컴퓨터교육학회 논문지 2025년 제28권 제3호
https://doi.org/10.32431/kace.2025.28.3.001

자연언어에서 형식언어로의 변환: 대학의 프로그래밍 교육 방법 제안*

Conversion from Natural Language to Formal Language: A Proposal for Programming Education Methods in Universities

권오영† · 박은진††
Oh-Young Kwon† · Eun-Jin Park†

요약

누구나 생성형 인공지능을 활용하여 자신에게 필요한 컴퓨터 프로그램을 작성할 수 있지만, 그것을 제대로 사용하려면 프로그램을 이해할 수 있어야 한다. 그러나 대학의 많은 학생들은 프로그래밍 학습을 어려워한다. 이 어려움을 해결하기 위해 생각(사고)의 표현 도구로 사용되는 언어의 기능을 프로그래밍에 접목한 방법을 제안한다. 먼저, 학습자들은 직관적인 사고로 문제를 해결하는 자신의 방법을 자연어 문장으로 표현하고 기록한다. 체계화되지 않은 직관적 사고를 프로그래밍에 필요한 알고리즘적 사고로 전환하기 위해, 기록된 문장의 논리와 프로그램 요소가 알고리즘 문장으로 전환될 때까지 문장을 수정하고 보완한다. 완성된 알고리즘 문장을 컴퓨터 명령문으로 변환하여 프로그램을 완성한다.

주제어 컴퓨팅사고, 문제해결, 프로그래밍 교육, 프로그래밍 어려움, 프로그래밍 지식 융합

ABSTRACT

Anyone can use generative AI to write computer programs tailored to their needs, but to use it effectively, one must understand programming. However, many university students find it challenging to learn programming. To address this difficulty, we propose a method that integrates the functions of language, which serves as a tool for expressing thought, into programming. First, learners express and record their intuitive problem-solving methods in natural language sentences. To transform their unstructured intuitive thinking into the algorithmic thinking required for programming, they revise and refine the recorded sentences until the logic and program elements form coherent algorithmic statements. Finally, these completed algorithmic statements are converted into computer instructions to complete the program.

Keywords Computational Thinking, Problem-Solving, Programming Education, Programming Difficulties, Integration of Foundational Knowledge in Programming.

†정회원 한국기술교육대학교 융합학과 교수
††정회원 한국기술교육대학교 융합학과 겸임교수 (교신저자)

논문투고 2024년 10월 08일
심사완료 2025년 02월 18일
게재확정 2025년 02월 19일
발행일자 2025년 03월 05일

* 본 논문은 2023년 한국기술교육대학교 교수 교육연구 진흥과제 지원을 받아 수행된 연구임.

1. 서론

인공지능이 일상에서 널리 사용됨에 따라, 전문가의 도움 없이도 누구나 생성형 인공지능을 통해 프로그램을 작성할 수 있다[1]. 그러나 생성된 코드에는 일관성 문제나 규칙 위반이 있을 수 있기에, 사용자는 코드의 정확성과 적합성을 평가하고 수정할 수 있는 프로그래밍에 대한 이해가 필요하다[2]. 이러한 프로그래밍의 필요성은 대학이 미래를 준비하는 학생들에게 반드시 가져야 할 능력 중 하나로 컴퓨터를 이용한 문제해결 능력을 제시하는 데에도 나타난다[3].

대학에서 컴퓨터 프로그래밍 교육은 필수화 및 보편화되는 경향을 보인다. 이는 2023년 SW중심대학에서 전공과 관계없이 모든 1학년 학생들이 필수 과목으로 프로그래밍 과목을 수강하는 것과, 2022년 60개 대학의 기초문해 교양 교육 강좌 중 12.6%가 컴퓨터 문해와 관련된 과목인 것으로 확인된다[4].

컴퓨터 프로그래밍은 그 필요성과 해당 교과목의 필수성 및 보편성에도 불구하고, 학습 난도가 높고 숙달이 어려워 학습 초기 단계에서 약 50%에 달하는 높은 실패율과 중도 탈락률을 보인다[5, 6]. 이는 프로그래밍 학습자들이 직면한 어려움을 여실히 보여주며, 이러한 문제를 해결하기 위한 연구와 더불어 학습자에게 적합한 프로그래밍 교육 방법 제안 및 적용에 대한 연구가 필요함을 시사한다.

본 논문은 이러한 필요를 해결하기 위해 프로그래밍을 언어적 관점에서 학습자의 직관적인 문제해결 능력을 프로그램으로 변환하는 방법을 제안하며, 이를 실제 수업에 적용해 그 효과를 확인하는 것을 목적으로 연구되었다. 이를 위해 2023년 2학기 K대학교의 1학년 대상의 프로그래밍 관련 수업에서 학습자가 겪는 어려움의 현상과 원인을 파악하였으며, 제안한 방법을 실습수업에 적용하는 과정에서 제출된 학생들의 과제물 분석과 비교그룹과의 성적 비교를 통하여 제안한 방법의 유효성을 보인다.

2. 관련 연구

2.1 프로그램 작성 단계와 오류

컴퓨터 프로그램은 문제를 해결하기 위해 그 처리 방법과 순서를 기술하여 컴퓨터에 주어지는 일련의 명령문 집합체이다[7]. 프로그램 작성 과정은 문제해결 단계(problem solving)와 구현 단계(implementation)로 나뉘며, Fig. 1 과 같이 각 단계에서 수행되는 작업의 순서를 구분할 수 있다[8].

