Advances in Education 教育进展, 2026, 16(9), 152-161
Published Online September 2026 in Hans. https://www.hanspub.org/journal/ae
https://doi.org/10.12677/ae.2026.1691889

Hans汉斯

基于认知负荷的C语言程序设计关键节点教学案例资源设计研究

田鹏旭，王梦荫，兰　艳，刘海涛

大连民族大学信息与通信工程学院，辽宁　大连

收稿日期：2026年7月31日；录用日期：2026年8月28日；发布日期：2026年9月4日

摘　要

C语言程序设计是大学低年级学生学习程序设计思想和计算思维的重要基础课程，但由于课程内容涉及语法规则、程序执行过程、内存分配和算法实现等多类知识要素，学生在学习过程中容易出现“课堂能听懂、独立编程易出错”的现象。为识别课程学习中的关键认知负荷节点，本文基于近三届学生在线闯关平台中的历史学习数据进行分析，采用Apriori关联规则算法挖掘失败知识点之间的共现关系，识别构成学生学习中的关键负荷节点。并基于认知负荷理论，从内在认知负荷、外在认知负荷和相关认知负荷三个角度解释学生出错原因，并围绕关键节点重构教学案例资源，提出配套教学案例等资源设计方案。研究为C语言程序设计课程案例库优化和关键知识点精准教学提供了数据驱动的设计思路。

关键词

认知负荷理论，C语言程序设计，Apriori算法，教学案例资源

Design of Teaching Case Resources for Key Learning Nodes in C Programming Based on Cognitive Load Theory

Pengxu Tian, Mengyin Wang, Yan Lan, Haitao Liu

College of Information and Communication Engineering, Dalian Minzu University, Dalian Liaoning

Received: July 31, 2026; accepted: August 28, 2026; published: September 4, 2026

Abstract

C Programming is a fundamental course for lower-year university students to develop programming

文章引用: 田鹏旭, 王梦荫, 兰艳, 刘海涛. 基于认知负荷的 C 语言程序设计关键节点教学案例资源设计研究[J]. 教育进展, 2026, 16(9): 152-161. DOI: 10.12677/ae.2026.1691889

concepts and computational thinking. However, as the course involves multiple types of knowledge, including syntax rules, program execution processes, memory allocation, and algorithm implementation, students often encounter the problem of being able to understand concepts in class but making frequent errors when programming independently. To identify key cognitive load nodes in the learning process, this study analyzes historical learning data collected from an online challenge-based learning platform over the past three cohorts of students. The Apriori association rule algorithm is employed to mine co-occurrence relationships among failed knowledge points and identify key cognitive load nodes in students' learning. Based on cognitive load theory, the causes of students' errors are further interpreted from the perspectives of intrinsic cognitive load, extraneous cognitive load, and germane cognitive load. Teaching case resources are then reorganized around the identified key nodes, and corresponding strategies for the design of teaching cases and related learning resources are proposed. This study provides a data-driven approach to optimizing the teaching case repository and supporting targeted instruction of key knowledge points in C Programming courses.

Keywords

Cognitive Load Theory, C Programming, Apriori Algorithm, Teaching Case Resources

1. 引言

C 语言程序设计是高等教育的基础课程。该课程不仅要求学生掌握基本语法，更要求学生理解程序执行顺序、变量变化过程、函数调用机制、数组存储结构和指针访问方式。对于初学者而言，计算思维的建立并不容易，与此同时，学生面临的知识点往往并不是孤立存在的，而是相互交织、层层递进的。

在实际教学中，学生普遍存在“上课能听懂，一写程序就出错”的现象。尤其是在循环嵌套、数组、函数、指针等章节，学生提交次数明显增加，调试时间也明显延长。这说明学生并非简单地“不熟练”，而是在某些关键节点上出现了认知负荷过高的问题。

传统教学中，教师通常根据经验判断哪些内容是难点，并据此增加例题或讲解时间。然而，这种方式存在两个不足。第一，教师能够感知学生“哪里难”，但难以精确判断学生错误之间的关联关系。第二，教学案例往往按照教材章节顺序安排，缺少对认知负荷突增节点的专门过渡设计[1]。

