第 1 章

赛前准备

1

欢迎踏入算法竞赛的精彩世界。在本书的第1章中，我们将为你揭开赛前准备的神秘面纱。从ACM-ICPC到蓝桥杯、天梯赛，我们将逐一介绍这些顶级赛事的赛制和特点，让你对算法竞赛有一个全面而深入的了解。同时，我们还将探讨竞赛语言的选择、编程环境的搭建以及训练平台的使用，助你在竞赛中游刃有余。最后，我们将提供能力要求和学习建议，引导你在算法竞赛的道路上稳步前行。让我们一起开启这段充满挑战与收获的旅程吧！

1.1 算法竞赛简介

算法竞赛是一种竞技活动，旨在通过解决一系列算法问题来展示和比较参赛者的编程能力。在算法竞赛中，参赛者需根据题目的要求设计和实现高效算法，并在限定时间内提交并运行程序。这些算法问题通常涉及计算机科学和算法设计的多个领域，如图论、动态规划、搜索、字符串处理、数论等。参赛者需灵活运用算法思维、编程技巧和创造力来解决这些问题。比赛通常设置时间限制，参赛者需在规定的时间内尽可能多地解决问题，设计出正确且高效的算法，并得到正确的输出。

提交的程序会经过自动评测系统的测试，以验证程序的正确性和效率。根据正确解决题目的数量和用时，参赛者被排名并获得相应奖励，激励参赛者不断提升算法设计和编程能力，寻求更优的解决方案。算法竞赛不仅是一项技术竞赛，还是学习和交流的平台。参赛者通过与其他优秀选手的竞争，学习他人的解题思路，分享经验和技巧，不断提升自身能力。

算法竞赛不仅考察参赛者的算法和编程能力，还培养他们的问题解决能力、团队合作意识和压力管理能力。算法和编程技能是基础，持续的训练和技术提升，以及对算法和数据结构的深入理解与应用至关重要。

此外，团队合作的重要性不可忽视。很多算法比赛是团队比赛，高效沟通、合理分工对成功解决问题至关重要。最后，良好的心态也非常关键。比赛中难免会遇到困难和挑战，保持冷静，坚持不懈，并从失败中汲取教训，是获得成功的重要因素。

以下是程序设计竞赛的常见赛制。

1. ACM-ICPC赛制

- 分数计算：只有问题完全解决时才能计入AC（Accepted）题数。分数还受提交时间和错误次数的影响，错误提交会增加“罚时”。AC题数越多且罚时越少的选手排名越高。
- 反馈方式：提供实时反馈，选手可在短时间内看到提交结果。
- 排名方式：按AC题数降序排列，若选手得分相同，则排名按照罚时升序排列。
- 优点：竞技感强，刺激，实时反馈增加紧张感。
- 缺点：无法通过部分得分或“骗分”提升成绩。

2. OI赛制

- 分数计算：每道题根据得分点计分（对了多少个点得多少分），选手只能提交一次（离线），每道题的最终得分为最后一次提交的得分。
- 反馈方式：无实时反馈。
- 排名方式：无罚时，总得分多者排名更高。
- 优点：有期待感（等待评测结果），可通过部分得分积累总分（有骗分机会）。
- 缺点：无实时反馈，需等待评测结果。

3. IOI赛制

- 分数计算：每道题根据得分点计分（对了多少个点得多少分），选手可提交多次，每道题的最终得分为最后一次提交的得分。
- 反馈方式：提供实时反馈，提交后可在短时间内看到结果。
- 排名方式：无罚时，总得分多者排名更高。
- 优点：提供实时反馈，有机会通过部分得分累积总分（有骗分机会），对选手更友好。
- 缺点：相较于ACM-ICPC，比赛的竞技感稍弱，可能不那么刺激。

1.1.1　ACM-ICPC 简介

ACM-ICPC（Association for Computing Machinery International Collegiate Programming Contest，国际大学生程序设计竞赛）是世界上最具声望、最具挑战性的大学生程序设计比赛之一，被誉为计算机领域的顶级赛事之一。

最初，ICPC由ACM（Association for Computing Machinery，国际计算机学会）赞助，尽管ACM现在已不再担任赞助方，比赛仍被习惯性地称为ACM-ICPC。

1. 竞赛形式与目标

ACM-ICPC是一项多轮的团队竞赛，旨在考察参赛者的算法设计和编程能力。每支参赛队伍由三名大学生组成，他们需在限定时间内解决一系列编程问题。

2. 题目类型

比赛题目涵盖计算机科学和算法设计的各个领域，包括图论、动态规划、搜索、字符串处理和数论等。每道题目要求参赛者设计并实现一个正确且高效的算法来解决问题。

3. 比赛规则

比赛通常持续数个小时（通常为 5 个小时），每支参赛队伍只能使用一台计算机，并且仅限使用官方指定的编程语言（例如 C/C++、Python）。参赛队伍需要根据题目要求编写程序，提交后由评测系统运行，生成正确的输出结果。

4. 评判与排名

提交的程序将由自动评测系统进行测试，以验证其是否能够正确、有效地解决问题，并在限定的时间内运行完成（即符合时间复杂度的要求），有些题目对内存的使用量有所限制，即符合空间复杂度的要求）。参赛队伍的排名根据解决问题的数量和用时长短，即解决问题越多且用时越短的参赛队伍排名越高。

5. 挑战与竞争

ACM-ICPC为参赛者提供了一个充满挑战和竞争的舞台，激发他们的创造力和解决问题的能力。参赛者需在紧张的环境中思考问题、分析算法、调试程序，并在限制时间内提交解决方案。

ACM-ICPC是分级别进行的比赛，分为区域赛、区域决赛和世界总决赛。区域赛的优秀队伍将获得参加区域决赛的资格，每个区域决赛的优秀队伍将获得参加国际总决赛的资格。国际总决赛汇聚了来自世界各地的顶尖参赛队伍，代表各个国家和地区进行最终角逐。

6. 赛季赛程

ACM-ICPC赛事及时间如表1-1所示。

表 1-1　ACM-ICPC 赛事及时间

赛　　事	时　　间
ICPC/CCPC 网络赛	8 月底至 9 月初
ICPC/CCPC 区域赛	9 月底至 11 月底
ICPC EC Final/CCPC Final	12 月中下旬
ICPC 世界总决赛	次年 4 月至 6 月

当然，具体的日期可能会根据每年的赛事安排有所不同。

7. 参赛建议

如果你决定开启自己的ACM-ICPC之旅，那么就义无反顾地走下去吧！以下是一些参赛建议。

- 基础知识打牢：这是任何事情的基础。首先，你需要确保编程基础知识扎实，包括对编程

语言、算法理论、数据结构的深入理解，以及对数学知识的掌握。

- 大量练习：ICPC是一个实践性很强的竞赛。你需要对各种问题进行大量实践，以掌握解决这些问题的策略和方法。参加在线竞赛，解决各类编程问题，尝试更复杂的题目，都是提升自己能力的有效方法。
- 学习新技术和算法：编程领域不断发展，总有新技术和算法出现。因此，保持学习新知识的热情和兴趣很重要。新的、之前未接触过的算法，有时可能是解决问题的关键。
- 团队协作：ICPC是一项团队竞赛，良好的团队协作至关重要。你需要学习如何与队友有效地沟通，分享思考和解决问题的方法，并合理分配任务。
- 比赛策略：ICPC的比赛规则和格式特殊，采用一定的比赛策略会非常有帮助。例如，你需要懂得如何在比赛开始时快速浏览并评估所有题目的难度，如何优先选择要解决的问题，以及如何有效利用提交次数和罚时等。
- 保持冷静和理智：在比赛压力下，保持冷静和理智非常重要。不要因一时的失败或困难而沮丧，保持冷静，理智分析问题，寻找解决方法，以积极的态度面对赛场。
- 积累比赛经验：多参加比赛可以帮助你适应比赛的压力和环境，提高你解决问题和团队协作的能力。

1.1.2　CCPC 简介

中国大学生程序设计竞赛（China Collegiate Programming Contest，CCPC）是中国高校中最具影响力的大学生程序设计竞赛之一，常年得到华为、腾讯等大厂的赞助。你可以将其理解为中国版的ICPC，但它们是两个不同的比赛。如果你能在ICPC或CCPC的任何比赛中取得奖项，都将受到大厂HR的关注。

CCPC的参赛队伍由三名大学生组成，每队需要在规定时间内解决一系列编程问题。比赛采用多轮赛制，每轮比赛都会有一组编程问题，参赛队伍通过编写程序解决这些问题，并提交给评测系统进行自动评测。注重参赛队伍的算法设计、编程实现和问题解决能力。队伍需要根据问题的要求，设计出高效、正确的算法，并在限定时间内完成编程实现。

CCPC是一项激烈的竞赛，参赛队伍需要在有限的时间内迅速分析问题、设计算法，并通过合理的编程实现获得正确的解答。比赛排名根据参赛队伍解决问题的数量和总用时来确定。通过参加CCPC，大学生能够锻炼自己的算法设计和编程能力，提高解决问题的能力和团队协作精神。