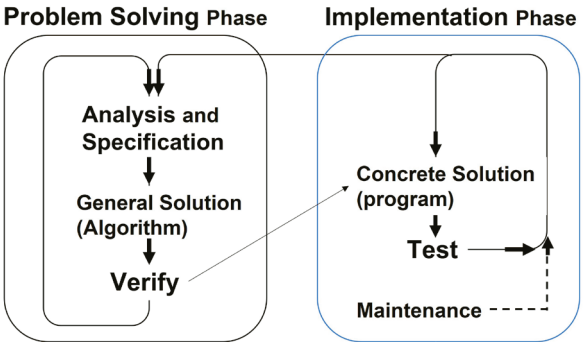

Figure 1. Programming process

문제해결 단계에서는 프로그래밍을 수행하기 위한 체계적 사고를 적용하는 단계로, 문제를 분석하고 알고리즘을 설계하여 해결 방안을 도출한다. 구현 단계에서는 도출된 해결 방법을 명령문으로 작성한다. 학습자가 겪는 어려움은 프로그램이 원하는 결과를 도출하지 못하거나 프로그램 오류(error)가 발생하는 경우로, 이는 예상치 못한 실행 과정, 논리적 문제, 알고리즘 오류 등의 원인에 의해 발생한다[9]. 오류는 구문오류와 결과 오류로 나뉘며, 결과오류는 논리오류와 런타임(runtime)오류로 구분된다[10].

구문오류가 없다면 프로그램을 실행하여 결과의 적합 여부를 테스트할 수 있으며, 목표한 결과가 나오지 않으면 결과오류가 발생한 것으로, 알고리즘 수행 절차가 올바르지 않을 때 많이 발생한다. 이는 문법적 지식이 필요조건일 뿐, 문제해결을 위한 충분조건이 아님을 의미한다. 논리오류는 프로그램의 제어 논리가 잘못되어 발생하며, 문법적 오류는 없지만 프로그램 흐름이 잘못되어 실행 불가능하거나 예상한 결과를 얻지 못할 때 발생한다[11]. 런타임오류의 예로는 0으로 나누기 등의 경우를 들 수 있다[12]. Fig. 2 는 프로그램 작성 단계에서 발생할 수 있는 오류와 오류의 종류(원인)에 따른 수정 작업의 대상(단계)을 구분하여 보여준다.

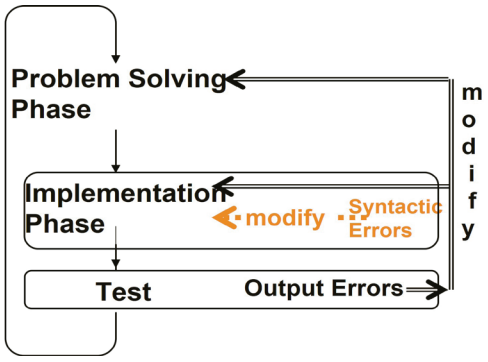

Figure 2. Program modification due to errors

논리오류는 문제해결 단계에서 알고리즘을 수정하고, 런타임 오류는 구현 단계에서 수정해야 한다. 프로그램 오류를 예방하고 해결하려면 각 단계에서 필요한 지식을 습득해야 한다.

2.2 프로그램 작성 단계별 필요 지식

2.2.1 문제해결 단계에 필요한 지식

컴퓨팅사고는 실세계의 문제를 컴퓨터를 사용해 해결하기 위한 사고 과정이다[13]. 김재경(2016)은 컴퓨팅사고와 프로그래밍 언어를 함께 학습한 그룹이 문제 분석 및 설계를 더 체계적으로 수행한 것을 확인하여, 컴퓨팅사고가 문제해결에 효과적임을 보여주었다[14].

프로그래밍 학습자가 사용하는 직관적 사고는 컴퓨터가 작업을 수행하는 방식과 다르며, 이를 프로그램으로 바르게 작성하려면 알고리즘적 사고가 필요하다. 알고리즘적 사고는 알고리즘을 구성하고 이해하는 데 필요한 능력으로 문제 분해, 과정 설계, 알고리즘 변환 능력을 포함한다[15]. 김동만 외(2020)는 인지 과학적 관점에서 초보 프로그래머인 학습자가 겪는 어려움의 원인을 개념적 지식의 부족으로 지목하였으며, 이는 의미 이해와 코드 작성에 필요한 설계 기술을 포함한다[16]. 추상 및 논리 개념 부족, 문제해결에 필요한 관련 지식 부족, 논리의 부정확성, 자연언어의 모호한 사용, 부정확한 적용 모델, 전략적 접근 방식의 경험 부족 등의 사전 지식의 부족은 실패율과 수업 포기로 이어지는 요인이 된다[17]. 프로그래밍에는 필수 지식인 프로그램 언어의 습득과 기반 지식인 문제해결 능력의 융합이 필요하다[18].

문제해결 단계에서 컴퓨팅사고, 알고리즘적 사고, 사전 지식, 개념적 지식 등이 필요하며, 해당 지식의 부재는 프로그램 작성의 어려움과 연결된다.

2.2.2 구현 단계에 필요한 지식: 언어적 관점에서

컴퓨터 프로그래밍 교육의 핵심은 형식언어와 자연언어 체계 간의 관계를 파악하고, 인간의 자연언어 논리와 의미를 형식언어의 형식적 논리와 의미로 대응시키는 능력을 기르는 것이다[19].

자연언어와 형식언어를 구분하는 특징을 살펴보면, 형식언어는 자연언어에 비해 문법이 간단하며 중의성이 없고 그 뜻이 명확하다[20]. 반면에 자연언어(인간의 언어)는 다양한 개념을 표현하지만, 예외 사항이 많으며 사용 방법이 복잡하다[21].