随着在线编程平台和闯关式教学的应用，学生的学习过程数据被较为完整地记录下来。学生在不同知识点中的提交次数、错误类型、通过状态和失败序列，为分析学习困难提供了数据基础。基于这些历史数据，可以进一步识别 C 语言学习中的关键认知负荷节点，并据此优化配套教学案例资源。

本文主要围绕以下问题展开研究：

1) C 语言程序设计课程中哪些知识点容易形成关键认知负荷节点？

2) 不同知识点的失败之间是否存在关联关系？

3) 如何根据历史闯关数据和认知负荷理论重构教学案例资源？

2. 理论基础与研究思路

2.1. 认知负荷理论

认知负荷理论[2]认为，学习者在完成学习任务时可利用的工作记忆资源是有限的。当学习任务所需

的信息加工量超过学习者能够承受的范围时，就容易产生认知超载，进而影响学习效果。认知负荷通常分为内在认知负荷、外在认知负荷和相关认知负荷三类[3]。其中，内在认知负荷来源于学习内容本身的复杂性，外在认知负荷来源于教学材料的组织和呈现方式，相关认知负荷则与学习者进行图式建构和知识内化的有效投入有关。

C 语言程序设计课程具有明显的高认知负荷特征。对于初学者而言，程序设计并不是简单记忆语法规则，而是需要同时理解问题抽象、程序执行顺序、变量变化过程、内存地址关系和算法实现方法。例如，在循环嵌套中，学生需要同时追踪内外层循环、循环变量变化和输出结果；在函数与指针学习中，学生需要同时理解形参、实参、变量地址和间接访问关系；在字符串处理中，学生还要综合处理字符数组、循环遍历、结束标志和边界控制等多个要素。

因此，C 语言教学中的关键问题不是单纯增加练习数量，而是要识别学生在哪些知识点上出现认知负荷突增，并通过案例拆解、过程可视化和前置任务强化等方式，降低不必要的外在认知负荷[4]，分解过高的内在认知负荷[5]，引导学生形成稳定的程序设计图式[6]。

2.2. Apriori 关联规则挖掘方法

为分析学生在不同闯关任务中的失败是否存在稳定关联[7]，本文采用 Apriori 算法[8]进行关联规则挖掘。Apriori 是一种典型的频繁项集挖掘算法，其基本思想是：如果某一组项目频繁同时出现，那么它们的子集也应当频繁出现；反之，如果某一组项目不是频繁项集，则包含它的更大项目集也不可能成为频繁项集。基于这一思想，Apriori 算法可以从大量事务记录中逐步筛选出频繁项集，并进一步生成关联规则。

在本文中，每名学生的闯关记录被视为一条事务，每个出现明显困难的闯关任务或错误标签被视为一个项目。例如，某学生在“循环边界控制”“数组下标越界”“字符串遍历错误”等任务中均出现多次提交失败，则可形成如下事务：

{循环边界控制，数组下标越界，字符串遍历错误}

Apriori 算法的作用不是判断某个知识点失败必然导致另一个知识点失败，而是从历史数据中发现“哪些失败经常共同出现”以及“哪些前序薄弱点可能与后续综合任务困难有关”。因此，本文将 Apriori 挖掘结果作为教学诊断和案例资源重构的辅助依据。

关联规则主要通过支持度、置信度和提升度进行评价。支持度表示某组失败项目在全部学生事务中共同出现的比例；置信度表示在出现前项失败的学生中，同时出现后项失败的比例；提升度表示前项失败对后项失败出现概率的提升程度。当某条规则同时具有较高支持度、置信度和提升度时，说明该规则具有较强的教学解释价值，可作为优化前置案例和拆解综合任务的重要依据。

2.3. 研究思路

本文的研究思路包括数据收集、关键节点识别、关联规则挖掘和案例资源重构四个阶段，见图 1。

第一，数据收集阶段。依托在线编程闯关平台，收集 2022~2024 三年学生 C 语言程序设计课程中的历史闯关记录，共计 1498 人，人均完成作业题目 150 道，包括闯关任务编号、知识点章节、提交次数、通过状态和首次通过时间等信息。此外，根据课程教学大纲、章节知识结构和闯关任务主要考查目标建立标签体系，由共同授课教师一同对每个闯关任务进行知识点标签和错误类型标签标注。例如，循环章节的任务可标注为“循环变量”、“循环边界”、“循环嵌套”等；数组章节的任务可标注为“数组下标”“数组越界”“二维数组行列关系”等；函数和指针相关任务可标注为“形参与实参”“地址传递”“解引用”“数组作为函数参数”等。对于标注存在分歧的任务，经讨论后形成一致结果。

