초등 예비교사의 컴퓨터과학에 대한 인식 변화를 위한 계산적 사고 기반 알고리즘 학습의 설계 및 적용

김병수 · 김종훈†
(제주대학교)

Design and Application of Learning Algorithms based on Computational Thinking for Changes in Prospective Elementary School Teachers' Perceptions about Computer Science

Byeong-Su KIM · Jong-Hoon KIM†
(Jeju National University)

Abstract

In this study, we designed and applied the learning program of various algorithms about computer science, which were based on computational thinking, to prospective elementary school teachers who were non-majors of this field. While they were learning, they could understand two fundamental functions of computational thinking: abstraction and automation. This learning program made them change their perceptions about computer science positively. They had been interested in learning algorithms and computer science itself, and they felt confident about teaching it.

Key words : Computational thinking, Algorithms, Computer science, Human computer

Ⅰ. 서 론

현 세대의 과학과 기술은 날로 발전해나가고 있으며 이러한 배경에는 IT가 중추적 역할을 해 왔음은 누구나 인정하는 사실이다(김병수·김종훈, 2011; 김태훈·김종훈, 2012; 현동림·김은길·김종훈, 2011; 현동림·송경철·김은길·김종훈, 2011). 그럼에도 불구하고 최근 IT의 핵심이라고 할 수 있는 학문으로서 컴퓨터과학(Computer science) 분야는 침체의 길을 걷고 있다. 이러한 현상은 우리나라뿐만이 아닌 전세계적 현상으로, 이 분야의 교육자와 전공자들이 함께 풀어야 할 최대의 과제로 부각되었다. 과거 이 분야의 학자들은 '컴퓨터과학이 곧 프로그래밍'임을 강조하며 그 본질보다는 기술 습득과 훈련에 치중한 면이 많았다. 많은 세대에 걸쳐 이루어진 과거의 교육으로 인해 일반인들에게는 컴퓨터과학은 어렵고 따분하며 전공자들만을 위한 특별한 학문으로 인식되어 버렸다(고건, 2007; 김자미·이원규, 2010; Carter, 2006; Denning, 2009; Klawe & Shneiderman, 2005; Nwana, 1997).

계산적 사고(Computational thinking)는 Wing(2006)에 의해 그 개념이 소개되었다. 모든 어린이들이 읽기(Reading), 쓰기(Writing), 셈하기(Arithmetic)

† Corresponding author : 064-754-4913, jkim0858@jejunu.ac.kr

의 3R을 배우는 것처럼 21세기를 살아가는 모든 사람들에게 계산적 사고 또한 필수적으로 배워야 할 필요가 있는 사고 기술(Skill)이라고 주장하였다.

CSTA(Computer Science Teachers Association) K-12 컴퓨터과학 기준안에서는 계산적 사고를 '컴퓨터로 실행될 수 있는 방법으로 문제를 해결하고자 하는 접근이며 학문의 경계를 넘어 적용·변환·자동화 될 수 있는 문제해결의 방법론'이라고 정의하였다. 계산적 사고를 지닌 학생은 단순히 도구의 이용자가 아닌 도구 제작자로서의 능력을 가진다고 말한다(CSTA, 2011).

많은 학자들은 이러한 정의가 실제 인간의 어떠한 사고 기술을 말하는 것인지에 대한 구체적인 예가 더욱 필요함을 자각하였다(Committe for the Workshops on Computational Thinking, 2011; National Research Council, 2010). 또한 교육적인 측면에서 그 방법이나 내용은 어떠해야 하는지에 대한 논의가 필요함을 인식하였다(Kramer, 2007; Newell & Perlis, 1967).

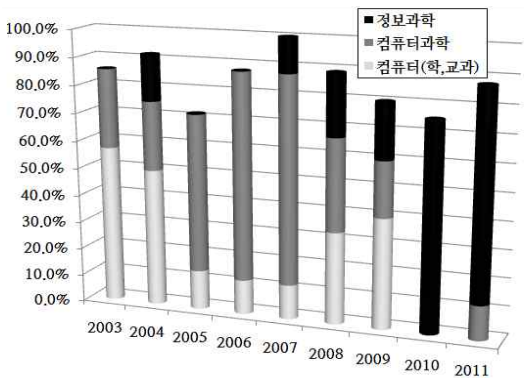
계산적 사고에 대한 광범위한 논의는 컴퓨터과학 교육의 목표·내용·방법의 재정립을 위한 중요한 시사점과 전환점을 이끌어 내고 있다는 점에서도 주목할 필요가 있다. Denning(2009)은 계산적 사고에 대한 전공자들간의 논의와 주장보다는 그들이 보여줄 수 있는 계산적 수행(Computational doing)으로 이러한 새로운 패러다임을 보급해야 할 필요성을 역설하였다.

본 연구에서는 미래의 인재들에게 이러한 사고 기술을 갖추게 하기 위한 가장 좋은 방법은 학생들을 가르칠 교사들을 먼저 가르치는 것이라고 판단하였다. 그들에게 계산적 사고의 개념을 소개하고 어떠한 사고 기술인지를 인식하게 하여 적극적으로 활용할 수 있는 능력을 갖추게 하는 것이 장기적으로 더 큰 파급력을 지닐 것으로 생각되어졌기 때문이다. 본 연구에서는 현재 예비교사들이 지니고 있는 컴퓨터과학에 대한 인식을 사전에 조사하고 계산적 사고 기반의 학습이 예비 교사들의 컴퓨터과학에 대한 인식을 긍정적으로 바뀌게 할 수 있는지에 대해 연구해 보고자 한다.