1.1.3　NOIP/NOI/ CSP-J/S 简介

1. 信息学奥林匹克联赛NOIP

全国青少年信息学奥林匹克联赛（National Olympiad in Informatics in Provinces，NOIP）自 1995 年起至 2022 年已举办 28 次，每年由中国计算机学会统一组织。初、高中或其他中等专业学校的学生可报名参加该联赛。NOIP 在同一时间、不同地点以各省市为单位，由特派员组织，采用全国统一大纲和试卷。联赛分初赛和复赛两个阶段。初赛主要考察通用和实用的计算机科学知识，以笔试

形式进行；复赛则为程序设计，要求在计算机上调试完成。参加初赛的选手必须达到一定分数线，才有资格参加复赛。联赛设有入门组（Junior）和提高组（Senior）两个组别，难度不同，分别面向初中和高中阶段的学生。

2. 全国青少年信息学奥林匹克竞赛

全国青少年信息学奥林匹克竞赛（National Olympiad in Informatics，NOI）由中国计算机学会（China Computer Federation，CCF）主办，旨在向中学生普及计算机科学知识，培养学生的算法设计和编程能力。

信息学奥赛与数学、物理、化学、生物奥赛并称为奥林匹克五大联（竞）赛，是高校自主招生中含金量最高的赛事之一，被视为进入名校的重要途径。信息学奥赛通常分为入门组和提高组两个级别，分别适合不同编程基础和学习阶段的学生。

3. CSP-J/S

CSP-J/S（Certified Software Professional - Junior/Senior，非专业级别的能力认证）是一项全国性的计算机算法和编程能力认证，分为入门组和提高组两个级别。该认证自2019年起每年在全国范围内举行，遵循统一大纲，旨在普及编程和计算机科学知识。在CSP-J/S中表现优异的参赛者将有机会优先参加后续的全国青少年信息学奥林匹克竞赛系列活动。

1）级别

（1）入门组

入门组是较低级别的比赛，主要面向小学生和初中生，题目难度相对较低。考核内容包括计算机基础知识、编程基础、数据结构入门及常见算法，如枚举、贪心算法、递归、动态规划、深度优先搜索（Depth-First Search，DFS）和广度优先搜索（Breadth-First Search，BFS）等。尽管入门组题目有一定难度，但相较于提高组仍显简单。

（2）提高组

提高组的难度更高，考核内容覆盖复杂的数据结构、高中及部分大学阶段的数学知识以及更具挑战性的算法。题目难度甚至可能超过本科计算机相关专业的课程水平。

2）与NOIP的关系

NOIP（National Olympiad in Informatics in Provinces，全国青少年信息学奥林匹克竞赛）是一项在秋季学期举办的全国性信息学竞赛。由中国计算机学会统一命题，并由各省特派员组织考试。在2019年前，NOIP分为入门组和提高组两个组别，改革后主要面向高中阶段的高水平选手。在NOIP中表现优异的选手将有机会参加省队选拔，进而获得参加全国青少年信息学奥林匹克竞赛的资格。

3）竞赛时间安排

CSP-J/S的竞赛时间安排见表1-2。

表 1-2　CSP-J/S 的竞赛时间安排

时　间	赛事考级名称	主　办　方
9 月	CCF 计算机软件能力认证（CSP-J）初赛	中国计算机学会
9 月	CCF 计算机软件能力认证（CSP-S）初赛	中国计算机学会
10 月	CCF 计算机软件能力认证（CSP-J）复赛	中国计算机学会
10 月	CCF 计算机软件能力认证（CSP-S）复赛	中国计算机学会

4. NOI冬令营

NOI冬令营（全国青少年信息学奥林匹克竞赛冬令营）是由中国计算机学会在寒假期间组织的为期一周的培训活动。NOI冬令营包括授课、讲座、讨论和测试等内容，旨在为选手提供深入的算法和编程技能培训，帮助他们在后续比赛中提升竞争力。

5. 国际信息学奥林匹克竞赛

国际信息学奥林匹克竞赛（International Olympiad in Informatics，IOI）是全球最具影响力的信息学竞赛之一，参赛选手为全世界各国通过本国计算机竞赛选拔出来的中学生。每个国家最多可派出4名选手参赛，这些选手通常在国家级信息学竞赛中表现优异，经过严格选拔后脱颖而出。

6. 中国国家队选拔

国际信息学奥林匹克竞赛中国国家队选拔赛（China Team Selection Competition，CTS）是面向中国国家队集训队员的选拔活动，通常与NOI冬令营同期举办。选拔内容包括作业测试、集训环节、冬令营交流、线下选拔测试及现场答辩等，最终选出4名选手组成中国国家队，代表中国参加IOI。

7. 省选

省选是在NOI冬令营之前由各省组织的省级代表队选拔赛，用以确定参加NOI冬令营的选手名单。自2022年起，省选采用全国统一命题和统一时间，在各省同步举行。每个省份基本名额为4名男生和1名女生。此外，根据实际情况可能会增加激励名额或重大贡献奖励名额。

8. 晋级路线

从CSP至IOI的晋级路线大致如图1-1所示。

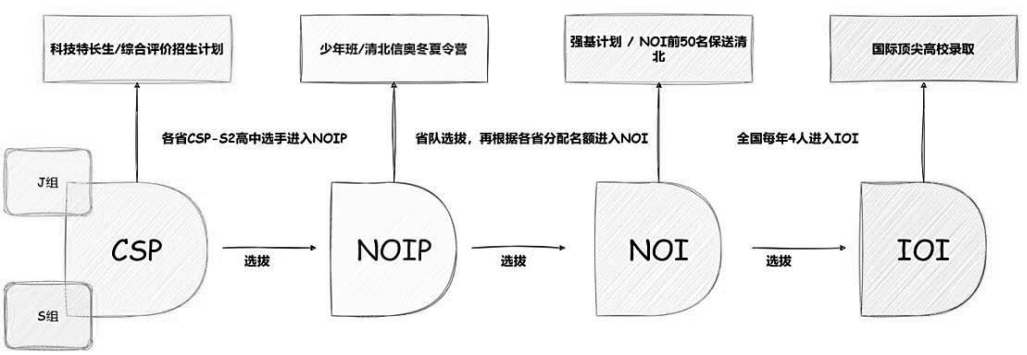

图 1-1　从 CSP 至 IOI 的晋级路线图

1.1.4　蓝桥杯简介

蓝桥杯全国软件和信息技术专业人才大赛是中国规模最大、影响力最广的IT技术类赛事之一。赛事分为个人赛和团队赛两个阶段，内容涵盖计算机程序设计、软件开发和算法设计等领域。

蓝桥杯由教育部、工业和信息化部等多部门联合主办，旨在选拔和培养优秀的软件和信息技术人才。比赛包括在线笔试与实践环节，参赛者需解决一系列编程和算法问题。题目范围广泛，涵盖计算机科学、软件工程及网络技术等多个方向，注重对参赛者综合能力和创新思维的考察。

参赛者需熟练掌握编程语言、算法和数据结构，具备分析和解决实际问题的能力。比赛不仅为参赛者提供展示才华和锻炼技能的平台，还为他们创作了与其他优秀人才交流和学习的机会。成绩优异的参赛者还有机会参加国际级竞赛与相关项目。

作为中国软件和信息技术领域最具影响力的大赛之一，蓝桥杯在培养和选拔优秀人才方面发挥了重要作用。

1.1.5　天梯赛简介

天梯赛是一项面向程序员的高水平竞技赛事，由全国高等学校计算机教育研究会主办，联合主办单位还包括教育部高等学校计算机类专业教学指导委员会、教育部高等学校软件工程专业教学指导委员会和教育部高等学校大学计算机课程教学指导委员会。

天梯赛的参赛者需在规定的时间内解决一系列编程问题并提交解答。问题内容涵盖算法设计、数据结构和编程技巧等方面，要求参赛者能够快速思考并实现高效的解决方案。天梯赛的评判系统会自动测试参赛者提交的程序，并根据正确性和效率评估成绩和排名。

天梯赛分为三个组别：珠峰争鼎（本科组）、华山论剑（本科组）和沧海竞舟（专科组）。

参赛规则包括：

- 本科生可报名参加“珠峰争鼎”或“华山论剑”组，专科生可自由选择任一组别。
- 每支参赛队最多由10名队员组成，队员必须为所属高校的在册本科生或专科生。若队伍中有本科生，则不得报名“沧海竞舟”组。
- 每所高校参赛队伍数量不设上限，但只有成绩最好的3支队伍参与计分和评奖。如果同一所高校的参赛队伍报名参加不同组别，则按照最高组别确定高校的排名，只计算该组别最佳3支参赛队伍的得分。
- 曾获得两届全国个人冠军奖的队员不得再次参赛。