Table 1. Typical diagnostic indicators in level-based learning data
表 1. 闯关数据中需要关注的典型诊断指标

数据指标	反映问题	可能的认知负荷解释
单题提交次数较高	学生反复试错	任务难度较高或案例过渡不足
首次通过率较低	多数学生首次无法完成	内在认知负荷较高
编译错误比例较高	语法表达困难	外在认知负荷或代码模板不足
答案错误比例较高	程序逻辑理解不足	程序执行图式尚未形成
运行时错误比例较高	越界、非法访问等问题	相关认知负荷较高
多知识点连续失败	前序知识影响后续任务	认知负荷累积

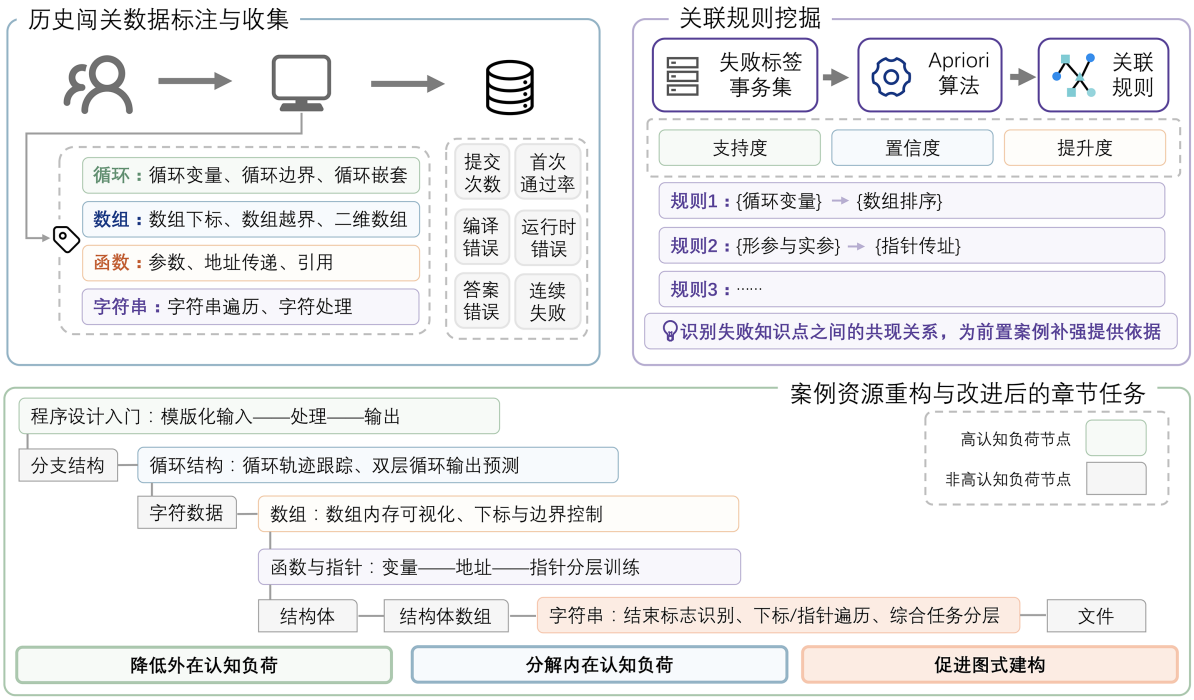

Figure 1. Research framework for teaching case resource design of key learning nodes in C programming based on cognitive load theory
图 1. 基于认知负荷的 C 语言程序设计关键节点教学案例资源设计研究思路

第二，关键认知负荷节点识别阶段。见表 1，本文以提交次数、首次通过率、错误类型比例和连续失败情况作为诊断指标。当某一知识点或任务标签出现提交次数升高、首次通过率下降、运行时错误或答案错误比例增加等现象时，说明学生在该节点需要投入更多认知资源。若错误率或提交次数在某一章节出现明显爬升，则可将其视为认知负荷突增节点。