Ⅱ. 기존 연구 및 재고찰

1. 컴퓨터과학, 계산적 사고의 용어 사용

국내에서 Computer science의 용어가 어떻게 해석·번역되었는지를 2003년에서부터 2011년까지의 학위논문, 학술지논문의 제목을 통해 조사해 보았다. [그림 1]은 수집된 자료 중에 Computer science가 정보과학, 컴퓨터과학, 컴퓨터(컴퓨터학, 컴퓨터교과 포함)로 번역되어 사용된 비율을 연도별로 나타내고 있다.



[그림 1] Computer Science 용어의 해석과 사용

2010년부터 Computer science는 '정보과학'으로 해석되어 보편적으로 사용되고 있다. Computer science를'컴퓨터 또는 컴퓨터과학'이라는 번역으로 인해 이 분야의 학문적 관심 대상이 컴퓨터 머신 자체라는 오해를 없애기 위해 최근에는 '정보과학'이라는 용어를 더욱 선호하여 사용하고 있다. 이러한 추세에 맞춰 Computational thinking이라는 용어 또한 '정보과학적 사고'로 해석되어 사용되어지고 있었지만 최근 '2009 개정 교육과정에 따른 중학교 선택 교과 교육과정'의 정보교과 목표 진술에서 Computational thinking이 '계산

적 사고'로 번역되어 사용되었다(교육과학기술부, 2011). Computational thinking이 계산(Computation), 계산수행(Computing)이라는 용어의 정의에 기반했기 때문에 이를 '계산적 사고'로 해석해야 관련 학문의 내용을 맥락적으로 이해할 수 있기 때문이라고 생각된다(Denning, 2003, 2007, 2010).

이러한 연유로 앞으로 Computational thinking이 '정보과학적 사고'보다는 '계산적 사고'로 통용될 가능성이 높다고 보여진다. 본 연구에서는 Computer science에 대한 최근 '정보과학'이라는 번역보다는 앞서 설명했듯이 컴퓨터의 의미를 확장시켜 '컴퓨터과학'이라는 용어를 사용하고자 한다. 이는 용어 자체가 가지는 계산의 의미를 더욱 부각시키기 위한 것이며 원어인 Computer science와의 의미차도 가장 좁게 줄일 수 있기 때문이다. 본 논문의 이하에서는 Computer science는 '컴퓨터과학'으로, Computational thinking은 '계산적 사고'로 번역하여 사용하고자 한다.

2. 계산적 사고의 핵심기능: 추상화, 자동화

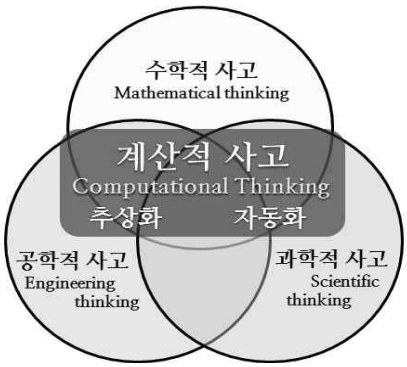
계산적 사고는 일종의 분석적 사고이다. 문제를 해결하기 위한 일반적인 접근에서는 수학적 사고를, 실세계의 제한안에서 크고 복잡한 시스템을 설계하고 평가하는 문제들에 대한 접근에서는 공학적 사고를, 인간의 행동과 감정, 지능, 계산 가능성을 이해하기 위한 문제의 접근에서는 과학적 사고를 공유한다(Wing, 2008).

계산적 사고는 인간이 가지고 있는 하나의 '사고 능력'이라고 보는 측면보다는 하나의 '사고 기술' 또는 '사고의 패러다임'이라고 해석하는 것이 올바르다고 판단된다(Kramer, 2007).

계산적 사고는 추상화(abstraction)와 자동화(automation)라는 2가지 핵심적인 기능을 가지고 있다. 추상화는 인간의 사고과정에서나 타 학문에서 자주 사용되는 개념이기도 하지만 계산적 사고의 추상화는 레이어를 사용한다는 개념에서 차이가 있다. 이는 복잡한 시스템을 설계하고 개발하는 과정에서 큰 힘을 발휘한다. 예를 들어 이전의 레이어 또는 다음 레이어와의 관계를 정의해 나가는 추상화 작업을 통해 시스템은 개발되며 사용자가 활용할 수 있는 레이어 단계까지 단순화된다.

자동화란 하나의 계산을 어떤 컴퓨터로 표현하는 것이 적절한지를 판단하고 선택하여 해당 컴퓨터와 대화가 가능한 언어 또는 계산 모델을 통해 계산을 알맞은 형식으로 표현하여 자동적으로 처리할 수 있게 하는 과정과 그 수행을 말한다.

계산적 사고의 자동화는 컴퓨터와 매우 밀접한 관계에 있다. 본 논문에서는 계산의 주체가 기계일 경우는 디지털 컴퓨터(Digital computer), 인간인 경우를 휴먼 컴퓨터(Human computer)로 표현하고자 한다.



[그림 2] 계산적 사고의 특징과 핵심 기능

[그림 3] 두가지 측면의 '컴퓨터'용어의 정의

알고리즘 학습의 측면에 있어서 본 연구에서 주안점을 두고 있는 부분은 학습자 또는 인간을 휴먼 컴퓨터로 바라보고 자신의 생각을 추상화하고 스스로 자동화 기능을 수행하거나 다른 이에게 자동화 기능을 수행할 수 있도록 하는 계산적 사고 기술을 습득하게 하는 것이다.