자연언어와 프로그래밍 언어 사이에는 의미적 차이로 인한 불일치가 존재한다. 이를 해결하기 위해 사전도구(dictionary tool)를 사용하여 자연언어와 그에 대응되는 형식언어를 키워드 맵 형식으로 등록하였다. 또한, 자연언어를 여러 기준으로 분류하였는데, 대표적으로 사용된 품사 기준 분류법은 명사와 동사를 구분하는 방식이다. 동사는 명사들 사이의 관계를 나타낸다고 보았고, 명사는 데이터와 관련되는 경우로 보았다[22].

문제에서 데이터를 구분하고 변수로 사용하는 것은 학습자들에게 두 가지 이점을 제공한다. 첫째, 학습자들이 프로그래밍의 시작을 무엇부터 해야 할지 고민하는 경우, 명사를 구별하여 프로그램에서 처리할 변수(variable)로 정의할 수 있다[23]. 둘째, 데이터 역할과 처리 요구 사항을 인지하고 구분하는 기준으로 삼을 수 있다[24].

Hermans 외(2017)는 “Programming is Writing is Programming”이라는 표현을 통해 글쓰기와 프로그래밍의 유사성을 강조하고 프로그래밍에 글쓰기 기술을 적용하였다[25]. 또한 글쓰기와 프로그래밍은 모두 고수준 아이디어를 저수준 문장이나 명령으로 번역하는 작업으로 보았다.

문제해결 방법은 자연언어나 의사코드를 사용하여 알고리즘으로 표현되며, 알고리즘은 컴퓨터 언어를 사용하여 프로그램으로 작성된다. 프로그램은 문제해결 방법을 자연어에서 형식언어로 바꾸어 표현한 것이므로, 언어적 관점에서 프로그래밍 과정을 보고 학습자의 어려움 해결의 실마리로 관련 연구 내용을 사용한다.

3. 프로그래밍 교과목과 학습자의 어려움

3.1 프로그래밍 교과목과 연구 대상

본 연구의 제안된 방법이 적용된 교과목과 수업 시간 및 교과 내용과 대상자는 다음과 같다.

컴퓨터 프로그래밍 관련 교과인 ‘문제해결과프로그래밍’ 은 필수 교과목으로, 모든 1학년 학생을 대상으로 하였다. 학습 목표는 문제해결에 사용되는 알고리즘을 적용하여 프로그램을 작성하고, 작성된 프로그램의 결과를 확인하며 검증하는 능력을 갖추는 것이다. 사용된 프로그래밍 언어는 파이썬이었으며, 1학기에 선수 과목인 ‘컴퓨팅사고’교과목을 통해 파이썬 언어를 수강하였다. 수강 시간은 한 학기 15주, 주당 4시간(이론, 실습 2시간씩)이 배정되었다. 이론 수업은 전공별로 진행되었으며, 실습은 분반별로 시행되었다. 수업 내용은 파이썬 문법 정리, 정보의 구조화(클래스), 알고리즘(분할정복, 그리디(Greedy), 정렬)과 마인크래프트를 이용한 기말 프로젝트로 구성되었다.

2023년 2학기 K대학교에서 진행된 본 연구는 에너지신소재화학공학부와 디자인ㆍ건축공학부 두 개 전공에서 전공별로 각각 2개 분반씩, 총 4개 분반(84명)을 연구 대상으로 선정하였다. 연구 대상 분반은 무작위로 선정되었으며, 해당 분반의 실습수업에서 나타난 학습자들의 어려움을 관찰하고 분석하였다. 그리고 제안된 프로그래밍 방법을 적용하여 프로그램을 작성하도록 하였다.

3.2 프로그래밍 학습자의 어려움

‘문제해결과프로그래밍’ 실습수업을 통해 프로그래밍 학습자들이 겪는 어려움을 파악한 결과, 여러 현상들이 확인되었다. 프로그래밍 실습 문제 중 ‘10개의 정수에서 가장 작은 값을 찾기’라는 간단한 문제를 해결하였던 학습자들은 숫자의 대소를 판단하는 학습된 직관을 사용하여 가장 작은 수를 쉽게 구분하였다. 이들은 왜 그 숫자가 가장 작은지에 대한 이유를 논리적으로 설명할 수 있었다. 문제를

인식하고 직관적으로 해결책을 도출하는 데에는 큰 어려움이 없었으나 직관적인 판단을 알고리즘적 사고로 전환하고, 이를 실제 프로그램 코드로 구현하는 과정에서는 상당한 어려움을 겪었다.

일부 학습자들의 경우에는 프로그램의 작성을 무엇부터 해야 할지 몰라 코딩(coding)을 시작도 하지 못하는 경우가 있었다. 이들은 프로그램을 작성하기 위한 기본적인 준비 작업, 예를 들어 변수 선언이나 적절한 명령문 선택에 대해 확신을 가지지 못해 코드를 반복적으로 수정하는 상황에 직면하기도 하였다. 또한, 일부 학습자들은 인터넷 검색을 통해 해당 문제의 해결책을 찾거나, 다른 학습자들의 코드를 참고하기도 하였다. 학습자들은 프로그래밍에 필요한 논리적 사고와 알고리즘적 접근법을 충분히 습득하지 못한 채, 프로그램을 작성하게 되었고, 명확한 방향을 잡지 못하는 어려움을 겪었다.

이 어려움을 프로그램 작성 과정에서 겪는 일시적인 현상으로 볼 수도 있지만, 그 근본적인 원인은 코딩 자체에 있는 것이 아니라, 문제해결 방법을 알고리즘으로 전환하는 데에 필요한 사고 전환 능력의 부재에 있는 것으로 판단되었다. 이를 해결하기 위해서는 코딩 과정뿐만 아니라, 문제를 알고리즘적으로 풀어내는 사고 훈련과 그 사고를 프로그램으로 변환하는 기술을 단계적으로 습득할 수 있는 교육적 접근이 필요하였다.