第三，关联规则挖掘阶段。将学生在各任务中的失败标签构建为事务数据集，采用 Apriori 算法挖掘失败标签之间的关联规则。例如，如果“循环变量混乱”与“排序程序实现失败”经常共同出现，则说明排序案例之前需要加强循环轨迹训练；如果“函数形参与实参理解错误”与“指针传址调用失败”经常共同出现，则说明指针章节之前需要强化函数调用和内存地址之间的联系。在关联规则挖掘过程中，每名学生的失败知识点集合构成一条事务记录。若学生在某闯关任务的首次提交中未能通过，则将该任务所对应的知识点标签记入该学生的失败事务。本文采用支持度(Support)、置信度(Confidence)和提升度

(Lift)对关联规则进行筛选。综合考虑规则覆盖范围、规则数量及其教学解释价值，将最小支持度设置为0.10，最小置信度设置为0.60，并进一步要求规则提升度大于1.20。其中，支持度阈值用于排除仅在少量学生中出现的偶然性组合，置信度用于保证前项失败学生中存在较稳定的后项失败现象，提升度则用于排除由于后项本身具有较高失败率而形成的表面关联。在不同的教学背景下，以上三个参数可以根据基本情况进行变化，比如，需要更广法的挖掘潜在的共现失败情况时，可以降低支持度，以获取更多的关联规则，后续再根据具体需求进行规则分析。

第四，案例资源重构阶段。根据关键节点识别和关联规则挖掘结果，对现有案例库进行针对性优化。对于关键节点章节，分析认知负荷原因，对于闯关题目进行改造或者题目优化。对于关联规则挖掘结果，则对于闯关任务进行拆解和前置任务优化。

3. C 语言关键认知负荷节点分析

在关键认知负荷节点识别阶段，通过对于诊断指标进行统计，识别出了若干典型的认知负荷结点，本节将对典型的认知负荷节点形成情况进行剖析和讨论。

(1) 程序设计入门：从自然语言到程序表达的首次认知跃迁

C 语言程序设计的第一个关键认知负荷节点出现在课程开始阶段。对于初学者而言，他们往往尚未建立程序设计的基本认知，需要同时熟悉编译器和运行环境等。所以不可避免地会造成相关指标的浮动。

根据题目标签分析可知，学生常见问题包括：不理解变量在程序中的作用，混淆数学表达式与 C 语言表达式，以及难以将自然语言描述转化为程序语句。虽然这些内容从教师视角看属于基础知识，但对于完全没有编程经验的学生而言，其内在认知负荷并不低。

该阶段的认知负荷主要来自两个方面。一方面，学生需要完成从自然语言问题到程序表达的首次转换；另一方面，学生尚未形成“输入－处理－输出”的基本程序结构逻辑。因此，第一章案例设计不宜过早追求题目变化和综合性，而应通过模板化案例和输入输出对照示例，降低外在负荷，帮助学生建立最基本的程序运行模型。

(2) 循环嵌套：程序执行顺序回跳与变量动态变化导致的负荷突增

从历史闯关数据看，学生在顺序结构和选择结构中的整体表现相对稳定，但进入循环结构后，提交次数和答案错误比例开始明显增加。其中，循环嵌套任务表现尤为突出，说明学生在程序执行过程理解上出现了明显困难。

循环嵌套之所以成为关键节点，主要原因在于程序执行顺序发生了明显变化。在顺序结构中，程序基本按照从上到下的方式执行；在选择结构中，学生主要判断不同条件下执行哪一条路径；而循环结构引入了重复执行、条件回跳和变量动态变化。循环嵌套进一步形成“外层循环控制轮次、内层循环完成具体操作”的双层执行关系，显著增加了学生同时追踪多个变量和多个执行层次的认知负荷。

学生常见错误包括混淆内外层循环变量、无法判断内层循环执行次数、循环边界设置错误、变量初始化位置错误、无法根据题意确定循环结构等。从认知负荷角度看，循环嵌套的难点并不只是语法形式，而是学生尚未形成稳定的循环执行逻辑，需要在脑中反复模拟程序运行过程，导致认知负荷被大量占用。针对该情况，教学案例应循序渐进，在初期闯关任务中降低外部认知负荷，简化题干，将题目重点聚焦于建立对于循环结构的理解。而后再加深题目难度，逐步过渡到解决多变量循环问题、以及多变量多重循环问题。