Ⅲ. 알고리즘 학습의 설계 및 적용

1. 계산적 사고를 위한 알고리즘 학습

이전까지의 컴퓨터과학 교육은 프로그래밍 언어 교육에만 치중되어져 왔다고 비판되어지고 있다(김병수·김종훈, 2012; 김정아·김병수·이지훤·김종훈, 2011). 이는 계산적 사고에서의 컴퓨터가 컴퓨터 머신으로만 초점이 맞추어졌기 때문이었다고 말할 수 있겠다. 하지만 컴퓨터과학 학문의 비전공자나 교양을 위한 학습으로는 프로그래밍 언어 자체에 대한 학습시간이 너무 많이 소비되어 많은 문제점을 낳았다. 또한 계산적 사고를 위한 학습이라면 굳이 컴퓨터 머신만을 자동화의 도구로 선택하여 사용할 필요는 없어 보인다.

본 논문에서는 추상화의 방법으로 '정보과학 알고리즘'에서 학습 내용을 추출하고 이에 대한 자동화 도구로는 휴먼 컴퓨터에 초점을 맞추었다. 학습자가 계산적 사고의 추상화와 자동화의 개념과 사고 기술을 정확히 인식하고 익혀 주도적이고 의식적으로 이를 사용할 수 있게 하는 것이 본 연구에서 개발하고자 하는 학습의 목표이다. 본 연구의 알고리즘 학습의 내용은 <표 1>과 같으며 각 학습 주제에 적절한 계산모델을 적용하고자 하였다. 대표적으로 활용된 계산모델과 알고리즘 문제에 대해서는 다음에서 설명하고자 한다.

<표 1> 알고리즘 학습 프로그램의 내용과 순서

구분	학 습 주 제	도입개념
1	1부터 100까지의 합 구하기	반복문
2	수열의 n번째 값 구하기	변수
3	구구단 나타내기	중첩반복
4	튜링기계 / 유한 오토마타	조건분기
5	로봇 프로그래밍	인공지능
6	소수 구하기	Brute-Force
7	Tromino 퍼즐	분할정복
8	샐러리맨의 비용계산	욕심쟁이
9	로봇 생쥐 실험	동적계획
10	다양한 알고리즘 문제 해결	

2. 계산모델 및 적용

알고리즘이란 어떠한 문제를 해결하기 위해 정의된 유한한 계산의 절차를 의미한다.

컴퓨터과학에서의 알고리즘 학습은 계산적 사고를 학습하기에 가장 알맞은 요소라고 할 수 있다. 그 이유는 알고리즘 학습을 통해 계산적 사고의 추상화, 자동화의 개념을 정확히 인지할 수 있고 학습 대상에 따라 학습 난이도를 쉽게 조절할 수 있으며 컴퓨터 머신을 사용한 자동화뿐 아니라 학습자가 직접 휴먼 컴퓨터의 역할을 하여 자동화를 경험할 수 있는 기회를 가질 수 있기 때문이다.

가. 튜링기계

알고리즘 학습 및 계산 원리에 대한 학습은 어떠한 컴퓨터에 상관없이 계산 모델을 통해 가능하다는 의견은 김형철(2011)의 연구에서도 논리적으로 제시되었다. 계산 모델이란 컴퓨팅 시스템에 대한 수학적 추상체를 말하는 것이며, 계산에서 사용될 수 있는 연산들의 집합과 각 연산의 비용을 정의하여 실제 문제해결에 사용된다. 이 연구에서는 이러한 계산 모델의 예로 튜링기계, 재귀함수, 람다계산, 유한상태 기계, 유한상태 오토마타를 들었다. 본 논문에서는 이 중 튜링기계가 다양한 학습자를 대상으로 하기에 적당하며 문제의 난이도를 쉽게 조절할 수 있기 때문에 알고리즘 학습을 하기에 좋은 계산 모델이라 판단하였다. 이를 응용하여 본 연구에서는 다음과 같은 학습문제를 제작하였다. 이를 통해 계산적 사

고의 추상화와 자동화가 정확히 어떤 개념인지 알아보고자 한다. 튜링기계 알고리즘 학습에 대한 예는 <표 2>와 같다.

<표 2> 튜링기계 알고리즘 학습의 문제 예시

튜링기계 학습의 배경 조건
어떤 수학자가 두루마리 종이, 연필과 지우개, 명령표를 가지고 있다. 두루마리 종이는 무한히 길며 한 칸씩 나뉘어졌다. 각 한 칸을 셀(Cell)이라 한다. 수학자는 명령표의 명령에 따라 동작을 시작/종료할 수 있고, 좌/우로 한 칸 이동하기 또는 제자리에 머무르기를 할 수 있다.
두루마리 각 셀의 기호
두루마리에는 각 셀에는 0, 1, *(빈칸)중 하나의 기호가 쓰여져 있다. 수학자는 이 두루마리의 각 셀에 0, 1, *(빈칸)만 기록할 수 있다. 이미 셀에 쓰여져 있는 기호는 지우개로 지워서 *(빈칸)을 만든다거나 다른 기호로 쓰기가 가능하다.
문제 예시
위의 조건들을 바탕으로 했을 때, 수학자가 왼쪽으로 이동하며 *(빈칸)이 나올 때까지 기호 0은 1로, 기호 1은 0으로 바꾸고 처음 위치로 돌아오는 명령표를 만들어보자. 단 시작위치는 *(빈칸)이 아니며 시작위치의 오른쪽에는 *(빈칸)이 있다고 가정한다.
문제 예시 그림
 ※ ?(물음표)는 어떠한 입력기호가 등장할지 모른다는 의미이다.

<표 2>에 대한 해답으로 다양한 알고리즘이 나올 수 있다. <표 3>이 그 한 가지 예라고 할 수 있다.