4. 프로그래밍 어려움 해결

4.1 프로그래밍 어려움 해결을 위한 방법 제안

프로그램 작성 방법을 학습하는 것은 단순히 코드를 작성하는 기술을 배우는 것이 아니라, 문제해결을 목적으로 사고(생각)하는 방식을 배우는 과정이다. 이 과정에서 프로그래밍 언어는 사고를 표현하는 도구로 사용된다. 도구를 제대로 사용하기 위해서는 프로그래밍 언어의 문법과 구조를 이해하는 것뿐만 아니라 문제 분석, 논리적 사고, 알고리즘 설계, 그리고 데이터 구조에 대한 이해 등이 필요하다. 다시 말해, 프로그래밍은 주어진 문제를 정의하고 해결책을 구조화하는 방법을 배우는 것이라고 할 수 있다. 프로그래밍 학습자의 어려움을 해결하기 위한 방법으로 Fig. 3 에 제시된 순서대로 프로그래밍을 수행할 것을 제안한다. Table 1 은 Fig. 3 의 수행 단계에서 사용되는 언어와 결과물이다.

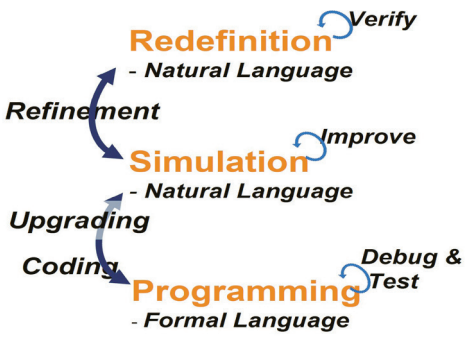

Figure 3. Suggested Programming Process

Table 1. Languages and Outputs in Programming Process

Step	Language	Outcomes
Redefinition	natural language	Problem understanding, Extraction of program elements
Simulation	natural language	Establishing logic
Programming	formal language	Program

프로그래밍 과정은 세 단계로 구성되며 원하는 결과를 얻을 때까지 각 단계를 재수행할 수 있다.

Redefinition 단계에서는 주어진 문제를 파악하여 그 문제를 학습자 자신의 문장으로 재정의하며, 문제에 제시된 데이터를 식별한다. 사용 언어는 자연어이며 결과물은 프로그램 요소가 포함된 재정의 문장이다. Simulation 단계에서는 문제해결 과정 설계를 위해 시뮬레이션을 수행하고, 문제해결 논리를 수립하기 위한 문장을 기술한다. 정제되고 논리적이며 체계적인 알고리즘적 사고의 기술을 목표로 한다. 사용 언어는 자연어이며 결과물은 문제해결 논리이다. Programming 단계에서는 알고리즘적 사고가 반영된 자연어 설명을 형식언어로 변환한다. 그 결과물은 프로그래밍 언어로 표현된 프로그램으로 컴퓨터가 실행할 수 있다. 본 연구는 Programming 단계에 앞선 Redefinition 단계와 Simulation 단계를 수행할 것을 제안한다.

4.1.1 문제 재정의: 문제 파악과 데이터 인지

주어진 문제를 해결하기 위해 문제가 무엇인지 파악해야 한다. 문제 파악을 위해 제시된 문제를 재정의한다. 재정의 문장에는 문제 내용 파악, 문제해결의 대상이 되는 데이터 인지, 결과로 나타날 데이터의 형식과 값 예측, 프로그램에서 변수로 사용해야 할 단어 등이 포함되어야 한다. 또한, 다른 해석의 여지가 없는 명확한 표현을 사용해야 하며, 관형사를 사용하는 경우라면 기준을 제시하여 모호함이 없도록 해야 한다.

재정의 방법은 문제를 작은 부분으로 나누고, 각 부분을 명확히 이해하는 것을 기본으로 하며, 문제를 문장으로 만들거나 스토리보드를 만드는 등의 구조화된 접근 방식을 사용한다. 그리고, 자신이 이해한 문제를 다른 사람에게

설명하는 과정도 포함한다. 문제 재정의는 여러 번의 수정과 검증 과정을 거치면서 정제된 문장으로 완성되어 간다. 교수자는 재정의 문장으로 학습자의 문제 이해 정도를 파악하고, 프로그램 완성도까지 예측할 수 있다.

4.1.2 시뮬레이션과 알고리즘

문제해결 과정의 설계와 논리 수립을 위해 시뮬레이션을 시행한다. 시뮬레이션을 수행한 학습자는 자신이 절차화되지 않은 직관적 사고를 사용하여 '자동 처리'하고 있다는 것을 인지할 필요가 있다. 왜냐하면 자신의 무의식적 사고 패턴이나 접근 방식을 인식하게 되면, 이를 의식적으로 조정하거나 사용할 수 있기 때문이다. 학습자 자신의 직관적인 문제해결 사고방식 인식을 돕기 위해, 문제해결 시뮬레이션 과정을 구두로 설명하고 문장으로 기록하게 한다. 오래된 기록의 도구인 종이와 연필의 사용은 학습자가 인터넷 등 외부 자료를 참조하지 않고 오로지 자신의 문제해결 능력에 집중할 수 있도록 돕는다. 학습자가 처음 작성한 문장은 완벽하지 않을 수 있지만, 글쓰기를 연습하듯이 문제해결 문장도 수정과 보완을 반복하는 과정을 통해 오류를 제거하고 자신만의 논리를 세울 수 있게 된다. 이때, 컴퓨터가 수행할 수 있는 명령 순서와 형식에 적합한 알고리즘적 사고로 전환하는 훈련이 필요하다. 이후, 학습자의 논리는 컴퓨터가 실행할 수 있는 명령 순서와 형식에 맞는 절차적 알고리즘으로 발전한다. 이렇게 작성된 문장을 컴퓨터 명령문으로 변환하면 실제 프로그램이 완성된다.