(3) 数组：从单个变量到集合数据组织的抽象困难

数组是 C 语言学习中的重要转折点。学生在前期学习变量时，通常将变量理解为单个数据存储单元；而数组要求学生理解一组同类型数据的统一组织方式，并通过下标进行批量访问。这意味着学生需要从

“单个变量处理”转向“集合数据处理”，其抽象层次明显提高。

数组节点中的常见错误包括数组初始化不完整、循环访问范围错误、二维数组行列关系混淆、无法将实际问题中的一组数据映射为数组结构等。尤其是在数组与循环结合的任务中，学生既要理解数组下标的有效范围，又要控制循环变量的变化，一旦循环边界设置不当，就容易引发答案错误或运行时错误。

因此，数组章节的重点不应只放在数组定义和基本使用上，还应突出数组和循环结构之间的关系。与指针章节不同，本节更强调数组作为集合数据组织方式所带来的抽象困难，以及数组下标和循环边界之间的关联。

(4) 函数与指针：从值传递到地址传递的模型建构困难

函数与指针之间存在密切联系，是学生理解 C 语言内存机制的重要节点。学生在初学函数时，通常能够模仿函数定义与调用的基本格式，但对形参、实参以及函数调用过程中数据如何传递的理解并不充分。尤其是在需要通过函数修改外部变量时，学生容易将普通变量的值传递与指针参数的地址传递混淆。

该节点的核心困难不在于指针语法本身，而在于学生没有把函数调用过程与内存地址关系建立联系。例如，在交换两个变量值的任务中，学生常常认为在函数内部交换形参就能够改变实参，却没有意识到普通变量作为参数时传递的是值的副本。进入指针参数后，学生又需要理解取地址、指针变量、解引用和间接修改变量之间的关系，认知负荷进一步增加。

学生常见错误包括：误认为形参变化会自动影响实参，不理解为什么需要传入变量地址，混淆指针和普通变量的含义，不能正确使用“&”和“*”，以及在指针参数函数中出现类型不匹配或非法访问等问题。

因此，本节点应重点围绕“函数调用－变量副本－变量地址－指针参数－间接修改”这一认知链条进行案例设计，而不是简单将指针作为孤立语法讲解。

(5) 字符串处理：循环、数组、指针和边界控制的综合负荷

字符串处理是 C 语言学习中的综合性高负荷节点。与普通数组相比，字符串不仅具有字符数组的存储形式，还涉及结束标志、字符串输入输出函数、字符遍历、数组空间和指针移动等内容。学生在该节点中出现的问题，往往不是单一知识点造成的，而是循环、数组和指针多个前序知识点未能充分整合的结果。

学生常见错误包括忽略字符串结束标志、循环遍历条件错误、字符数组空间不足、字符串复制或连接时越界、使用指针遍历字符串时指针移动错误、混淆字符和字符串等。尤其是在字符串综合处理任务中，学生需要同时判断循环何时结束、当前访问哪个字符、目标数组是否越界、指针是否正确移动，这会显著增加内在认知负荷。

因此，字符串章节不宜直接进入复杂综合题，而应按照“字符数组存储－结束标志识别－下标遍历－指针遍历－综合操作”的顺序逐级展开，使学生逐步整合前序知识，避免在综合任务中一次性认知超载。

4. 基于 Apriori 的失败知识点关联分析

通过 Apriori 算法对于失败标签之间的关联规则进行挖掘，可以得到各个知识点之间的潜在关联，为教学案例的系统优化提供理论支撑，从以往的“经验教学”转向“数据驱动”的教学模式。本文讨论算法挖掘到的若干典型关联规则。

(1) 关联规则一：循环嵌套失败与排序类任务失败

规则 1：{循环变量} → {数组排序}

该规则说明，在循环变量和循环边界控制上存在困难的学生，后续在排序类程序中更容易出现失败。

排序任务通常需要双层循环、数组访问、条件判断和数据交换共同完成。如果学生尚未形成稳定的循环执行逻辑，就会在排序任务中同时面对过多认知元素。

从认知负荷角度看，排序程序的失败不完全来自排序算法本身，也与循环嵌套理解不足密切相关。学生如果无法判断外层循环和内层循环分别控制什么，就很难进一步理解“每一轮比较后最大值逐步后移”这一过程。因此，在排序案例之前，应增加循环轨迹跟踪任务，让学生先通过表格记录循环变量、数组元素和每轮结果的变化，再进入完整排序程序编写。