其他规定：

- 每队必须配备至少一名教练，教练必须是参赛高等学校的正式教师，负责指导和联络工作，并确保队员符合规程要求。
- 参赛队伍必须提供由学校或学院教务部门出具的队员身份证明。
- 欢迎中学生友情参赛，但仅能参与个人奖评选，且必须经竞赛专家委员会批准后方可报名（可参加任一组别）。

1.2　语言和工具

在算法竞赛中，选择合适的竞赛语言、搭建高效的编程环境，以及利用优质的训练平台是取得优异成绩的关键。本节将引导读者了解这些核心要素，帮助读者在竞赛中发挥最佳水平。

1.2.1　竞赛语言

高水平赛事中常用的编程语言包括C++、Java和Python，它们各具特色，适用于不同场景。C++以高执行速度和丰富的标准库著称，适合处理大规模数据和复杂算法，同时提供对指针和内存管理的细粒度控制。Java具有简洁的语法、强大的面向对象特性和丰富的标准库，适用于大型项目和图形界面开发，但执行速度稍慢。Python则凭借其简洁高级的语法和广泛的第三方库备受欢迎，学习门槛低，但执行效率较低，不适合高性能要求的竞赛题目。

常用的编译工具包括支持多语言的VS Code、快速轻便且功能强大的Sublime Text，以及适用于Java的IntelliJ IDEA和适用于C++的CLion。虽然赛事允许使用多种编程语言，C++因其高效性和强大的标准库（如STL）而被广泛推荐。然而，最终的选择取决于个人偏好、熟悉程度和竞赛要求。建议选择自己最擅长的语言和熟悉的开发环境（包括编译工具），以便专注于解题和算法实现。

1.2.2　编程环境

算法竞赛大多使用C++，因为C++具有强大的标准库和高效的内存管理能力，提供丰富的数据结构和算法工具。C++的高执行速度使其能够应对大规模数据和复杂计算，可以满足算法竞赛的高性能要求。此外，由于C++的灵活性和广泛使用，使得大量算法竞赛资源和题解多以C++编写，方便选手学习和参考。

对于尚未接触过C++的选手，可以优先使用自己熟悉的语言（如Python），并在后续学习算法的过程中逐步掌握C++。

1.2.3　训练平台

Codeforces是全球广受欢迎的在线编程竞赛平台，题目涵盖算法、数据结构和动态规划等领域，定期举办比赛以提升选手的竞赛技能。

AtCoder是日本知名的在线竞赛平台，提供各种难度的编程题目，通过定期比赛让选手与全球选手竞争，以便在竞赛中不断提高选手的水平。

LeetCode是一个全面的刷题平台，为用户提供丰富的面试题和算法题，配有讨论区和解题，方便用户交流学习。

HDU、POJ和ZOJ等学校的OJ平台也有大量经典题目，适合练习C++语法和基础算法。本书有配套的在线判题系统（Online Judge，OJ），名为星码StarryCoding，特别适合初学者使用。

此外，许多高校会组织校内编程竞赛和训练活动。读者可以参加校内或周边学校的竞赛，与

其他同学切磋技艺，并向优秀选手学习，能够快速积累经验和成长。需要注意的是，在选择训练平台时，应根据自身水平和目标，挑选适合的题目类型和难度，循序渐进，从基础逐步挑战更高难度，以不断提升解题能力和编程技巧。

准备参加竞赛或学习编程语言时，读者可能会遇到一些常见的术语，如表1-3所示。

表 1-3　常见的术语

缩　　写	英文全称	中文解释
cf	Codeforces	全球最大算法平台
atc	AtCoder	日本算法平台
WF	World Final	全球总决赛
AC	Accepted	答案正确/通过
WA	Wrong Answer	答案错误
PE	Presentation Error	格式错误
RE	Runtime Error	运行时错误
CE	Compilation Error	编译错误
MLE	Memory Limit Exceed	超出内存限制
TLE	Time Limit Exceed	超出时间限制/运行超时
OLE	Output Limit Exceed	输出超出限制

1.3　能力要求和学习建议

如果你想参加算法竞赛，这里有一些建议，可能对你有所帮助。

1.3.1　如何迈出算法竞赛第一步

加入ACM校队的第一步是打下坚实的基础并不断练习。首先，学习并掌握计算机科学和算法的基础知识，例如数组、链表、树、排序算法以及图论等。刷题是提高技能的重要方法之一，可以通过在线平台（如Codeforces、AtCoder、LeetCode和牛客等）解决各种难度的编程问题。

参加校内竞赛展示自己的实力，争取取得好成绩，以吸引校队的关注。积极参与ACM的训练和选拔活动，在这个过程中评估自己的水平，同时向教练和学长请教以提升解题能力。通过ACM校队选拔赛展示你的解题能力和团队协作精神。

加入ACM社群，例如学校的ACM俱乐部或训练队，与其他选手交流、分享经验，提升整体水平。与此同时，关注最新的算法和竞赛动态，不断学习新的知识与技巧。通过参加其他竞赛、项目或研究，拓展自己的视野，逐步提升实力，最终成功进入ACM校队。

只要你对算法充满热情，现在就开始行动吧！

1.3.2　如何合理且高效地训练

首先，从基础题目开始，建立扎实的计算机科学和算法基础。通过刷一些基础和简单题目，总结通用模板和解题思路，以便应对类似或改编题目。解决问题后，务必进行详细分析与复盘，回顾解题过程，探索其他可能的解法，分析时间和空间复杂度，并学习更加优雅、高效的解法。

组建或加入竞赛团队，与队友一起训练和刷题，通过分享思路和经验促进学习，提升团队合作与沟通能力。定期参加本地、区域或在线编程竞赛，积累实战经验，锻炼在竞赛环境下的应变能力、时间管理和调试代码的技巧。

此外，阅读优秀选手的解题报告、博客和教程，学习他们的思维过程和解题策略，从中汲取灵感和指导。由于编程竞赛题目和技术不断发展，持续更新自己的算法和数据结构知识，掌握新的解决方法尤为重要。设定阶段性目标，挑战更高难度的题目，追求更好的竞赛排名，持续突破自我，挑战极限，提高技能水平。

1.3.3　补题和总结的重要性

补题和总结是编程竞赛中不可或缺的环节。竞赛本身是在高压环境下进行的，对心态、经验和技巧的提升具有重要作用，但其中的知识盲点和漏洞往往需要通过赛后补题来完善。

在比赛中遇到未能解决的问题，赛后可以专门花时间学习和研究这些问题，寻找解决方案。这不仅能帮助你掌握新知识和技能，还能帮助你加深对复杂概念的理解。每一个问题都是新的挑战，通过补题，你能提升解决问题的能力，并强化思考和分析问题的能力。

同时，总结同样重要。总结是反思与改进的过程，通过回顾比赛表现，发现自身优点和不足。总结可以帮助你理清思路，加深对问题和解法的理解。你可以通过记录自己的思考过程和解决步骤，清晰地了解解题策略与方法。这样的记录还可作为未来的思考基础，为下一场比赛提供宝贵的经验和指导。

补题与总结是从竞赛中学习和提升的关键环节。通过不断地补题和总结，你不仅能从错误中吸取教训，还能显著提高自己的能力，为未来的竞赛做好准备。

1.3.4　如何正确看待算法竞赛的付出和收益

参加算法竞赛需要投入大量的时间和精力。为了在比赛中取得好成绩，你需要学习新的算法和数据结构，解决训练题目中的问题，参加模拟比赛等。这些准备工作可能会占用你的休息时间，并影响其他活动。然而，这些付出是提升编程能力和解题技巧的必要过程。通过比赛，你会接触到课堂上难以深入学习的许多高级算法和数据结构（其中一些在实际工作中可能用得不多），以显著提升自己编写高质量代码和快速解决问题的能力。

尽管在短期内可能会感到疲劳或压力，但这些努力会带来坚实的技能和宝贵的经验积累，对个人成长和职业发展都大有裨益。这些能力不仅在比赛中有用，在实际的软件开发中也能显著提高工作效率。此外，优秀的比赛成绩可能为你赢得奖项，并成为简历上的亮点，但更重要的是这些成

绩背后的能力和经验。因此，不应过度关注奖项本身，而应重视学习与自我提升的过程。

团队比赛还能够培养团队协作与沟通能力，这是任何职业都极为重要的技能。无论未来是否从事编程相关工作，这些能力都会让你受益匪浅。

总的来说，参加算法竞赛虽然需要付出许多努力，但其收益也是显而易见的。不过，每个人的目标和情况都不同，你需要根据自身情况决定是否参赛，以及如何平衡投入的时间和精力。更重要的是，无论是否选择参与，都要明白比赛只是提升和展示自己能力的一种途径，结果固然重要，但在这个过程中的学习和收获更值得重视。

最后，提醒各位同学，算法竞赛并非一条轻松且高性价比的道路。它要求你承受孤独、接受失败，同时也能带来追逐荣誉的满足感。如果你选择这条道路，请做好充分准备，享受其中的挑战与成长。