4.2 제안된 프로그래밍 방법 적용

이론 수업에서 다룬 알고리즘 문제들을 실습 시간에 제안된 방법을 적용하여 프로그래밍하도록 하였다. 그중 '분할정복 알고리즘' 단원의 '이분(이진) 탐색' 문제에 제안된 방법을 적용하였던 과정을 제시한다.

이진탐색은 정렬된 리스트에서 중앙값을 기준으로 범위를 절반씩 줄여가며 원하는 값을 찾는 탐색 방법으로, 반복적으로 중앙값을 비교하여 찾고자 하는 값이 중앙값보다 작으면 왼쪽 절반으로, 크면 오른쪽 절반으로 그 범위를 계속 좁혀가며 찾는다.

4.2.1 Redefinition 단계 적용

문제해결을 위해서는 문제 파악이 필수이지만, 제시된 문제를 읽고도 문제에 제시된 조건이나 제한 사항 등을 정확히 파악하지 못하는 학습자가 있었다. 다음은 수업 시간에 제출된 학습자들의 재정의 문장과 이에 대한 교수자의 피드백이다.

학습자들의 재정의 문장
학생A: 여러 방법 중 가장 효율적인 선택을 하는 것.
학생B: 중간값과 비교하여 수를 찾는 방법.
학생C: 주어진 리스트를 반으로 나누어 탐색 범위를 좁혀가며 원하는 값을 찾아내는 방법.
학생D: 어떤 정렬된 배열에서 중앙값(이때 중앙값은 리스트 인덱스의 가장 작은 값과 큰 값을 더한 것)을 기준으로 배열 전체에서 반을 탐색하고, 찾고자 하는 값과 중앙값(인덱스 아님)을 대조 비교해 가며 효율적으로 찾는 방법.

재정의 문장에 대한 교수자의 피드백
학생A: '효율적'의 기준을 알 수 없음
　　탐색 대상과 탐색할 값에 대한 설명 없음.
　　이진탐색 방법에 대한 설명 없음.
학생B: 탐색의 대상 데이터가 명확하지 않음.
　　중간값의 기준(크기, 차례, 평균값) 모호함.
학생C: 탐색에 필요한 데이터를 명확히 구분함.
　　반으로 나누는 방법은 설명되지 않음.
학생D: '효율적'의 기준을 알 수 없음
　　탐색의 대상 값과 탐색 값을 정확히 구분함.

재정의 문장을 분석하면 학생들의 이해도 정도를 알 수 있었다. 학생A는 대상 데이터와 처리 과정을 서술하지 않아, 문제를 파악하지 못한 것으로 판단하였다. 학생B는 탐색 대상 데이터를 명시하지 않았으며 중간값의 기준이 제시되지 않았다. 학생C는 대상 데이터를 제시하였으나, 탐색 범위를 좁히는 방법에 대해 언급하지 않았다. 반면 학생D는 대상 데이터를 명확히 구별하고, 문제해결 방법을 자세하고 정확하게 기술하였다.

학습자들의 프로그램을 보면 문제 재정의가 명확하고 문제에 대한 이해가 구체적일수록 프로그램 명령문의 완성도가 높은 것을 알 수 있다. Fig. 4 는 학생A의 프로그램이다. 변수 a, b가 사용되었으나 그 값을 알 수 없다. 조건문이 완성되지 않고 비어 있다.

```
myList= [0, 9, 10, 15, 18, 20, 29, 35, 41]
while True:
  if a < b :

  elif a > b:

  else a == b:
```

Figure 4. Program of 'Student A'

Fig. 5 는 학생C의 프로그램이다. 필요한 변수들이 정의되었고 조건식이 논리 구조에 맞게 사용되었다. 재정의 문장에서 구체적으로 설명되지 않았던 탐색 범위를 좁히는 방법은 프로그램으로 제시되어 있다.

```
myList= [1, 3, 13,20, 27]
x=3
low = 0
high = len(myList) - 1
while low <= high :
  mid = (low + high) // 2
  if myList[mid] == x :
    print(f"{x}를 {mid}번 째에서 찾음")
    break
  elif myList[mid] < x :
    low = mid + 1
  else:
    high = mid - 1
```

Figure 5. Program of 'Student C'

4.2.2 시뮬레이션 단계 적용과 프로그래밍

문제해결 시뮬레이션 문장은 데이터 처리 과정의 반복 시행으로 수행 속도가 느렸지만, 문제해결 과정을 상세히 파악할 수 있게 했다. 교수자는 이 과정에 '느리게 생각하기'라는 이름을 붙이고 학습자들에게 자신만의 문제해결 방법과 논리를 세워 프로그램으로 작성할 수 있음을 강조하였다.

Fig. 6 은 이분 탐색이 어떻게 동작하는지를 이해하기 위해 학습자들이 숫자 카드를 사용하여 시뮬레이션하는 장면이다. 데이터는 카드에 기재하였고, 데이터의 위치는 색종이의 위치 이동으로 표시하였다.

Figure 6. Binary Search Simulation Using Number Cards

Fig. 7 은 학습자가 문제를 해결하는 과정을 설명하여 기록한 것으로, 문장에서 명사, 동사를 구별하고 프로그램에서 데이터로 쓰일 부분과 연산할 부분을 밑줄로 표시하였다.