对应的案例优化方向是：在排序案例前增加“循环变量跟踪表”、“双层循环输出预测”“4 个元素排序纸笔演算”和“半成品排序代码填空”等前置任务，降低学生直接面对完整排序程序时的认知负荷。

(2) 关联规则二：函数参数理解错误与指针传址调用失败

规则 2：{形参与实参} → {指针传址}

该规则说明，在函数形参与实参关系上理解不充分的学生，后续在指针传址调用任务中更容易出现失败。函数参数传递是连接函数与指针的重要中间节点。学生如果只停留在函数调用的语法形式上，而没有理解函数调用时实参值如何传递给形参，就很难进一步理解为什么某些任务必须通过变量地址和指针参数完成。

该规则在教学中最典型的表现是“交换两个变量值”任务。学生往往先写出如下逻辑：在函数内部交换两个形参的值，并期待主函数中的两个变量也发生改变。但实际上，普通变量作为函数参数时传递的是值的副本，函数内部修改形参并不会改变实参。只有当函数接收变量地址，并通过指针进行间接访问时，才能修改主函数中的变量。

从认知负荷角度看，学生在该节点上的困难并不是单纯“不懂指针”，而是没有建立“函数调用过程－变量副本－变量地址－指针参数－间接修改”之间的完整模型。也就是说，指针传址调用的负荷来源于函数机制和内存地址机制的叠加。

对应的案例优化方向是：在指针参数案例之前，先设计普通变量传参失败案例，再设计传入变量地址的修正案例。通过对比 swap(a,b)与 swap(&a,&b)的运行结果，引导学生理解值传递与地址传递的区别。同时配套绘制函数调用前、调用中、调用后的变量关系图，使学生看到形参、实参和指针指向对象之间的联系。

(3) 关联规则三：指针基础错误与字符串综合操作失败

规则 3：{指针引用，循环终止条件} → {字符串综合}

该规则说明，同时在指针访问和循环终止条件上存在困难的学生，在字符串综合处理任务中更容易失败。字符串处理需要通过循环逐个访问字符，同时判断字符串结束标志和数组边界。如果学生对指针移动、字符访问和循环终止条件理解不足，就容易出现越界访问、漏读字符或结果错误等问题。

从认知负荷角度看，字符串处理是循环、数组和指针多个知识点综合作用的结果。学生在处理字符串时，不仅要知道字符数组的存储方式，还要理解'\0'的作用，并在循环中正确控制下标或指针移动。任何一个前序知识点没有形成稳定图式[9]，都会在字符串综合任务中集中暴露。

对应的案例优化方向是：将字符串综合处理拆解为“字符数组存储观察－结束标志识别－下标方式遍历－指针方式遍历－字符串复制或查找综合任务”五个层级。通过逐层递进的方式，帮助学生将循环、数组和指针知识整合起来，避免直接进入复杂字符串综合题造成认知超载。