第 2 章

基础语法

2

掌握一门编程语言是参加竞赛的基础。本章以C++语言为例，系统介绍其基础语法和编程要领。通过学习本章的内容，相信你能够快速上手C++编程，并用这门语言高效地解决实际问题。

2.1 第一个程序：Hello World

为了编写代码，我们需要安装一个IDE（Integrated Development Environment，集成开发环境）。适合新手的常用IDE包括Dev-C++和Code::Blocks等，这些软件的安装过程相对简单，建议读者自行查阅相关资料完成安装。

不建议新手使用Visual Studio或VS Code，因为它们的配置相对复杂。当然，这也归因于它们的功能过于强大，新手暂时用不到如此多的功能。

在计算机编程中，“Hello, World!”程序通常用于教授编程语言的基础语法，或用于测试编程语言、编译器或开发环境。本节将介绍如何用C++编写一个“Hello World!”程序，开始你的编程学习之旅。

2.1.1 程序示例

以下是一个C++的“Hello, World!”程序的示例：

```
#include <iostream>

int main() {
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

这段程序使用C++的标准输入/输出库（iostream）中的cout对象和endl操作符，将字符串“Hello, World!”输出到控制台。程序的最后返回状态码0，表示程序正常结束。

下面解释一下上面这段文字中的一些概念：

- 标准输入/输出库：C++官网提供的库，用于处理输入/输出操作（通常是在控制台上）。开发者可以调用其中的函数或对象来实现数据的输入/输出，例如从键盘读取数据或将内容输出到屏幕。
- 库：类似于一个工具箱，提供了各种各样的工具和功能，开发者通过包含或引入（#include）即可直接使用，避免从零开始编写复杂的功能。
- 字符串：由一系列字符组成的文本序列，用于表示文字内容。在C++中，字符串常以双引号包裹，例如"Hello"或"Welcome"，也就是以双引号作为起止符号。

如果暂时看不懂这些概念，也不用担心，可以先尝试将代码复制到开发环境中，然后运行程序看看效果。在学习编程语法时，尤其需要注意符号的正确性，例如冒号、括号等，不要写错或漏写。

2.1.2　头文件

在C++中，头文件是包含程序所需的函数声明、变量声明和类型定义的文件。

例如，`#include <iostream>`这一行代码引入了iostream头文件。

iostream库提供了基本的输入/输出功能，例如我们在上面的“Hello, World!”程序中用到的std::cout和std::endl。

在算法竞赛中，<bits/stdc++.h>这个“万能头文件”被广泛使用，它几乎涵盖所有常用的头文件，包括输入/输出库、各种标准模板库（Standard Template Library，STL）、数学库等。然而，需要注意的是，有些较旧的在线判题系统可能并不支持这个万能头文件。

开发者只需要引入头文件，就可以使用库中的所有内容，包括宏、对象、函数、方法等，从而大幅度提高开发效率。

2.1.3　命名空间

在C++中，命名空间是一种用于防止命名冲突的机制。

命名空间可以包含变量、函数、类型等。例如，在上面的“Hello, World!”程序中，std::cout和std::endl实际上位于std命名空间中。写成std::cout表示我们要使用的是std命名空间中的cout对象。

当然，我们也可以在程序开头加上`using namespace std;`这行代码，这样在后续代码中就可以直接使用cout和endl，而不需要加上std::前缀。

然而，在大型或复杂的程序中，为了防止命名冲突，通常推荐使用完整的std::cout和std::endl。

下面两种写法都是正确的。

使用using namespace std;的代码：

```
#include <iostream>       // 引入输入/输出流库
using namespace std;      // 使用标准命名空间，可以直接使用cout和endl而无须加上std::前缀
```

```
int main()       // 主函数，程序的入口点
{
    cout << "Hello, World!" << endl;   // 输出“Hello, World!”并换行
    return 0;     // 返回0表示程序正常结束
}
```

不使用using namespace std;的代码：

```
#include <iostream>  // 引入输入/输出流库
int main()           // 主函数，程序的入口点
{
    std::cout << "Hello, World!" << std::endl; // 使用std::前缀来访问标准命名空间中的cout
和endl
    return 0;         // 返回0表示程序正常结束
}
```

2.1.4　main 函数

在C++程序中，main函数扮演着至关重要的角色。它是整个程序的入口点，也是操作系统（包括几乎所有在线判题系统OJ）默认的启动程序的起点。

每个程序只允许有一个main函数。

在前面的“Hello, World!”程序中，`int main()`这行代码定义了一个返回值为int类型的main函数，该函数不接受任何参数（关于函数和参数的详细介绍将在后续章节展开）。

函数体由一对花括号（{}）包围，在花括号内的所有代码都属于这个函数。

在main函数中，`return 0;`这行代码表示该函数的返回值为0。

在操作系统中，程序的返回值0通常表示该程序正常结束。如果发现程序运行完毕后返回值不是0，则说明程序在执行过程中发生了某种异常，也就是出现运行时错误（Runtime Error，RE），这时需要检查程序中是否有隐藏的错误，一般是数组越界或非法、无限递归等。

2.2　输入与输出

对于任何一个程序来说，输入/输出是与用户交互的核心功能。用户通过输入给程序提供数据，程序再通过输出向用户显示处理结果。

2.2.1　scanf 和 printf

scanf和printf是C语言中的输入/输出函数，不过在C++中也可以使用，并且具有较高的执行速度。

- scanf：用于从标准输入（通常是键盘）读取数据。例如，`scanf("%d", &num);`用于读取

用户输入的一个整数，然后把这个整数的值存储到变量num中。在这个函数中，"%d"是格式符，表示要读取的数据是整数类型。注意，在num前面要加上“&”表示取出num这个变量的地址，因为C语言是面向过程的编程语言，并且没有引用类型，所以想要在函数内改变一个变量的值，就得把变量的地址传入函数。

- printf：用于向标准输出（通常是屏幕）写入数据。例如，`printf("%d", num);`就是把变量num的值显示在屏幕上。同样，"%d"表示我们要输出整数类型的数据。

2.2.2　cin 和 cout

C++提供了更高层次的输入/输出方式，分别是cin和cout，它们是基于标准输入/输出对象的。

- cin：用于从标准输入读取数据。例如，`cin >> num;`用于读取用户输入的一个整数（假设num是一个整数变量），然后把这个整数的值存储到变量num中。
- cout：用于向标准输出写入数据。例如，`cout << num;`用于把变量num的值显示在屏幕上。

与scanf和printf相比，cin和cout的优势在于，它们能够自动识别变量类型，无须指定格式符，这使得代码更加简洁易读。例如以下代码，虽然输入/输出的数据类型不同，但是我们无须修改输入/输出的代码，无论num和str两个变量是什么数据类型，cin和cout两行代码都不针对不同数据类型进行修改。

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    int num;              // 定义一个整型变量num
    string str;           // 定义一个字符串变量str
    cin >> num >> str;    // 从标准输入读取一个整数和一个字符串，分别赋值给num和str
    cout << num << str;   // 将num和str输出到标准输出
    return 0;             // 返回0表示程序正常结束
}
```

在这个示例程序中，首先引入（即包含）了头文件，然后使用std命名空间避免频繁加前缀。接下来，定义了一个整型变量num和一个字符串变量str。通过使用cin对象，我们可以从标准输入读取一个整数和一个字符串，并将它们分别存储在变量num和str中。最后，我们使用cout对象将num和str的值输出到标准输出。

与scanf和printf相比，cin和cout的速度稍慢，在处理大量输入/输出的情况下，通常可以通过关闭同步流来提高读写速度，这也是算法竞赛中比较好的实践。这可以通过以下代码实现：

```
ios::sync_with_stdio(0);
cin.tie(0);
cout.tie(0);
```

2.2.3　各种输入/输出示例

1. 输入/输出整数

C：

```
    scanf("%d", &num); printf("%d", num); // 使用C语言的scanf函数读取整数，然后使用printf函数输出整数
```

C++：

```
    cin >> num; cout << num; // 使用C++的cin对象读取整数，然后使用cout对象输出整数
```

2. 输入/输出浮点数

C：

```
    scanf("%f", &x); printf("%f", x); // 使用C语言的scanf函数读取浮点数，然后使用printf函数输出浮点数
```

C++：

```
    cin >> x; cout << x; // 使用C++的cin对象读取浮点数，然后使用cout对象输出浮点数
```

3. 输入/输出字符串

C：

```
    scanf("%s", str); printf("%s", str); // 使用C语言的scanf函数读取字符串，然后使用printf函数输出字符串
```

C++：

```
    cin >> str; cout << str; // 使用C++的cin对象读取字符串，然后使用cout对象输出字符串
```

4. 输入/输出数组

循环的相关知识将在后续章节讲解，此处仅初步了解即可。

C：

```
    for(i = 0; i < n; i++) {
        scanf("%d", &arr[i]);      // 使用循环逐个读取数组元素
    }
    for(i = 0; i < n; i++) {
        printf("%d ", arr[i]);     // 使用循环逐个输出数组元素
    }
```

