进度条向前走

1. 暑期社会实践
2. 安全考试（天目湖群里看到，没有听说）

# 四.6.31~8.31（可以从github上获取先下载到本地）（了解相关路线，获取学习路线）

一个月内学习完java语法

关于视频剪辑，秀米，pre, adobe,inDesign进行排版, 等一些剪辑，排版软件进行掌握学习

## 编程指北，对应学习路线，结合起来

## kwc学习路线，结合做网页（需要前端和后端知识的）和服务器相关的

## 每个语言或技术学完，做一个对应的项目或系统，考取认证最高的证书

## 写完对应的项目，整理改造，完成大作业

## 哈佛大学CS50课程，各种小玩具都尝试的做一下

## 互联网时代，学习大学一些课程，网上资源，哈佛，MIT看课

## 英文学习，英文字母，学习英语意识

每天要读文章，读技术博客，刷题，刷项目，github, 获取经验，计划不断变化，

1. 每天数据结构和算法（细水长流，english spoken, 英语文献，英语书籍，）
2. 写blog
3. STL源码：
4. Linux
5. 设计模式（每天一两个，一共23，卡吗网有，还可以找别的刷题软件，clone到本地了）
6. 软件工程相关知识（）
7. 提前学习Java语法

## 联系前端和上面所学知识点

1. 前端需要上面设计模式的几种，也需要数据结构和算法、计算机网络、操作系统、git命令，github代码托管、Linux命令、docker容器
2. 了解前后端：后端有C++, Java
3. 了解yasi相关的知识，

# 五.大二

1. 前端相关知识（Java为主），
2. Java学习（跨平台、生态好、问题网上都可以找到）
3. 后端相关知识（更加锻炼思维能力）
4. 这太少了，更多专业的书籍，都要去看
5. Go语言（腾讯、字节注重go语言），嵌入式，人机交互

### 知识点

* 数据库（SQL也是一种数据结构）
* 开发框架
* 包管理工具
* Redis（后端、）
* 消息队列
* Nginx
* 微服务（架构）
* 容器
* 架构设计

1. python学会写一些小脚本，提高生产力，各种库、各种生态

# 六.要参加的比赛

1. csp认证
2. 蓝桥杯，leetcode（周赛先练练手）
3. 数模竞赛了解一下
4. 准备yasi

# 七.已经了解过一门语言，相应计划

 深化C/C++知识:

* 学习高级特性,如模板编程、STL库的使用
* 了解内存管理和性能优化技术
* 探索多线程编程

 拓展新的语言:

* Python: 对于数据分析、人工智能方向很有用
* Java: 企业级应用开发的主流选择
* JavaScript: 如果您对Web开发感兴趣

 专注领域:

* 系统编程: 深入学习操作系统原理,可以考虑参与开源项目
* 嵌入式系统: 结合您的C/C++基础,学习微控制器编程
* 游戏开发: C++在游戏引擎开发中广泛使用
* 网络编程: 学习socket编程,网络协议等

 计算机科学基础:

* 加强数据结构和算法学习
* 深入理解计算机体系结构
* 学习编译原理

 软件工程实践:

* 学习版本控制(如Git)
* 了解持续集成/持续部署(CI/CD)
* 掌握测试方法论

 数据库知识:

* 学习SQL,了解关系型数据库
* 探索NoSQL数据库

 云计算和容器技术:

* 了解云服务平台(如AWS,Azure)
* 学习Docker和Kubernetes

 安全知识:

* 了解常见的安全漏洞和防护措施
* 学习密码学基础

 软技能:

* 提高英语阅读和写作能力
* 培养团队协作和沟通技巧
* 学习项目管理基础

 持续学习:

* 关注技术博客和论坛
* 参与开源项目
* 尝试写技术博客,分享您的学习经验

# 八.前后端学习路线

前端学习路线：

1. HTML5 和 CSS3
   * 学习语义化HTML
   * 掌握CSS布局技术（Flexbox, Grid）
   * 响应式设计
2. JavaScript
   * ES6+新特性
   * DOM操作
   * 异步编程（Promise, async/await）
3. 前端框架
   * 选择一个主流框架深入学习（如React, Vue, Angular）
   * 状态管理（如Redux for React, Vuex for Vue）
4. 构建工具和包管理器
   * npm 或 yarn
   * Webpack, Vite 等
5. CSS预处理器
   * Sass 或 Less
6. 版本控制
   * Git

后端了解：

1. 后端语言入门
   * 可以选择Node.js（因为您已经学习了JavaScript）
   * 或者Python（易学且应用广泛）
2. 数据库基础
   * 关系型数据库（如MySQL）
   * NoSQL数据库（如MongoDB）
3. API开发
   * RESTful API设计原则
   * GraphQL基础
4. 服务器知识
   * 基本的Linux命令
   * 了解Web服务器（如Nginx）
5. 网络基础
   * HTTP/HTTPS协议
   * 基本的网络安全知识

学习建议：

1. 项目驱动学习：边学边做小项目，如个人博客、待办事项应用等。
2. 使用在线资源：Coursera, freeCodeCamp, MDN Web Docs等。
3. 参与开源项目：这能帮助您了解实际开发流程。
4. 建立个人作品集：展示您的项目和技能。
5. 关注技术社区：如GitHub, Stack Overflow等。
6. 练习代码调试：这是一项重要的技能。

# 九.学习C++如何进阶

1. 深入理解C++核心特性：
   * 深入学习模板元编程
   * 掌握多线程编程（C++11引入的thread库）
   * 理解内存管理和智能指针
   * 学习lambda表达式和函数对象
2. 现代C++特性：
   * 深入学习C++11/14/17/20的新特性
   * 了解即将到来的C++23标准
3. 标准模板库（STL）：
   * 深入理解STL的设计哲学
   * 熟练使用各种容器和算法
4. 设计模式：
   * 学习GoF的23种设计模式在C++中的应用
5. 高级主题：
   * 学习RAII（资源获取即初始化）技术
   * 理解编译时多态和运行时多态
   * 探索元编程和模板特化
6. 性能优化：
   * 学习编写高性能C++代码
   * 理解CPU缓存和内存模型
   * 学习使用性能分析工具
7. 跨平台开发：
   * 了解不同编译器的差异
   * 学习跨平台库如Boost
8. 项目构建和管理：
   * 学习使用CMake等构建系统
   * 掌握版本控制（如Git）的高级用法
9. 测试和调试：
   * 学习单元测试框架（如Google Test）
   * 掌握高级调试技巧
10. 应用领域：
    * 游戏开发：学习游戏引擎（如Unreal Engine）
    * 系统编程：深入学习操作系统原理
    * 嵌入式系统：学习实时操作系统和硬件编程

推荐书籍：

* 《Effective C++》和《More Effective C++》 - Scott Meyers著
* 《C++ Templates: The Complete Guide》 - David Vandevoorde等著
* 《C++ Concurrency in Action》 - Anthony Williams著
* 《Modern C++ Design》 - Andrei Alexandrescu著

实践建议：

1. 参与开源项目，阅读高质量C++代码
2. 尝试重写一些常用的数据结构或算法
3. 开发一个小型项目，如简单的游戏引擎或数据库系统