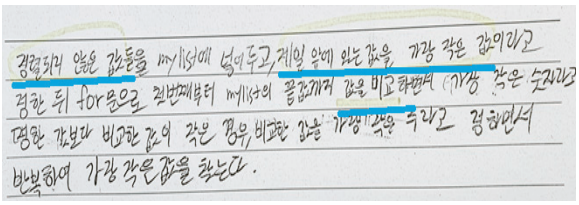

Figure 7. Description of Problem-Solving Methods and Data Separation

다음은 Fig. 7 의 일부 글귀에 대응되는 프로그램 명령문으로, 파이썬 언어의 들여쓰기를 적용한 것이다.

- 정렬되지 않은 값들을 myList에 넣어두고,
  myList=[22,7,15,1,57,39,8,11,72,99]
- 제일 앞에 있는 값을 가장 작은 값이라고 정한 뒤
  min=myList[0]
- for 문으로 첫 번째부터 myList의 끝값까지
  for k in myList:
- 값을 비교하면서 가장 작은 숫자라고 명한 값보다 비교한 값이 작은 경우,
  if min > k :
-비교한 값을 가장 작은 수라고 정하면서 반복하여
  min = k
-가장 작은 값을 찾는다.
  print(min)

프로그래밍 경험이 부족한 초보 학습자들에게 과제를 제시하여 문제해결 프로그램을 작성할 기회를 제공하였다. 과제는 프로그램 코드의 중요한 부분에 주석 추가하기, 프로그램의 전체적인 구조를 설명하는 보고서 작성하기, 프로그램 설명 동영상 제작하기 등이었다.

4.3 제안된 프로그래밍 방법 적용 결과

본 연구는 '문제해결과프로그래밍' 교과의 2개 전공 총 167명 중에서 84명의 학생들을 대상으로 제안된 프로그래밍 방법을 적용하여 실습수업을 수행하였으며, 본 연구에서 제안한 프로그래밍 방법을 적용한 그룹과 비적용 그룹의 성적을 비교하였다. 비교된 성적은 2023년 2학기 성적이며, 그 내용은 Table 2 와 같다. 전공 X에서 A등급을 받은 28명 중 64.3%에 해당하는 18명은 제안된 프로그래밍 방법론으로 학습한 학생들이었다. 이는 비교그룹의 35.7%에 비해 약 1.8배 높은 비율이다. 전공 Y의 A, B등급과 전공 X의 B등급에서는 제안된 프로그래밍 방법으로 학습한 그룹과 비교그룹 간의 비율이 44~50% 범위로 유사하게 나타났으며, 두 그룹 간에 큰 차이를 보이지 않았다. 전공 X의 A등급 비율에서 보이는 비교그룹과의 차이는 제안된 방법의 잠재적 효과성을 보여주지만, 제안된 방법 외에도 전공이나 구성원의 특성, 수업 분위기 등의 다른 요소들이 시험 성적에 영향을 미칠 수 있으므로 추가적인 연구와 데이터를 통해 검증할 필요가 있다.

Table 2. Grade Comparison

Major	Participants Rated/Total	Grade Counts Rated/Total (Rated Percentage)	
		A Grade	B Grade
X	41 / 82	18/28 (64.3%)	13/29 (44.8%)
Y	43 / 85	14/29 (48.3%)	15/30 (50%)

위와 같은 한 학기 수업의 결과만으로는 제안된 프로그래밍 방법의 효과를 단정 짓기 어려우므로, 학생들이 작성한 재정의 문장에서 문제해결 프로그램에 필요한 것으로 판단되는 요소를 분별하여 제안된 방법의 유효성을 정성적으로 확인하고자 한다.

문제는 이진탐색 문제이며 재정의 문장에 이진탐색의 필요한 요소(정렬된 리스트, 중앙값 사용, 범위 축소, 반복)가 포함되었는지를 판별한다. 또한, 필수 요소의 포함 여부와 성적, 성적과 구체적(절차적) 표현의 연관성을 제시한다. 다음은 학생들의 재정의 문장을 그대로 옮긴 것으로, 띄어쓰기와 오탈자 등을 수정하지 않았음과 성적별 무작위 선택을 밝힌다.

학생A: 정렬된 리스트에서 찾고자 하는 값을 찾기 위해 중간값을 기준으로 작은지를 비교하여 절반씩 줄여가 빠르게 찾을 수 있는 탐색방법.

학생 B: 정렬되어 있는 숫자들에서 원하는 수를 찾는 방법 중 하나로, 가운데 위치에 있는 수와 원하는 수를 비교해 범위를 절반씩 좁히며 찾는 방법.

학생C: 내가 찾고자 하는 숫자를 숫자리스트의 중앙값과 비교한 후 리스트의 범위를 줄여가며 찾는 방법.

학생D: 정렬된 오름차순 데이터 값들에 대해, 중간값과 위 비교를 통해 범위를 줄여나가 특정 데이터를 차는 과정

학생E: 탐색범위를 오름차순으로 정렬한 후, 중간값과 찾아야하는 값을 비교하며 범위의 반씩 줄여나가는 탐색 방법

학생F: 이진탐색은 정렬된 자료에서 중앙값을 찾고 탐색값과 비교하여 중앙값이 탐색값보다 크면 탐색범위를 처음부터 중앙값 이전수 까지로 줄이고, 반대로 중앙값이 탐색값보다 작으면 탐새범위를 중앙값 다음 수부터 끝까지로 줄여서 중앙값이 탐색값과 같을때까지 반복한다.

학생G: 작은 값과 가장 큰 값의 중앙값을 찾아 원하는 값과 비교하여 범위를 줄여나가는 탐색 방식.

학생H: 찾고자하는값을 중간값과 비교하며 범위를 좁혀가는 탐색방법.