C++：

```
    for(int i = 0; i < n; ++i) {
        cin >> a[i];               // 使用循环逐个读取数组元素
    }
```

```
for(int i = 0; i < n; ++i) {
    cout << a[i] << ' ';       // 使用循环逐个输出数组元素，并在元素后输出一个空格字符
}
```

2.3　常用的基础数据类型和数学运算

在编程时，我们会处理很多数据，而数据在计算机中的表示需要依赖各种数据类型。同时，对数据的操作离不开各种运算，包括基本的加、减、乘、除以及更复杂的数学函数。

2.3.1　基本数据类型

在编写代码解决问题或参与竞赛时，我们必须明确要处理的数据类型。选择合适的数据类型可以简化问题，而错误的选择可能会导致错误的结果甚至程序无法编译。

- 整型（int）：在算法竞赛中，整型用得非常多，它用于存储整数。例如，在进行计数或作为数组的索引时，整型是不可或缺的选择。如果整数的数值范围超过了32位int的数值范围（−2, 147, 483, 648~2, 147, 483, 647），可以使用长整型（long long），它的数值范围是−9, 223, 372, 036, 854, 775, 808~9, 223, 372, 036, 854, 775, 807。
- 浮点型（float、double）：浮点型用于存储小数，包括正数、负数和零。当需要精确到小数点后几位时，就需要用到此类型。在算法竞赛中，double的使用更为频繁，因为它的精度高于float。
- 字符型（char）：字符型用于存储单个字符。例如，在涉及字符比较、查找或需要存储大量字符等需求时，就会用到字符型。字符数组构成字符串，可以处理更复杂的字符数据。
- 布尔型（bool）：布尔型只有两个值：真（true）和假（false）。在处理逻辑判断时，布尔型非常有用。例如，要判断一个数是否为质数，可以定义一个布尔型变量来存储结果。

此外，标准模板库（STL）提供了更多强大的数据类型，例如vector、map、string和set，能够帮助选手编写更高效的算法和实现复杂的数据结构，详见第4章。

2.3.2　常用的数学运算

基本运算和数学函数是编写算法的基本工具，理解并熟练使用这些工具能够帮助我们高效地解决问题。

- 基本运算：加法（+）、减法（−）、乘法（*）、除法（/）、取余（%）。需要注意的是，整数除法会直接舍弃小数部分，例如9/2的结果是4。取余运算（也称为取模运算）在处理循环问题或解决数论问题时，非常有用。
- 数学函数：C++的标准库提供了一些常用的数学函数，如sqrt（求平方根）、pow（求幂）、fabs（求绝对值）。这些函数在处理数学问题时非常有用。例如，在计算两点之间的欧几里得距离时，通常会用到sqrt函数。

【例1】假设有一个坐标平面，两个点的坐标分别为($x1$, $y1$)和($x2$, $y2$)，我们需要计算两点之间的距离

可以使用下面的距离（distance）公式：

$$\text{distance} = \sqrt{(x2-x1)^2 + (y2-y1)^2}$$

对应的C++实现代码如下：

```
#include <cmath> // 引入cmath库，用于计算平方根和幂运算

// 计算两点之间的距离
double distance = sqrt(pow(x2-x1, 2) + pow(y2-y1, 2));
```

对于平方、立方等计算，为了效率和精度，建议直接使用变量相乘而非pow函数。

```
#include <cmath> // 引入cmath库，用于计算平方根和幂运算
double dx = x2 - x1, dy = y2 - y1;
double distance = sqrt(dx * dx + dy * dy);
```

【例2】若干基本数学运算的整合

```
#include <iostream>
#include <cmath> // 引入cmath头文件以使用数学函数
using namespace std;

int main() {
    // 声明变量
    int a = 5;
    int b = 3;
    double c = 2.5;
    double d = 1.5;

    // 基本算术运算
    int sum = a + b;           // 加法
    int difference = a - b;    // 减法
    int product = a * b;       // 乘法
    int quotient = a / b;      // 整数除法
    int remainder = a % b;     // 求余数

    // 输出基本算术运算结果
    cout << "Sum: " << sum << endl;
    cout << "Difference: " << difference << endl;
    cout << "Product: " << product << endl;
    cout << "Quotient: " << quotient << endl;
    cout << "Remainder: " << remainder << endl;

    // 高级数学运算
    double power = pow(c, d);      // 乘方
    double squareRoot = sqrt(c);   // 平方根
    double naturalLog = log(c);    // 自然对数
```

```
    double commonLog = log10(c);   // 常用对数
    double sine = sin(c);          // 正弦
    double cosine = cos(c);        // 余弦
    double tangent = tan(c);       // 正切

    // 输出高级数学运算结果
    cout << "Power: " << power << endl;
    cout << "Square Root: " << squareRoot << endl;
    cout << "Natural Log: " << naturalLog << endl;
    cout << "Common Log: " << commonLog << endl;
    cout << "Sine: " << sine << endl;
    cout << "Cosine: " << cosine << endl;
    cout << "Tangent: " << tangent << endl;

    return 0;
}
```

2.4　分支语句

在编程时，我们经常需要根据条件来决定执行哪些代码，这就需要用到分支语句。正确并熟练地使用分支语句是每个竞赛选手的基本技能之一。

2.4.1　if 语句

if语句是C++中最常见的分支语句之一，用于根据条件进行分支控制。其基本形式如下：

```
// condition是需要满足的条件
if (condition) {
    // codes to be executed if condition is true
    // 条件满足后，就会执行此处花括号中的语句块
}else{
    // 条件不满足，执行这里的语句块
}
```

其中，condition是一个布尔表达式，如果其值为true，则执行if块中的代码；否则执行else块中的代码。例如，我们可以用if语句来检查一个数是否为奇数：

```
// 定义一个整数变量num，并赋值为3
int num = 3;
// 使用 if 语句检查 num 是否为奇数
if (num % 2 == 1) {
    // 如果 num 除以2的余数等于1，则说明它是奇数
    // 输出 num 是奇数的信息
    cout << num << " is an odd number.";
}
```

如果要在condition中使用复合条件，则需要利用逻辑运算符&&和||进行逻辑运算。例如，若希

望同时满足条件c_1和c_2，则可以这样编写：

```
if(c1 && c2){
    // 当c1、c2同时为真时，执行这里的语句块
}
```

如果希望c_1和c_2两个条件只要有一个为真时，可以这样编写：

```
if(c1 || c2){
    // 当c1和c2中至少一个为真时，执行这里的语句块
}
```

如果想要构造更复杂的条件，可以配合圆括号来编写（圆括号的优先级最高，类似于四则运算中的圆括号）。需要注意的是，在C++的条件表达式中，不存在数学中常用的方括号（[]），所有括号都用圆括号（()）：

```
if(c1 || (c2 && c3)){
    // 当c1为真，或(c2, c3)同时为真时，执行这里的语句块
}else if(c2){
    // 当上一个条件不满足，但是满足c2为真时，执行这里的语句块
}
```

&&和||是逻辑运算符。

&&是“逻辑与”运算符，只有当两个操作数都为真（true）时，结果才为真。如果某一个操作数为假（false），那么无论另一个操作数是什么值，结果都是假。

||是“逻辑或”运算符，只要两个操作数中有一个为真，结果就为真。如果某一个操作数为真，那么无论另一个操作数是什么值，结果都是真。

在算法竞赛中，if语句被广泛用于控制程序的流程，例如在动态规划中，我们需要根据特定条件来更新状态值，或者在图论中，根据结点是否已访问来决定是否进一步探索等。

以下是一些if语句的示例程序。

【例1】根据年龄判断是否成年

程序代码如下：

```
// 定义一个整数变量 age，并赋值为18
int age = 18;

// 使用 if 语句检查 age 是否大于或等于18
if (age >= 18) {
    // 如果 age 大于或等于18，则输出提示信息
    cout << "你已经成年了。" << endl;
}else{
    cout << "你还没成年。" << endl;
}
```

【例 2】根据分数判断成绩等级

程序代码如下：

```
// 定义一个整数变量score，并赋值为85
int score = 85;

// 使用 if-else 语句判断分数等级
if (score >= 90) {
    // 如果分数大于或等于90，则输出"Grade: A"
    cout << "Grade: A" << endl;
} else if (score >= 80) {
    // 如果分数大于或等于80且小于90，则输出"Grade: B"
    cout << "Grade: B" << endl;
} else if (score >= 70) {
    // 如果分数大于或等于70且小于80，则输出"Grade: C"
    cout << "Grade: C" << endl;
} else if (score >= 60) {
    // 如果分数大于或等于60且小于70，则输出"Grade: D"
    cout << "Grade: D" << endl;
} else {
    // 如果分数小于60，则输出"Grade: F"
    cout << "Grade: F" << endl;
}
```

2.4.2　三目运算符

除if语句外，C++还提供了一种更为简洁的分支控制方式，那就是三目运算符（?:），也被称为三元运算符。它的形式如下：

```
condition ? expression_if_true : expression_if_false
```