위의 학습을 통해 학습자는 계산적 사고의 추상화와 자동화의 과정을 모두 거쳐야만 한다. 예를 들어 학습자가 <표 2>에서 제시된 문제에서 일어날 수 있는 모든 가능한 경우들을 고려하며 <표 3>과 같은 명령표를 작성하는 과정 자체와 결과물을 추상화라고 할 수 있다.

그리고 이 명령표를 스스로 실행해보거나 다른 학습자에게 넘겨주어 실행해보게 할 수 있다. 이것이 자동화이다. 자동화 과정에서는 <표 4>와

<표 3> 추상화 단계: 튜링기계 알고리즘 문제의 해

정답 예시

<명령표>

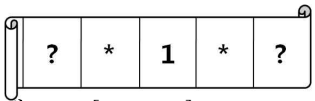
명령 번호	현재 상태	입력 기호	출력 기호	수학자 이동	다음 상태
①	시작	0	1	왼쪽	시작
②	시작	1	0	왼쪽	시작
③	시작	*	*	오른쪽	리턴
④	리턴	0	0	오른쪽	리턴
⑤	리턴	1	1	오른쪽	리턴
⑥	리턴	*	*	왼쪽	끝(멈춤)

- 시작상태: 수학자의 첫 상태가 시작상태이다. 기호 0은 1로, 기호 1은 0으로 바꾸면서 계속 왼쪽으로 이동한다. 만약 *(빈칸)이 나오면 오른쪽으로 한 칸 이동하며 수학자의 다음 상태는 리턴으로 바뀐다.
- 리턴상태: 기호 0, 1을 모두 그대로 두고 *(빈칸)이 나올 때까지 오른쪽으로 이동한다. 만약 *(빈칸)이 나오면 이는 처음 위치의 오른쪽에 있었던 *(빈칸)이므로 왼쪽으로 한 칸 이동하며 끝 상태가 되며 동작을 멈춘다.

같이 실제 많은 사례수의 문제들을 명령표에 따라 작동해 보아야 한다.

<표 4> 자동화 단계: 문제 사례에 대한 작동

사례 1 : [* 1 *]



1. 시작상태 : [* 1 *]
2. 명령번호 ② : [* 0 *]
3. 명령번호 ③ : [* 0 *]
4. 명령번호 ④ : [* 0 *]
5. 명령번호 ⑥ : [* 0 *]

사례 2 : [* 1 0 *]

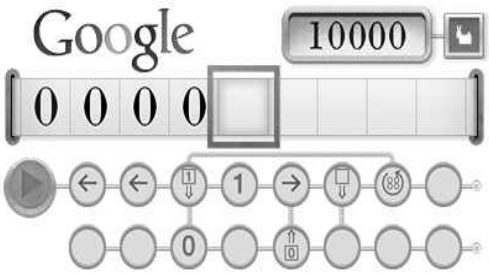
1. 시작상태 : [* 1 0 *]
2. 명령번호 ① : [* 1 1 *]
3. 명령번호 ② : [* 0 1 *]
4. 명령번호 ③ : [* 0 1 *]
5. 명령번호 ④ : [* 0 1 *]
6. 명령번호 ⑤ : [* 0 1 *]
7. 명령번호 ⑥ : [* 0 1 *]

사례 3 : [* 1 1 0 0 *]

1. 시작상태 : [* 1 1 0 0 *]
2. 명령번호 ① : [* 1 1 0 1 *]
3. 명령번호 ① : [* 1 1 1 1 *]

4. 명령번호 ② : [* 1 0 1 1 *]
5. 명령번호 ② : [* 0 0 1 1 *]
6. 명령번호 ③ : [* 0 0 1 1 *]
7. 명령번호 ④ : [* 0 0 1 1 *]
8. 명령번호 ④ : [* 0 0 1 1 *]
9. 명령번호 ⑤ : [* 0 0 1 1 *]
10. 명령번호 ⑤ : [* 0 0 1 1 *]
11. 명령번호 ⑥ : [* 0 0 1 1 *]

모든 문제 사례에 대해 명령표가 올바르게 작성되었는지를 파악하는 것은 불가능하다. 하지만 오작동 될 수 있는 문제 사례를 찾아봄으로써 학습자가 만들어 놓은 알고리즘의 오류 유무를 알아낼 수 있다. 튜링 기계 알고리즘 학습 문제의 난이도를 조절하는 방법은 튜링 기계의 기본 조건은 그대로 둔채 문제만 수정하는 방법과 다른 형태의 튜링 기계를 정의하는 방법으로 가능하다.



[그림 4] 튜링 탄생 100주년 기념 구글 두들

[그림 4]의 튜링 기계는 <표 2>에서 설명한 튜링기계와 조금 다른 형태를 가지지만 계산적 사고를 필요로 하는 학습으로 활용될 수 있다. 본 연구에서는 앞서 설명한 다양한 튜링기계의 작동 원리를 초등 예비교사들에게 설명하고 다양한 문제를 통해 계산적 사고를 익히게 하였다. 그리고 <표 5>와 같은 문제를 제시하였다. 이 문제에서 헤더를 수학자로, 테이블은 명령표와 같은 기능이다.

이 문제에 대한 학습자의 해답은 매우 다양했으며 효율적인 알고리즘들을 찾아볼 수 있었다.