학생I: 끝값들의 중간위치 값을 살피며 범위를 줄여가며 원하는 수를 찾는 탐색이다.

학생J: 주어진 리스트에서 중간값과 구하고자 하는 값을 비교하여 반으로 줄이며 찾아간다.

학생K: 중간값과 비교하여 수를 찾는 방법.

학생L: 첫번째 값과 마지막 값을 더해 2로 나눠 찾고자 하는 값을 찾는 탐색 방법

Table 3 은 재정의 문장에 나타난 이진탐색의 필수 요소의 포함 여부를 나타낸다.

Table 3. Indication of the Inclusion of the Essentials of Binary Search in the Redefined Sentence

Student	Sorted List	Use of the median	Range reduction	iteration
A	o	o	o	x
B	o	o	o	x
C	x	o	o	x
D	o	o	o	x
E	o	o	o	x
F	o	o	o	o
G	x	o	o	x
H	x	o	o	x
I	x	o	o	x
J	x	o	o	x
K	x	o	x	x
L	x	o	x	x

학생A부터 학생F까지의 문장은 이진 탐색 알고리즘을 세부적으로 설명하여 문장이 다소 길며, 탐색의 구체적인 과정을 명확하게 설명하고 있으며, 해당 학생들은 A등급을 받았다. 학생G부터 학생L까지의 문장은 핵심 아이디어를 간략하고 직관적으로 표현하였으나 구체적인 과정이나 알고리즘의 흐름을 상세하게 설명하지는 않으며, 해당 학생들은 B등급 이하를 받았다.

재정의 문장에 이진탐색의 필수 요소가 포함되어 있고 탐색의 과정이 구체적이며 알고리즘적 사고가 나타난 문장을 작성한 학생이 높은 등급을 받은 것으로 판단된다.

향후, 새로운 학기의 프로그래밍 수업에도 제안된 프로그래밍 교육 방법을 적용하여 그 효과를 검증하고자 한다.

5. 결론

‘문제해결과프로그래밍’ 교과목 수업에서 학생들은 프로그램 작성에 어려움을 겪었다. 학습자들은 파이썬 문법을 이미 습득하였고 주어진 문제의 해결 방법을 시뮬레이션하고 자연언어로 설명할 수 있었다. 하지만 그 문제해결 방법을 프로그래밍하는 것은 어려워하였다. 본 연구는 그 어려움의 원인이 단순히 코딩 능력의 부족에 있지 않고, 문제해결을 위한 알고리즘적 사고와 그것을 코드로 구현하는 능력의 부족에서 비롯된 것으로 보았다.

프로그래밍 학습은 프로그램 언어를 사용하여 문제를 해결하는 사고 방식을 배우는 과정이다. 사고(생각)는 언어를 사용하여 표현되므로, 자연언어로 표현되는 문제해결 방법을 형식언어인 컴퓨터 언어로 바꾸는 변환 방법을 배우고 익힐 수 있다면 프로그램 작성의 어려움은 해결될 것이다. 이러한 논리를 학습자들이 겪는 어려움 해결에 적용하여 다음의 단계를 거쳐 프로그래밍하도록 하였다. 먼저, 문제를 파악하기 위해 자신만의 표현법으로 문제를 재정의하였다. 재정의 문장에서는 처리해야 하는 데이터를 인지하고 프로그래밍 요소로 추출하였다. 그리고 문제해결 방법을 시뮬레이

션으로 수행하고 그 수행 과정을 종이와 연필을 사용하여 글로 작성하였다. 기록된 문장의 논리와 프로그램 요소가 알고리즘으로 전환될 때까지 문장을 수정하고 보완하였다. 이후, 완성된 알고리즘 문장을 컴퓨터 명령문으로 변환하여 프로그램을 완성하였다.

종이와 연필을 이용한 문제 재정의와 시뮬레이션 과정의 기록은 인터넷이나 다른 도움 없이 자신만의 논리를 세우는 도구가 되었다. 이 기록 과정을 통해 시뮬레이션에서 처리되는 계산과정을 인지하고 규칙을 발견하여 자신만의 논리 체계를 구축할 수 있었다. 이러한 일련의 기록 과정은 체계화되지 않은 직관적 사고를 절차화된 알고리즘적 사고로 전환하는 과정으로, '느리게 생각하기'로 이름하였다. 또한, 학기말 성적에서 본 연구의 유효성을 단편적으로 확인하였다.

향후 본 연구 내용을 프로그래밍 관련 교과에 확대 적용하고 학생들의 성적을 비교하여, 본 연구가 학생들의 프로그램 작성에 실제적인 도움이 되는지를 확인하고자 한다. 또한, 교수자가 피드백하였던 프로그래밍 학습자들의 알고리즘 문장과 프로그램을 AI가 대신할 수 있을지를 연구하여 학생들의 프로그래밍에 도움을 주고자 한다.

참고문헌

[1] Gill, S. S., Xu, M., Ottaviani, C., Patros, P., Bahsoon, R., Shaghaghi, A., & et al. (2022). AI for next generation computing: Emerging trends and future directions. *Internet of Things, 19*, 100514. https://doi.org/10.1016/j.iot.2022.100514

[2] Oedingen, M., Engelhardt, R. C., Denz, R., Hammer, M., & Konen, W. (2024). ChatGPT Code Detection: Techniques for Uncovering the Source of Code. *AI, 5*(3), 1066-1094. https://doi.org/10.3390/ai5030053

[3] Majeed, B. H., Jawad, L. F., & Alrikabi, H. T. (2022). Computational thinking (CT) among university students. *International Journal of Interactive Mobile Technologies, 16*(10), 244–252. https://doi.org/10.3991/ijim.v16i10.30043

[4] Kim, S., Yoon, S., Kim, I., & Hong, S. (2023). *Composing and operating of liberal arts education curriculum in Korean universities*. Korean National Institute for General Education.