如果condition为真（true），则表达式的值为expression_if_true，否则为expression_if_false。

例如，我们可以使用三目运算符来找出两个数中的较大值：

```
int a = 3, b = 4;
int max_value = a > b ? a : b;    // 返回4
```

请务必注意，三目运算符会返回一个值，而不是执行一条语句，同时需要保证冒号两边的值类型相同。

```
// 此处关于函数的知识将在后续讲解
// 返回两个数中较大的数
int getMax(int a, int b)
{
    // 以下写法是错误的，因为三目运算符会返回一个值，而不是像if那样执行一段语句
    a > b ? return a : return b;
    // 以下写法也是错误的，冒号两边需要有相同的数据类型
    return a > b ? a : “Hello”;

    // 以下是正确的写法
```

```
    return a > b ? a : b;
}
```

三目运算符可以使代码更为简洁，增强可读性。但在某些场景下，过度使用三目运算符可能会让代码变得更加难以理解（尽可能避免嵌套使用三目运算符）。因此，我们应当谨慎使用，以防止代码变得难以阅读和维护。

2.5　循环语句

在解决问题和优化算法的过程中，我们经常需要多次执行某段代码，这时就需要用到循环语句。正确地使用循环语句可以极大地提高代码的效率和可读性。

2.5.1　for 循环

for循环是C++中最常见的循环结构之一，它的基本格式如下：

```
for (initialization 初始化条件; condition 循环执行条件; increment 步长) {
    // 将要被执行的代码
}
```

在for循环中，我们首先进行初始化操作，然后检查条件是否满足。如果条件满足，则执行循环体中的代码，接着进行自增操作，再次检查条件，如此往复，直到条件不满足为止。

例如，我们可以使用for循环来打印出0~9这10个数字：

```
// 使用for循环打印数字0~9，每个数字后面跟一个空格
for (int i = 0; i < 10; ++i) {
    cout << i << " "; // 输出当前数字i和一个空格
}
// 打印结果：0 1 2 3 4 5 6 7 8 9
```

在算法竞赛中，for循环被广泛用于重复执行某项任务，例如遍历数组或向量（vector），或者执行固定次数的操作等。

嵌套循环也非常重要，但要注意，嵌套循环可能会极大地增加时间复杂度，因此需要谨慎使用。例如，我们要枚举所有二元组$(a,b),(a,b\in[1,n],a<b)$，可以这么编写：

```
// 此处外层的花括号可以省略，因为后面只有一条for语句
// 遍历所有可能的i和j的组合，其中i<j且i,j的范围是1~n（包括n）
for(int i = 1; i <= n; ++i)             // 外层循环，从1开始递增到n
    for(int j = i + 1; j <= n; ++j)     // 内层循环，从i+1开始递增到n
    {
        // 当满足条件时，输出i和j的值，用空格分隔，并换行
        cout << i << ' ' << j << '\n';
     }
```

更多时候，我们会将循环和分支语句结合使用，读者可在本书后面的示例代码中逐渐理解。

实际上，还有一种分支语句为switch，但在算法竞赛中几乎用不着，因此此处不作详细介绍，

其功能完全可以通过if…else语法来实现。

2.5.2　while 循环

while循环是另一种常见的循环结构，它的基本格式如下：

```
while (condition) {
    // 只要condition满足，代码块中的代码就会反复执行
}
```

while循环会在条件满足的情况下不断执行循环体中的代码，直到条件不再满足为止。

例如，我们可以使用while循环来找出不大于给定值的最大的2的幂：

```
int val = 100;
int power = 1;
while (power * 2 <= val) {
    // 只要power * 2 <= val条件满足，就让power乘以2
    power *= 2;
}
// 当退出while循环时，一定满足条件：power * 2 > val
// 此时的power是最大的满足power <= val的2的幂
cout << "The largest power of 2 less than or equal to " << val << " is " << power <<
".";
```

在算法竞赛中，while循环被广泛用于处理需要根据条件动态决定次数的任务。例如，在不确定目标何时满足的情况下，重复尝试某个操作，或者在需要逐步缩小范围以达到目标的算法（如二分搜索等）中，while循环也非常有用。

还有一种语法是do…while，但在实际编程中用得不多，并且其应用场景几乎可以被while循环代替。此处不再赘述，读者可以自行了解。

2.6　数组

数组是编程中最基础且最常用的数据结构之一，它可以存储多个同类型的元素。在算法竞赛中，我们经常使用数组来保存和操作数据。

2.6.1　数组的结构

数组是一种线性数据结构，可以存储固定数量的同类型元素。数组中的元素在内存中是连续存储的，每个元素都有一个索引，用于唯一标识该元素。索引从0开始，但在大多数情况下，0号位置是空置的，从1号位置开始，这样做是为了方便编写代码。

例如，我们可以声明一个包含5个整数的数组（见图2-1）：

```
int arr[5];
```

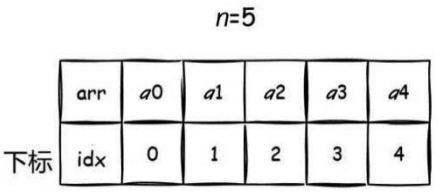

图 2-1　包含 5 个整数的数组

在这个数组中，arr[0]是第一个元素，arr[4]是最后一个元素。这种计算方法称为0−index0，但是在算法竞赛中，我们有时会使用1−index1，即将arr[1]当作第一个元素。实际上，使用哪种方法并不会有太大差别，读者可以根据个人偏好和熟练程度选择其中一种。

2.6.2　开辟数组空间

1. 数组的声明

在C++中，我们可以在声明数组时确定其大小。例如，声明一个包含10个整数的数组：

```
int arr[10];
```

注意，数组的大小必须是常量，不能使用变量来确定数组的大小。如果想创建一个可变长度的数组，可以使用STL中的vector容器。

在不同位置开辟的数组，其初始值也有所不同。在栈区（即函数内部）开辟的数组，数组元素的初始化值是未定义的，可能是随机混乱的值或内存中残余的垃圾值。而在堆区（即全局或静态区）开辟的数组，所有元素会自动将每字节都归零（即自动初始化为0）。

示例代码如下：

```
const int N = 10;
int a[N];        // 全局变量，a={0, 0, 0, 0, …}

int main()
{
    int b[N];    // 局部变量，b={-7, 5, -23, 3, -1966129712, …}，内容仅为示例，实际是一些随机的值
    return 0;
}
```

上述代码首先定义了一个常量整数N，其值为10。然后，在全局作用域中声明了一个名为a的整型数组，其大小为N（即10）。由于a是全局变量且未显式初始化，因此数组a的所有元素都被默认初始化为0。

在main函数内部，声明了一个名为b的整型数组，大小同样为N。与全局变量a不同，局部变量b没有被显式初始化，因此它的初始值是不确定的。这些值可能是内存中的任意垃圾值，也可能是编译器自动分配的值。在这个示例程序中，注释中提到的随机值仅用于说明，实际上这些值是不确定的。

这段代码展示了如何声明和使用全局和局部数组，并说明了它们的初始化差异。全局数组a被初始化为全零，而局部数组b的元素值是不确定的。

2. 二维数组

在实际使用中，还有一个非常重要的数组类型——二维数组，其结构如图2-2所示。

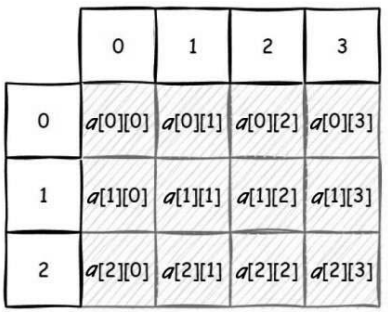

图 2-2　一个 3 行 4 列的二维数组

例如，以下代码用来创建一个二维数组，并输出数组的一个元素：

```
const int N = 1003;
const int M = 2003;
int a[N][M];              // 创建一个1003 × 2003大小的二维数组

cout << a[1][2] << '\n'; // 输出数组a的第1行第2列的元素
```

上述代码首先定义了两个常量N和M，分别赋值为1003和2003。然后声明了一个名为a的二维数组，其大小为1003行2003列。最后，使用cout输出数组a的第1行第2列的元素，即a[1][2]。

在算法竞赛中开数组时，我们一般会多开几个空间，例如题目要求1000大小，我们就可以开到1003,1005,1007等大小，这在编程时会提供一些便利。

3. 字符数组

关于数组，还有一种字符数组，即C原生字符串，值得了解。字符数组实际上是一个char类型的数组，每个元素占据一个字节（Byte）的存储空间。假如想存储一个长度为n的字符串，需要创建n+1大小的字符数组，如图2-3所示。