<표 5> 난이도가 높은 튜링기계 알고리즘 문제

문제
문자 A, B, *(공백)으로만 구성된 임의의 길이의 문자열(문자열이란 의미는 공백없이 모두 붙어 있음을 의미)에 대해, 헤더가 왼쪽으로 이동하며 *(공백)기호를 만날 때까지 같은 문자가 2개 이상 연속되어 출현하면 이를 모두 S로, 그렇지 않은 문자는 P로 대체시키는 튜링기계의 테이블을 정의하시오. (단, 초기 위치의 오른쪽 셀은 *이다.)
예시그림
예를 들자면, 아래의 초기 문자열을

*	A	A	B	A	B	B	*

시작시 헤더위치 ↑

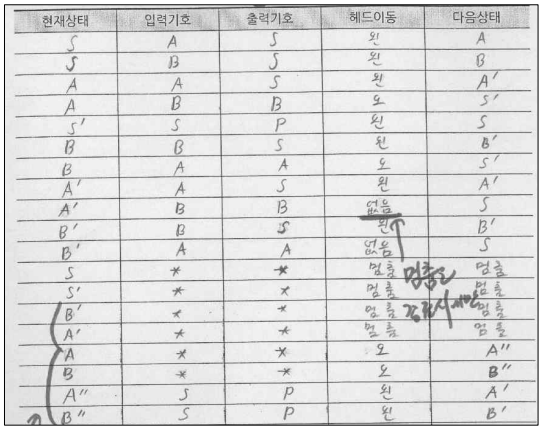
다음과 같이 바꾸는 것이 목적이다.

*	S	S	P	P	S	S	*

↑ 종료시 헤더위치

[그림 5] 알고리즘 1 (양○○ 학생)

[그림 5]의 알고리즘 1의 기본 아이디어는 처음 위치에서 왼쪽으로 이동하며 같은 문자가 두 개 연속 나올 경우에 오른쪽으로 한 칸 이동하여 S로 바꾸고, 다시 왼쪽으로 한 칸 이동하여 S로 바꾸게 한다는 것이다.



[그림 6] 알고리즘 2 (조○○ 학생)

[그림 6]의 알고리즘 2는 알고리즘 1과는 다른 접근 방법을 취한다. 이 방법은 처음부터 기호를 S로 바꾸면서 왼쪽으로 이동하게 된다. 다음 왼쪽 셀의 기호가 이전 셀의 기호와 다르면 오른쪽으로 한 칸 이동하여 S를 P로 바꾸고 제자리로 돌아온다는 것이 기본 아이디어이다. 알고리즘 1에서는 25개의 명령줄이 필요했다면 알고리즘 2는 단 19개의 명령줄로 문제를 해결하여 상대적으로 더 효율적이라고 말할 수 있다. 이렇게 같은 문제를 가지고도 자신만의 알고리즘을 창의적으로 만들어내고 계산 모델에 적절한 언어로 표현하고 휴먼 컴퓨터로서 이러한 명령들을 실행하고 오류를 찾아내는 과정 자체에서 계산적 사고 기술을 습득할 수 있게 되는 것이다. 더욱 고무적인 것은 많은 학생들이 이러한 학습이 기존의 컴퓨터 수업에서는 해보지 못한 '흥미로운 활동'이라는 반응을 보인 것이다.

나. 순서도와 순서도 인터프리터 Raptor

순서도를 활용한 알고리즘 학습은 알고리즘 시각화와 관련된 연구로 이미 많은 기존 연구에서 그 효과성은 검증되었다(오경숙, 2009; 이은혜, 2007). 먼저 학습자에게 <표 6>과 같이 순서도에서 활용되는 기호의 종류와 그 기능을 알려주고 <표 7>과 같이 예시를 제시하여 익히게 한다.

<표 6> 순서도 기호의 종류와 기능

기 호	이 름	기 능
	단 말	순서도의 시작과 끝
	처 리	처리 작업
	조건/판단	조건명시
	출 력	화면 또는 서류 출력
	흐름선	처리흐름과 기호연결
	반 복	조건에 따라 내용 반복

<표 7> 반복문의 예시

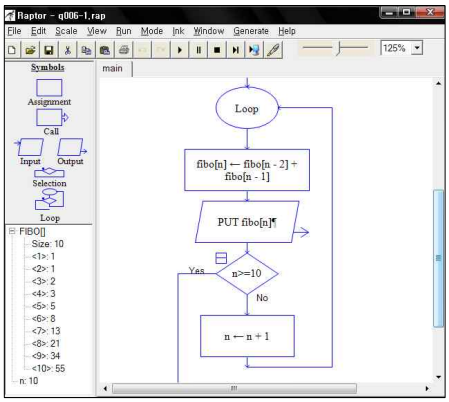
예 시	출력 (왼쪽부터)
LOOP N←1, 5, 1 N	(N의 초기값은 1이며, 5까지 1씩 증가하며 기호 안의 내용을 반복하게 됨) 1 2 3 4 5
LOOP N←1, 5, 1 N*2	2 4 6 8 10

이와 같은 학습 이후에는 <표 8>과 같은 문제들을 제시하여 계산적 사고를 이끌어 낼 수 있다.