[5] Lindoo, E. (2018). Back to the Basics in an Effort to Improve Student Retention in Intro to Programming Classes. *The Journal of Computing Sciences in Colleges, 34*(2), 72-79. https://dl.acm.org/doi/10.5555/3282588.3282599

[6] Margulieux, L. E., Morrison, B. B., & Decker, A. (2020). Reducing withdrawal and failure rates in introductory programming with subgoal labeled worked examples. *International Journal of STEM Education, 7*(1), 1-16. https://doi.org/10.1186/s40594-020-00222-7

[7] Wilson, K. (2022). Introduction to Computer Programming. *The Absolute Beginner's Guide to Python Programming.* Apress, Berkeley. https://doi.org/10.1007/978-1-4842-8716-3_1

[8] Dale, N., & Weems, C. (2013). *Programming and problem solving with C++: Comprehensive*. Jones & Bartlett Publishers.

[9] Ettles, A., Luxton-Reilly, A., Denny, P. (2018). Common logic errors made by novice programmers. In Proceedings of the 20th Australasian Computing Education Conference (ACE '18). *Association for Computing Machinery,* New York, NY, USA, 83–89. https://doi.org/10.1145/3160489.3160493

[10] Tran, D. T., & Huh, J.-H. (2023). *Exception handling. In Principles, Policies, and Applications of Kotlin Programming* (pp. 100-101). IGI Global. https://doi.org/10.4018/978-1-6684-6687-2.ch005

[11] Telecommunications Technology Association. (n.d.). *Standardization terminology dictionary.* Telecommunications Technology Association. https://terms.tta.or.kr/main.do

[12] Sebesta, R. W. (2022). *Concepts of Programming Languages (12th ed.).* Pearson.

[13] Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33-35. https://doi.org/10.1145/1118178.1118210

[14] Kim, J. K. (2018). Effect of computational thinking on problem solving process in SW education for non-CS major students. *Journal of Korea Multimedia Society, 22*(4), 472-479. https://doi.org/10.9717/kmms.2019.22.4.472

[15] Futschek, G. (2006). Algorithmic thinking: The key for understanding computer science. In R. T. *Mittermeir (Ed.), Lecture Notes in Computer Science* (Vol. 4226, pp. 159–168). Springer. https://doi.org/10.1007/11915355_15

[16] Kim, D. M., & Lee, T. W. (2020). Review of cognitive difficulties of students to learn computer programming. *Proceedings of the Korean Society of Computer Information Conference, 28*(2), 225-228.

[17] Qian, Y., & Lehman, J. (2017). Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education (TOCE), 18*(1), 1-24. https://doi.org/10.1145/3136620

[18] Yusoff, K. M., Ashaari, N. S., Wook, T. S. M. T., & Ali, N. M. (2020). Analysis on the requirements of computational thinking skills to overcome the difficulties in learning programming. *International Journal of Advanced Computer Science and Applications, 11*(3), 244–253. https://doi.org/10.14569/IJACSA.2020.0110329

[19] Lee, J., & Lee, K. (1998). The effects of metaphoric instruction on novices' learning of C language programming. *Journal of Cognitive Science, 9*(4), 75-93.

[20] Goschnick, S. (Ed.). (2018). *Innovative Methods, User-Friendly Tools, Coding, and Design Approaches in People-Oriented Programming.* IGI Global. https://doi.org/10.4018/978-1-5225-5969-6

[21] Zock, M. (1988). Natural languages are flexible tools; that's what makes them hard to explain, to learn, and to use. *Advances in Natural Language Generation: An Interdisciplinary Perspective* (pp. 181-196). London: Pinter.

[22] Omori, Y., & Araki, K. (2010). Tool support for domain analysis of the software specification in natural language. *TENCON 2010 - 2010 IEEE Region 10 Conference* (pp. 1065-1070). Fukuoka, Japan: IEEE. https://doi.org/10.1109/TENCON.2010.5686435
[23] Abbott, R. J. (1983). Program design by informal English descriptions. *Communications of the ACM, 26*(11), 882-894. https://doi.org/10.1145/182.358441
[24] Sajaniemi, J., & Kuittinen, M. (2005). An experiment on using roles of variables in teaching introductory programming. *Computer Science Education, 15*(1), 59–82. https://doi.org/10.1080/08993400500104071
[25] Hermans, F., & Aldewereld, M. (2017). Programming is writing is programming. *Science and Engineering of Programming, 1*, 1–8. https://doi.org/10.1145/3079368.3079413

권요영

· 1990년 연세대학교 전산과학과 (이학사)
· 1992년 연세대학교 대학원 전산과학과(이학석사)
· 1997년 연세대학교 대학원 컴퓨터과학과(공학박사)
· 1997년 ~ 2000년 한국전자통신연구원 선임연구원
· 2000년 ~ 현재 한국기술교육대학교 융합학과 교수

관심분야 : 고성능컴퓨팅, 임베디드 시스템, 시스템 소프트웨어, 공학교육

oykwon@koreatech.ac.kr

박은진

· 1991년 단국대학교 전자계산학과 졸업(이학사)
· 2004년 국기술교육대학교 전기전자대학원 정보통신전공(공학석사)
· 2011년 한국기술교육대학교 전기전자대학원 컴퓨터공학전공(공학박사)
· 2011년 ~ 현재 한국기술교육대학교 융합학과 겸임교수

관심분야 : 컴퓨터 그래픽스, 프로그래밍 언어교육, 공학교육

ejinpark@koreatech.ac.kr