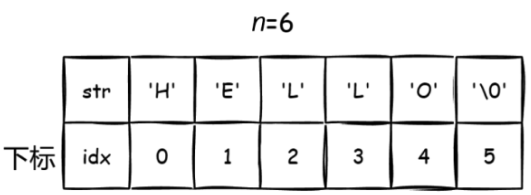

图 2-3　字符数组

因为字符数组的最后一位存储的是结束符'\0'，它的出现意味着字符串到此为止。在C/C++中，用单引号引起来的都是字符常量。字符数组有多种初始化方式，相比其他数组，它更为灵活。

```
char s[] = "Hello";  // 字符串长度为5，但字符数组长度为6

// 表示将字符串输入str[1]开始的字符数组中
// 注意此时需要创建n+2大小的字符数组，因为第0位没有被使用
char str[N];
cin >> str + 1;
```

上述代码将输入的字符串存储到一个字符数组中，但从数组的第1个位置（索引为1）开始存储，而不是从第0个位置开始。这样做的目的是让数组的第0位保持空白，或用于其他目的。

首先，定义了一个字符数组str，其长度为N。然后，使用`cin >> str + 1;`语句将输入的字符串存储到数组str中，从第1个位置开始。这里的+1表示将输入的字符串存储到数组的第1个位置，而不是第0个位置。

需要注意的是，为了确保有足够的空间存储输入的字符串以及可能的空字符（字符串结尾的'\0'），数组str的大小应该至少为N+2。这样，即使输入的字符串长度为N−1，仍然可以在数组末尾添加一个空字符作为字符串结束标志。

2.6.3　数组元素初始化

当我们声明数组时，可以同时对其进行初始化。例如：

```
int arr[5] = {1, 2, 3, 4, 5};
```

如果只初始化数组的部分元素，未被初始化的元素将被赋予默认值（对于int类型，默认值为0）。例如：

```
int arr[5] = {1, 2};                    // 剩余的元素置为0
//arr = {1, 2, 0, 0, 0};
```

如果想要将所有元素都初始化为同一值，可以使用std::fill函数。例如：

```
int arr[5];
std::fill(arr, arr+5, 1);               // 将[arr, arr + 5]这段内存的值初始化为1
```

当然，最常用的方法是直接用for循环对数组进行初始化，这样最为灵活。例如：

```
int arr[5];
for(int i = 0;i < 5; ++ i)arr[i] = 0; // 将数组arr初始化为0

int a[N][M];
// 初始化二维数组
// 将二维数组的[1 ~ n][1 ~ m]区域初始化为0
for(int i = 1;i <= n; ++ i)
    for(int j = 1;j <= m; ++ j)
        a[i][j] = 0;
```

2.6.4　数组和指针的关系

在C++中，数组名是一个指向数组第一个元素的指针。例如：

```
int arr[5] = {1, 2, 3, 4, 5};
int *p = arr;
```

此时，p指向数组arr的第一个元素，我们可以通过解引用指针来访问元素：

```
cout << *p;          // 打印结果：1
cout << *(p+1);      // 打印结果：2
```

需要注意的是，尽管数组名可以作为指针使用，但数组名并不是一个真正的指针，它是一个常量指针，不能更改其值。例如，以下代码是错误的：

```
int arr[5], arr2[5];
arr = arr2;          // error: array type 'int [5]' is not assignable
```

在计算机编程中，指针是一种变量类型，它存储了某个变量的内存地址。通过指针，我们可以间接地访问和操作该内存地址上的数据。

在C++中，指针的使用非常普遍，因为它们提供了一种高效的方式来处理内存和数据结构，尤其是在数组、字符串和动态分配的内存处理中。

指针变量的声明包括指针的类型和指针变量的名称。例如，`int* p;`声明了一个指向int类型的指针变量p。要使指针指向一个特定的变量，可以将变量的地址赋给指针，代码如下：

```
int value = 10;
int* p = &value;     // 指针p现在存储了变量value的地址
```

通过解引用指针（使用*操作符），我们可以访问或修改指针所指向的值：

```
cout << *p;          // 输出10，因为p指向value
*p = 20;             // 通过指针p修改value的值
cout << value;       // 输出20
```

指针加上一个整数可以访问连续内存块中的其他元素，这对于处理数组特别有用。例如，如果我们有一个整数数组，则可以创建一个指向数组第一个元素的指针，并通过增加指针值来访问数组中的其他元素：

```
int arr[] = {1, 2, 3, 4, 5};
int* p = arr;        // p指向数组的第一个元素

cout << *p;          // 输出1
cout << *(p+1);      // 输出2
cout << *(p+2);      // 输出3
// 以此类推
```

理解指针的概念对于成为一名熟练的C++程序员至关重要，因为它们在许多高级编程技术和库中都发挥着核心作用。

2.7　函数

函数是一个封装了可以执行特定任务的代码块，它接受一些参数并返回一个值。

在算法竞赛中，我们经常使用函数来提高代码的可读性和重用性，减少代码冗余，并以模块化方式解决问题。

有些算法必须使用函数来实现，或者说，使用函数实现会更加高效，例如递归、DFS搜索、最短路径等。

2.7.1　函数的声明和实现

函数的声明包括函数名、返回类型、参数列表等。例如，下面是一个名为sum的函数，它接受两个整数作为参数，并返回它们的和：

```
int sum(int a, int b);
```

这是函数的声明，告诉编译器有一个名为sum的函数，但并没有提供具体的实现。函数的实现包括函数的具体代码，其格式如下：

```
函数声明和实现格式为：
[返回类型] [函数名](参数表)
{
    函数体
}
```

示例如下：

```
int sum(int a, int b) {
    return a + b;
}
```

在这里，我们实现了sum函数，它返回a和b的和。函数的声明和实现可以在同一个地方，也可以分开。如果分开，通常将函数的声明放在头文件中，函数的实现放在源文件中。

需要注意的是，一个函数只能返回一次，一旦执行了return语句，函数就会终止，函数中的后续代码将不再执行。

2.7.2　函数的调用

当我们需要使用函数时，可以通过函数名和参数列表来调用它。例如，可以调用上面的sum函数来计算1和2的和：

```
int result = sum(1, 2);
```

在这里，sum(1, 2)就是函数的调用，它会返回1和2的和，然后将结果赋值给result。

2.7.3　Lambda 函数

C++11引入了一种新的函数类型：Lambda函数，也称为匿名函数。Lambda函数可以在需要函数但不想定义函数的地方使用。

例如，我们可以使用Lambda函数来定义一个自定义的排序规则：

```
vector<int> v = {3, 1, 4, 1, 5, 9};
sort(v.begin(), v.end(), [](int a, int b) {
    return a > b; // 降序排序
});
```

在这里，我们定义了一个Lambda函数[](int a, int b) { return a > b; }，它接受两个整数作为参数，并返回它们的比较结果。然后将这个Lambda函数传递给sort函数，告诉它我们需要按照降序排序。

Lambda函数可以捕获外部变量，使得代码更加灵活。例如，我们可以定义一个Lambda函数，它捕获了外部的变量x：

```
int x = 10;
auto is_less_than_x = [x](int a) {
    return a < x;
};
```

在这里，is_less_than_x是一个函数，它接受一个整数a，返回a是否小于x。这段代码使用了auto关键字自动推导函数的返回值类型，这个特性需要在C++11标准以上才能实现，否则会报错。但是请注意，尽管使用了auto进行自动推导，返回值类型也只能是一种，不能“在不同的函数出口返回不同的类型”。

Lambda函数的捕获列表还可以采用其他方式，例如使用“=”表示通过值捕获所有变量的拷贝，而使用“&”表示通过引用捕获所有外部变量。

2.8　结构体

结构体是一种可以存储多个不同类型数据的数据结构，对于封装和操作复杂的数据非常有用。在算法竞赛中，我们经常使用结构体来表示复杂的数据，如点、矩形、图的边等。

2.8.1　结构体的定义

在C++中，可以使用struct关键字来定义结构体。结构体的定义包括结构体名和一系列成员。例如，我们定义一个表示二维点的结构体：

```
// 定义一个名为point的结构体，表示二维空间中的一个点
struct point {
    int x, y; // 结构体有两个整型成员x和y，分别代表点的x坐标和y坐标
};
```

在这里，我们定义了一个名为point的结构体，它有两个整型成员x和y。

我们可以创建一个point类型的对象p，并初始化它的成员：

```
point p;
p.x = 1; // 设置p的x坐标为1
p.y = 2; // 设置p的y坐标为2
```

或者可以在创建对象时直接初始化成员，注意这里内部元素的类型必须和结构体定义时完全对应：

```
point p = {1, 2};
```

在上述例子中，我们定义了一个表示二维空间点的简单结构体，并展示了如何创建该结构体的实例以及如何初始化它们的成员。

2.8.2　结构体数组

与其他类型一样，我们也可以创建一个结构体的数组。例如，创建一个point的数组：

```
point points[10];
```

在这个数组中，每个元素都是一个point的对象。我们可以初始化数组中的元素，例如：

```
for (int i = 0; i < 10; i++) {
   points[i].x = i;
   points[i].y = i * i;
}
```