<표 8> 순서도 문제의 추상화와 자동화

문 제 (가)	문 제 (나)
START LOOP A←0, 1, 1 LOOP B←0, 1, 1 A+B STOP	START S←0 LOOP A←1, 5, 1 LOOP B←1, 2, 1 LOOP C←1, 2, 1 S←S+1 S STOP

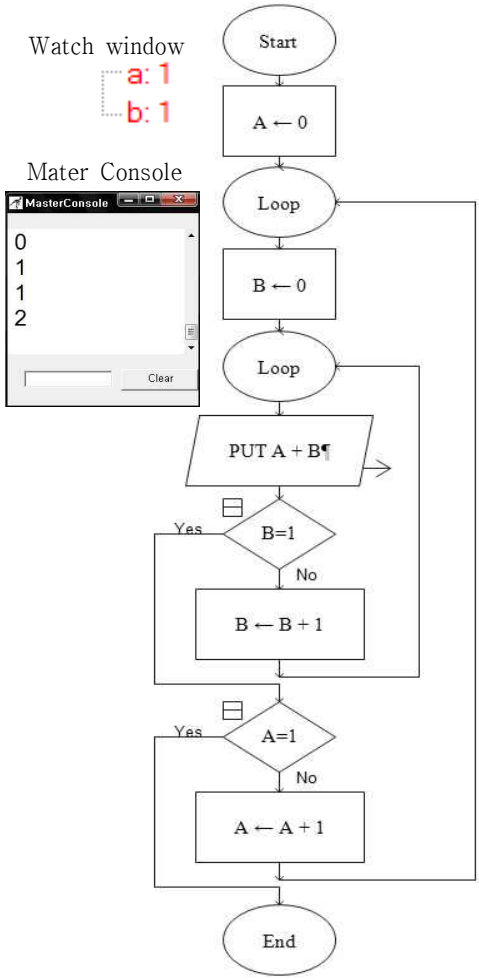
<표 8>과 같은 순서도에서 출력을 예상하는 문제는 추상화된 표현을 이해하고 휴먼 컴퓨터로서의 자동화의 기능을 수행할 수 있는지를 요구하는 유형이다. 추상화를 강조하기 위한 문제라면 출력을 먼저 제시하고 순서도 전체를 작성하게 하거나 부분적으로 비어있는 순서도를 그려넣는 유형이 좋다. Raptor는 학생들이 자신의 알고리즘을 순서도를 통해 표현하고 실행해볼 수 있게 하는 프로그래밍 환경을 제공하는 순서도 인터프리터(Flowcharts interpreter)이다(Carlisle, Wilson, Humphries & Hadfield, 2005; Wilson, Carlisle, Jumphries & Moore, 2012). Raptor는 기본적으로 제공되는 6개의 심볼을 이용하여 순서도를 직접 만들고 이를 실행해볼 수 있다.



[그림 7] Raptor 실행 화면

실행 절차를 눈으로 확인할 수 있으며 데이터들의 변화와 출력값을 얻을 수 있다는 것이 가장 큰 장점이다. 비전공자들을 위한 프로그래밍 교육에 Raptor가 활용된 기존 연구에서 학습자들은 자신들의 알고리즘을 만드는데에 효과적으로 이를 활용하였다고 보고되었다(Carlisle, Wilson, Humphries & Hadfield, 2004).

예를 들어 <표 8>의 문제 (가)를 Raptor에서는 [그림 8]과 같이 만들고 실행할 수 있다. 이는 자동화 도구가 디지털 컴퓨터인 경우로 휴먼 컴퓨터의 자동화와 비교하며 자신의 오류를 찾아보는 디버깅 작업이 편리해진다는 이점이 있다.

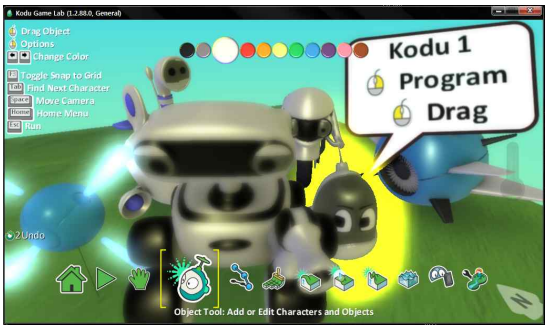


[그림 8] <표 8> 문제 (가)의 Raptor 순서도

다. 타일을 이용한 프로그래밍: 코듀(KODU)

Kodu는 마이크로소프트사에서 개발한 Xbox용 게임 디자인 툴로 2009년 CES(Consumer Electronics Association)에서 처음 발표되었고 2010년에는 PC버젼이 배포되었다(Microsoft research, 2012; Moran, 2011). 실세계와 흡사한 물리적 환경이 제공되는 3D 환경, 타일을 이용한 손쉬운 프로그래밍 환경, 로봇과 인공지능, 병렬프로그래밍의 개념이 적용된 가상환경을 제공한다는 장점이 있다(Fowler & Cusack, 2011; MacLaurin, 2011;

Stolee & Fristoe, 2011). 초등학교 저학년 학생에서부터 어른까지 누구나 프로그래밍을 통해 게임, 시뮬레이션, 스토리텔링 작품을 만들 수 있다는 것이 최대의 장점이다.



[그림 9] Kodu의 로봇 객체들

Kodu의 프로그래밍은 [그림 10]과 같은 타일 조합 형식이라 초보자에게 매우 쉬운 편이다.

[그림 10] 타일 형식의 프로그래밍

[그림 10]의 명령을 텍스트형 명령문으로 바꾸고 이를 해석해본다면 <표 9>와 같다.

<표 9> Kodu의 명령문과 해석

Kodu의 명령문: Kode
① **When** see apple, **Do** move toward
② **When** bump apple, **Do** eat it
해 석
① 사과가 보이면, 그쪽으로 움직여라.
② 사과에 부딪히면, 그것을 먹어라.

Kodu의 명령 코드(Code)를 Kode라고 부르는데, Kode의 각 행은 크게 When 구문과 Do 구문으로 나뉜다. When은 상황인식과 관련되며 기존 프로그래밍에서의 IF문 또는 While문의 기능을 하는 구문이다. Do 구문은 객체가 실제 행하게 되는 동작들에 대한 명령이다. Kodu는 각기 다른 센서(see, bump, hear, got 등)에 대해 동작을 병렬처리 한다. 하지만 복수개 명령문의 When 구문에서 같은 센서를 이용하면서 그 조건 또한 참(True)일 경우는 명령문의 번호가 낮은 것부터 먼저 실행된다.