5. 关键节点配套教学案例资源设计

5.1. 循环嵌套章节：设计“循环轨迹跟踪”案例

针对循环嵌套中的执行顺序理解困难，设计循环轨迹跟踪案例。案例不要求学生一开始直接写完整

代码，而是先完成程序执行过程的纸笔追踪。案例设计可分为三步：

1) 第一步，给出单层循环代码，让学生填写每一轮循环中变量 i 和值 sum 的变化。

2) 第二步，给出双层循环代码，让学生填写外层变量 i、内层变量 j 和输出结果之间的关系。

3) 第三步，给出简单图形输出任务，例如输出矩形、三角形或乘法表，引导学生理解外层循环控制行、内层循环控制列。

该案例的目标是降低学生对循环执行顺序的外在认知负荷，帮助学生形成“循环变量–执行次数–输出结果”之间的稳定图式。

5.2. 数组章节：设计“数组内存可视化”案例

针对数组学习中的抽象困难，设计数组内存可视化案例。该案例要求学生在写代码之前，先画出数组元素在内存中的连续存储示意图。案例任务包括：

1) 标注数组 a[0]到 a[4]的元素值和地址。

2) 观察循环变量 i 与数组下标之间的关系。

3) 分析数组求和、查找最大值等程序中每一步访问的数组元素。

该案例的目标是帮助学生将数组从“多个变量的集合”理解为“连续存储空间”，从而为后续指针与数组、函数数组参数学习奠定基础。

5.3. 函数章节：设计“数组作为函数参数”的对照案例

针对函数中使用数组时出现的错误，设计普通变量传参与数组传参的对照案例。案例分为两组：

1) 第一组展示普通变量作为函数参数时，函数内部修改形参不会影响实参。

2) 第二组展示数组作为函数参数时，函数内部修改数组元素会影响原数组。

通过对照，学生可以直观看到普通变量传参与数组传参的差异。随后引入数组名表示首地址的解释，并通过图示说明形参数组和实参数组访问的是同一段连续存储空间。该案例的目标是降低学生在函数参数传递中的内在认知负荷，避免学生将普通变量传参经验错误迁移到数组参数中。

5.4. 指针章节：设计“变量–地址–指针”分层案例

针对指针学习中的抽象困难，设计分层递进案例。

1) 第一层，只讨论普通变量和地址，要求学生理解变量值和变量地址之间的区别。

2) 第二层，引入指针变量，要求学生画出指针变量保存地址、指向普通变量的关系。

3) 第三层，引入引用操作，要求学生解释*p 访问的是谁。

4) 第四层，引入指针与数组关系，要求学生比较 a[i]和*(a + i)的等价关系。

该案例的目标是避免学生一开始就面对复杂指针程序，而是逐步建立变量、地址、指针和数组之间的关系。

5.5. 字符串章节：设计“边界控制与指针遍历”案例

针对字符串处理中容易出现的越界和指针访问错误，设计字符串边界控制案例。案例分为五个层次：

1) 第一层，观察字符数组和字符串的存储差异，明确字符串末尾存在结束标志'\0'。

2) 第二层，用下标方式遍历字符串，统计字符个数。

3) 第三层，用指针方式遍历字符串，观察指针每次移动后指向的字符。

4) 第四层，设计字符串复制和查找任务，强调目标数组空间和循环终止条件。

5) 第五层，设计字符串综合处理任务，例如统计元音字母、查找指定字符、删除指定字符等。

该案例的目标是将字符串综合操作拆解为若干低负荷任务，使学生逐步整合循环、数组和指针知识，避免在综合任务中一次性认知超载。

通过案例调整，结合新一届学生闯关数据显示，部分关键节点的首次通过率(提升 7%)和平均提交次数(平均降低了 0.8 次)呈现改善趋势，初步说明基于关键认知负荷节点进行案例重构具有一定可行性。

6. 教学实施与教学反思

上述案例资源并非作为独立的补充练习集中使用，而是根据关键认知负荷节点嵌入相应章节的课堂教学和在线闯关过程。在循环嵌套教学中，将“循环轨迹跟踪”安排在完整程序设计任务之前，通过变量变化记录和程序执行过程分析，帮助学生建立内外层循环之间的执行关系；在函数与指针教学中，将普通变量传参与地址传递的对照案例作为知识衔接任务，使学生在进入指针参数之前理解形参、实参与变量地址之间的关系；在字符串教学中，则将综合任务拆分为字符数组存储、结束标志识别、下标遍历、指针遍历和综合操作等不同层次，并分别用于课堂演示、前置训练和在线闯关。通过这种方式，将案例资源由原有的章节配套练习转变为围绕关键认知负荷节点进行的分层教学支持。结合新一届学生的闯关过程数据可以发现，经过案例调整后，部分关键知识点的首次通过情况、重复提交次数和典型错误发生情况呈现出改善趋势，说明针对前序薄弱知识进行强化、对高负荷综合任务进行拆解具有一定的教学应用价值。

教学实践表明，程序设计课程中的学习困难并不完全由当前章节知识本身造成，学生在综合任务中出现错误，往往与前序知识尚未形成稳定认知图式有关。例如，排序程序中的困难可能来自循环嵌套和数组访问能力不足，指针传址中的问题可能与函数参数传递机制理解不充分有关，而字符串综合任务则进一步叠加了循环、数组、指针和边界控制等多方面要求。因此，单纯按照教材章节顺序增加例题和练习数量，并不能有效解决知识之间累积形成的认知负荷。教学案例设计应从“增加练习量”逐步转向“识别关键节点–强化前置知识–拆解综合任务–逐步形成图式”的设计思路。同时，在线闯关平台所记录的提交次数、通过情况和错误类型等过程数据，也不应仅作为学生学习结果的记录工具，而可以进一步作为教师诊断教学难点和调整案例资源的重要依据。通过持续分析学生学习过程中的共性错误及其关联关系，可以使案例资源建设由教师经验驱动逐步转向数据与教学经验共同驱动，为课程内容的动态调整和精准教学提供支持。