上述for循环用于初始化一个名为points的数组。数组中的每个元素都是一个具有x和y属性的对象。循环从0开始，直到小于10，每次迭代都会执行以下操作：

- 将当前索引i的值赋给points[i].x。
- 将当前索引i的平方值赋给points[i].y。

最终，数组points将被填充为以下形式：

```
points[0].x = 0; points[0].y = 0;
points[1].x = 1; points[1].y = 1;
points[2].x = 2; points[2].y = 4;
points[3].x = 3; points[3].y = 9;
…
points[9].x = 9; points[9].y = 81;
```

我们也可以在创建数组时直接初始化元素，例如：

```
point points[5] = {{0, 0}, {1, 1}, {2, 4}, {3, 9}, {4, 16}};
```

在算法竞赛中，结构体的使用是非常广泛的。掌握结构体的定义和使用，可以帮助我们更有效地表示和处理复杂的数据。

在星码StarryCoding网站上有语法基础的相关课程，有需要的读者可以了解一下。

2.9　推荐代码规范

在算法竞赛中，了解一些代码规范，可以显著加快我们编程和解决问题的速度。本节推荐一些在竞赛中可能会用到的代码规范。

2.9.1　使用头文件 bits/stdc++.h

在算法竞赛中，通常需要使用各种库的函数，如<iostream>用于输入/输出、<algorithm>用于一些通用算法（排序、查找等）、<vector>用于动态数组等。这些库分布在各种各样的头文件中。如果我们需要在每个程序中逐一包含这些头文件，不仅非常麻烦，而且在紧张的竞赛中可能会遗漏一些必要的头文件。为了解决这个问题，GCC编译器提供了一个非标准的头文件bits/stdc++.h，它包括几乎所有的标准库头文件，只需一行代码即可：

```
#include<bits/stdc++.h>
```

使用这个头文件的好处是可以节省许多编写头文件的时间，而且不会忘记包含必要的头文件。但请注意，这是GCC特有的头文件，不适用于所有编译器。

2.9.2　使用 std 命名空间

C++的标准库中的所有内容都放在std这个名字空间中。这意味着我们在使用这些内容时，通常需要加上std::前缀，如std::cout、std::endl等。为了避免这种麻烦，我们可以使用`using namespace std;`让std名字空间中的所有内容在全局范围内可见，这样可以直接使用这些内容，而不必每次都加上std::前缀：

```
using namespace std;
```

然而，使用全局的`using namespace std;`可能会引起名字冲突，因此在大型项目中，这种做法并不推荐。但在算法竞赛中，为了方便和提高编程速度，我们通常会使用这种方法。

2.9.3　代码缩进规范

良好的缩进是保持代码清晰易读的关键。在算法竞赛中的C++代码，我们通常使用4个空格进行缩进。使用空格进行缩进而不是制表符（Tab）的原因是，不同的编辑器对制表符的显示可能不同，而空格的显示是统一的。

其实，如何书写缩进并不严格要求，按照自己的习惯即可，不必过于纠结。

例如：

```
for (int i = 0; i < n; i++) {
    if (i % 2 == 0) {
```

```
            cout << i << " is even." << endl;
        } else {
            cout << i << " is odd." << endl;
        }
    }
```

在这个例子中，我们使用了4个空格的缩进，使得代码的结构一目了然。

2.9.4　代码换行规范

合理的代码换行可以使代码更加清晰易读。我们通常在每个语句后以及每个代码块的开始和结束处进行换行。例如：

```
for (int i = 0; i < n; i++) {
    cout << i;
    if (i < n - 1) {
        cout << ", ";
    }
}
cout << endl;
```

在这个例子中，我们在每个语句后都进行了换行，使得代码易于阅读。我们也可以在一些较长的表达式中适当地进行换行，以提高代码的可读性。

2.9.5　for 循环规范

for循环是经常使用的一种控制结构。在使用for循环时，需要明确循环变量的起始值、终止条件和更新方式。为了防止出错，通常在for循环内部只使用循环变量，而不修改它：

```
for (int i = 0; i < n; i++) {
    // do something with i
}
```

在这个例子中，清楚地指出了循环变量i的起始值（0）、终止条件（i < n）和更新方式（i++）。

在循环体内，只使用i，而不修改它，这样可以防止一些由于误修改循环变量导致的错误，当然，当你对算法足够熟悉时，可以为了便利而修改循环变量。

2.9.6　使用 longlong 类型是好习惯

在处理大整数时，需要注意整型溢出的问题。int类型的数值范围为−2 147 483 648~2 147 483 647，如果我们的数超过这个范围，就会出现溢出，导致结果错误。在处理可能会超过int数值范围的整数时，通常使用long long类型，其数值范围为−9 223 372 036 854 775 808~9 223 372 036 854 775 807，足够大：

```
long long a = 1e18;
```

在这个例子中，使用long long类型来存储一个非常大的数。这样可以防止溢出，保证结果的正确性。

2.9.7　不要过分压行

虽然压行可以使代码看起来更短，但过分地压行会降低代码的可读性。应该在保证代码简洁的同时，尽量使代码清晰易读。

例如，可以使用空行来分隔不同的代码块，使代码结构更加明显。

```
// 过分压行的代码，可读性非常差
#include <iostream>
using namespace std;
int f(int n,int a=0,int b=1){return n<2?a:f(n-1,b,a+b);}
int main(){int n; cin>>n; cout<<f(n)<<endl; return 0;}

//可读性较好的代码

#include <iostream>
using namespace std;

int f(int n){
    return n <= 2 ? 1 : f(n - 1) + f(n - 2);
}

int main()
{
    int n;
    cin >> n;
    cout << f(n) << endl;
    return 0;
}
```

2.9.8　不要轻易使用宏定义

虽然宏定义可以简化代码，但过度使用宏定义可能会导致代码难以理解，而且容易出错。应该谨慎使用宏定义，只在必要时使用。例如，可以用宏定义来简化一些常用操作，但应避免用宏定义来改变语言的基本语法，这可能会使代码难以阅读和维护。

举一个具体的例子，以下代码你能发现什么问题吗？

```
#define N 1e5 + 5
int a[N], t[N * 4];  // t申请N的4倍大小的空间
```

初看代码似乎没有问题，但提交后会遇到运行时错误（RE），这是因为发生了数组越界。错误的原因在于为数组t分配的内存空间不足。问题出在宏定义N的外面没有加上括号。由于宏定义本质上是文本替换，因此在定义数组t时，其大小计算实际上是$10^5+5\times4=10^5+20$，而不是我们预期的$(10^5+5)\times4$。

2.9.9　适当撰写注释

在算法竞赛的紧张环境下，撰写注释看似会消耗宝贵的几秒钟，但这短暂的时间损失并不会直接导致你无法解题。重要的是，当你对某些算法不够熟悉时，详尽的注释可以起到极大的帮助。

例如，在进行特殊判断后，应当注明变量的预期范围；在一系列复杂的while循环操作之后，记录下此时已经满足的关键条件等。这样做不仅有助于你在回顾代码时快速理解和调试，也方便参赛队友快速把握你代码的核心逻辑。

示例如下：

```
if(n <= 2){
    // 特判
    return;
}

// 此时有n > 2
for(int i = 1, j = 0;i <= n; ++ i)
{
    while(j < n && !cnt[a[j + 1]])cnt[a[++ j]] ++;
    // 此时[i, j]为合法区间
    // 对[i, j]的操作
}

for(int i = 2;i <= n; ++ i)
    if(a[i] < a[i - 1])
        return;
// 此时a数组满足升序
```

对于任何想要提高编程能力的人来说，明确理解和掌握编程规范是十分重要的。

编程规范的目标是提高代码的可读性和可维护性，减少错误，并提高编程效率。它们包括但不限于合理地使用头文件、命名空间、缩进、换行，以及恰当地使用循环、数据类型和控制结构等。以下是一些值得注意的细节：

- 使用bits/stdc++.h头文件能够帮助我们省去写许多其他头文件的时间，并避免在忙碌的编程过程中遗漏某些必要的头文件。
- 使用std命名空间可以方便我们在全局范围内直接使用C++标准库中的所有内容，而不必每次都加上std::前缀。
- 保持良好的代码缩进和合理的代码换行，可以使代码结构清晰，提高代码的可读性。
- 当处理可能超过int类型数值范围的大整数时，习惯使用long long类型，以避免整数溢出的问题。
- 避免过度压行和过度使用宏定义，这些做法可能会让代码变得难以理解和维护。

2.10　语法练习题

以下语法练习题较为简单，请读者自行实现和检验，本书不提供答案。

（1）数组计算：输入n（3≤n≤10）个整数，求出其中最大的3个数字，并计算这n个整数的平均值和中位数。

（2）圆的计算：输入一个圆的半径r，求出其面积和周长（用浮点数表示）。

（3）字符串压缩：实现一个函数，接收一个字符串作为输入，输出其压缩后的形式。例如，输入"AAABBBCCDAA"，输出"3A3B2C1D2A"。

（4）斐波那契数列：编写一个程序，生成斐波那契数列的前n（1≤n≤10^5）项。

（5）素数生成器：编写一个程序，生成并打印出前n（1≤n≤100）个素数。