[그림 11] 타일 형식의 프로그래밍

예를 들어 [그림 11]의 명령문인 경우는 ①번 명령만 실행된다고 생각하면 된다. 또한, 조건문의 확장이나 실행동작의 확장은 [그림 12]와 같이 들여쓰기 또는 페이지 삽입/이동을 통해 가능하다.

[그림 12] 들여쓰기 및 페이지 이동(switch)

Kodu의 기본적인 문법은 앞서 설명한 튜링기계나 순서도의 논리와 일치한다고 할 수 있다. 다만 병렬처리에 대한 개념을 좀 더 이해할 필요가 있다는 것이 차이점이다. 이러한 학습을 통해 계산적 사고를 향상하기 위한 문제는 다음의 <표 10>과 같이 제시될 수 있다.

<표 10>의 문제는 실행활에서 활용되는 청소로봇을 만들어보자는 의미이다. 이러한 문제를

학습자에게 제시하였을 경우 병렬처리, 센서에 대한 개념을 정확히 이해해야 정답을 찾을 수 있었다. 이를 정확히 이해하지 못한 학습자들은 공통적으로 <표 11>의 일반적인 오답 유형을 작성하여 제출하였다.

<표 10> Kodu 문법을 활용한 알고리즘 문제

문 제
로봇에게 "쓰레기를 주워 휴지통에 버리게 하기" 동작을 프로그래밍 하려고 한다. 보기에 있는 모든 단어를 한 번씩만 사용하여 명령문을 완성해라. (로봇은 한 번에 하나의 쓰레기만 주을 수 있다.)
보 기
버리기 소유하기 보기 움직이기 움직이기 부딪히기 부딪히기 줍기 쓰레기 쓰레기 쓰레기 쓰레기쪽으로 쓰레기를 쓰레기를 휴지통쪽으로 휴지통

<표 11> <표 10>의 일반적인 오답 유형과 정답

일반적인 오답 유형
1 WHEN 보기 쓰레기 DO 움직이기 쓰레기쪽으로 → 2 WHEN 부딪히기 쓰레기 DO 줍기 쓰레기를 3 WHEN 소유하기 쓰레기 DO 움직이기 휴지통쪽으로 → 4 WHEN 부딪히기 휴지통 DO 버리기 쓰레기를
정 답
1 WHEN 소유하기 쓰레기 DO 움직이기 휴지통쪽으로 → 2 WHEN 부딪히기 휴지통 DO 버리기 쓰레기를 3 WHEN 보기 쓰레기 DO 움직이기 쓰레기쪽으로 → 4 WHEN 부딪히기 쓰레기 DO 줍기 쓰레기를

학습자는 이러한 학습을 통해 하나 하나의 단어가 로봇의 센서, 동작기(모터)와 연결되는 점, 코드의 조합에 따라 전혀 다른 작동이 일어난다는 점을 통해 추상화의 개념을 이해할 수 있었다. 자신이 만든 프로젝트(World)를 실행해봄으로써 자신이 계획했던 문제 해결 방법대로 자동화가 되는지를 확인해 볼 수 있었다.

3. 추상화의 다양한 접근

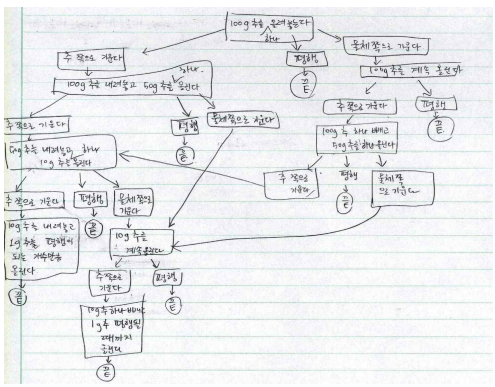
알고리즘의 학습 방법에는 앞서 설명한 계산 모델을 활용하는 방법 이외에도 다양한 방법들이 존재한다. 퍼즐, 퀴즈 등의 Unplugged/Pencil-and-Paper 학습 방법도 좋은 알고리즘 학습 활동이라 할 수 있다. 하지만 이러한 학습 방법에서 중요한 것은 정답을 구하는 것이 아니라 무엇을 추상화해야하며 자동화를 위해 어떻게 표현할 수 있는지에 대해 고민해야 한다는 것이다. 예를 들어 <표 12>와 같은 문제에 대해 학습자는 추상화, 자동화의 개념을 적극적으로 활용할 수 있어야 할 것이다.