7. 结语

本文围绕 C 语言程序设计课程中学生普遍存在的学习困难，提出了一种基于认知负荷诊断的关键节点教学案例资源设计思路。通过对历史闯关数据进行分析，本文识别出若干关键认知负荷节点，并利用 Apriori 算法挖掘失败知识点之间的关联关系。在此基础上，结合认知负荷理论，从程序执行顺序、内存模型理解和综合任务拆解三个角度解释学生出错原因，并设计了循环轨迹跟踪、数组内存可视化、函数参数传递对照、指针分层训练和字符串边界控制等教学案例资源。

研究表明，历史闯关数据能够为 C 语言程序设计课程案例库建设提供重要依据。教师不仅可以据此判断学生“哪里容易错”，还可以进一步分析“为什么会错”以及“教学案例应如何拆解”。该研究为 C 语言程序设计课程的精准教学、案例库优化和过程性学习支持提供了可操作的实践路径。

但同时，本研究仍存在一定局限性。首先，研究数据来源于同一课程在线闯关平台中近三届学生的学习记录，课程组织方式、学生基础和平台任务设置具有一定特殊性，因此在不同平台、课程上的适用情况仍需验证和讨论。其次，Apriori 算法所揭示的是不同失败知识点之间的统计关联关系，不能直接解释为知识点之间的因果关系。未来可结合更多课程和学习平台开展跨样本验证，并引入学生认知负荷量

表、学习过程访谈以及对照实验等方法，对关键节点识别结果及教学案例优化效果进行进一步验证。

基金项目

本文系 2025 年度国家民委高等教育教学改革研究课题“‘以铸牢中华民族共同体意识为主线’多学科助力 ICT 人才双创能力培养路径探索”(课题编号：2025-GMJ-002)的研究成果。

参考文献

[1] Chen, O., Castro-Alonso, J.C., Paas, F. and Sweller, J. (2017) Extending Cognitive Load Theory to Incorporate Working Memory Resource Depletion: Evidence from the Spacing Effect. *Educational Psychology Review*, **30**, 483-501. https://doi.org/10.1007/s10648-017-9426-2

[2] Sweller, J. (1988) Cognitive Load during Problem Solving: Effects on Learning. *Cognitive Science*, **12**, 257-285. https://doi.org/10.1207/s15516709cog1202_4

[3] 葛超宇, 袁徐庆. 认知负荷理论下互动影片的设计方法研究[J]. 新媒体研究, 2021, 7(2): 94-96.

[4] 杨磊, 司国东, 王春桃, 等. 基于认知负荷理论的计算机组成原理实验教学设计与应用[J]. 实验室研究与探索, 2025, 44(1): 177-181.

[5] Sweller, J. and Chandler, P. (1994) Why Some Material Is Difficult to Learn. *Cognition and Instruction*, **12**, 185-233. https://doi.org/10.1207/s1532690xci1203_1

[6] 胡定磊. 基于认知负荷理论的计算机网络课程教学优化研究[J]. 学周刊, 2025(10): 41-44.

[7] 孙运平, 高玉春. Web 前端课程认知负荷优化研究[J/OL]. 电脑与电信: 1-6. https://doi.org/10.15966/j.cnki.dnydx.20260311.001, 2026-08-21.

[8] Agrawal, R., Imieliński, T. and Swami, A. (1993) Mining Association Rules between Sets of Items in Large Databases. *Proceedings of the* 1993 *ACM SIGMOD International Conference on Management of Data*, Washington, 25-28 May 1993, 207-216. https://doi.org/10.1145/170035.170072

[9] 罗群英, 陈仕品, 张剑平. 基于认知负荷理论的网络课程设计: 以国家精品课程“现代教育技术”的网络课程为例[J]. 开放教育研究, 2009, 15(2): 61-66.