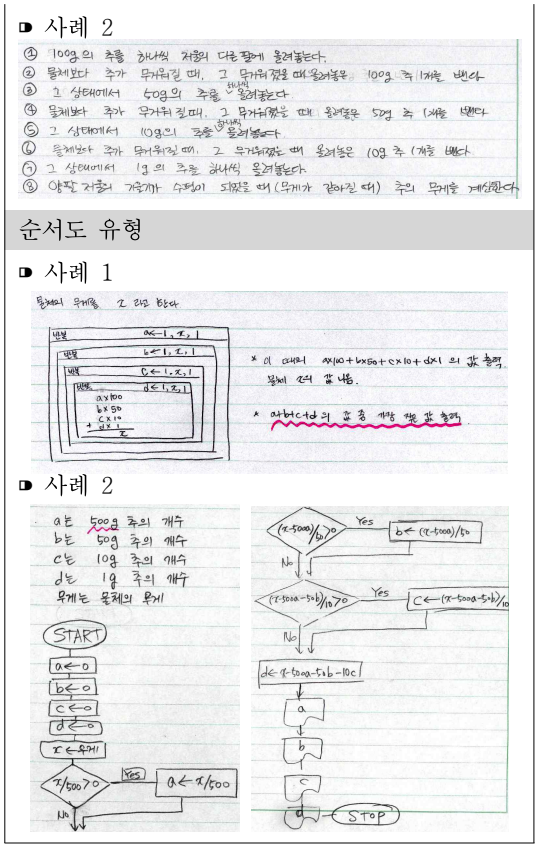
<표 12> 알고리즘 학습 문제

문 제
과학시간에 저울을 이용해서 물체의 무게를 재려고 합니다. 추는 100g, 50g, 10g, 1g의 종류가 있으며 추의 개수는 무게를 재는 데에 부족함이 없습니다. 저울과 추를 이용하여 물체의 무게를 재는 알고리즘을 만들어 보세요. ※ 표현방법은 그림, 글, 수식, 순서도, 의사코드(Pseudo-code) 등 모두 가능합니다.

<표 12>의 문제를 학습자들에게 제시하자 다양한 해답이 나왔다. <표 13>은 이 문제에 대한 다양한 추상화를 이용한 문제해결 접근 방법이다.

<표 13> 알고리즘 추상화 사례

유한 오토마톤 유형
◗ 사례 1 



학습자들의 해답에서 볼 수 있듯이 계산적 사고를 활용한다는 의미는 자신의 아이디어를 계산 모델에 알맞은 언어로 추상화하고 이를 디지털 컴퓨터 또는 휴먼 컴퓨터가 자동화할 수 있도록 표현하는 능력을 갖추는 것을 말한다.

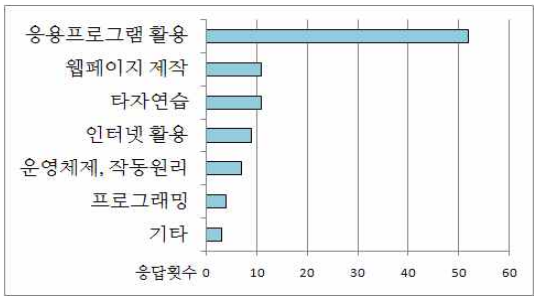
Ⅳ. 연구 방법 및 결과

본 연구에서 개발한 학습 프로그램을 초등 예비교사인 교육대학교 학부생 2학년 69명을 대상으로 실시하였다. 학습 주제와 내용은 앞서 설명한 바와 같이 <표 1>과 같으며 한 학기(2012년 3월~7월)동안 2시간 연차시 수업으로 총 15회 실시하였다.

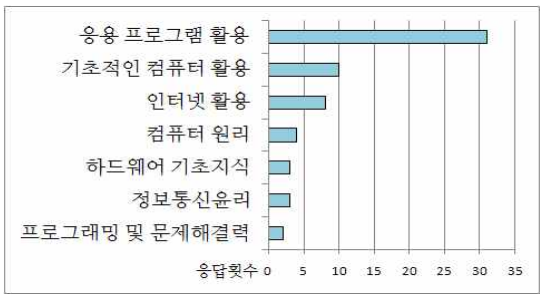
본 연구에서는 학습 전에 걸쳐 이들에게 초·중등교육에서 컴퓨터과학 교과수업에서 학습했던 내용, 컴퓨터 수업에 대한 흥미도, 초등 교육과정에서의 컴퓨터과학 교육에 대한 자신감 및 계산적 사고에 대한 사전지식에 대해 설문하였다. 학습 후에는 컴퓨터과학 학습에 대한 흥미도, 자신감의 변화, 계산적 사고에 대한 이해, 알고리즘 학습의 필요성에 대해 설문하였다.

1. 사전 설문

초·중등교육에서 컴퓨터과학 교육으로 받았던 학습들을 기억나는 대로 모두 쓰게 하였다. 또 초·중등교육과정에서 학생들이 컴퓨터과학 교육에서 배워야 할 내용들이 무엇인지 쓰게 하였다.



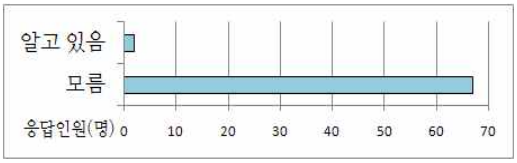
[그림 13] 컴퓨터과학 학습 경험



[그림 14] 컴퓨터과학 교육에서 학생들이 배워야 한다고 생각하는 학습 내용들

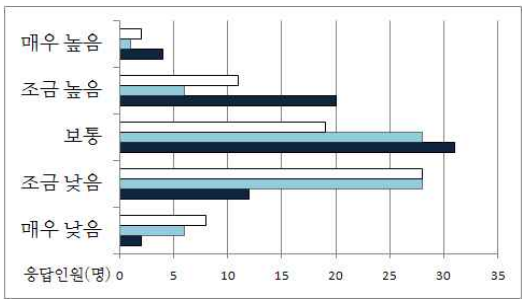
예비 초등교사들은 초·중등교육에서 응용프로그램(워드프로세서, 스프레드시트 등) 활용 학습을 주로 기억하고 있으며 또한 이러한 학습 활동이 다음 세대에도 계속되어져야 한다고 생각하고 있었다. 그리고 계산적 사고에 대해 알고 있는지

에 대해서도 조사하였다.



[그림 15] 계산적 사고에 대한 사전 지식

대부분의 초등 예비교사들은 계산적 사고에 대한 학습을 알지 못했다. 이들에게 평소 컴퓨터과학 수업에 대한 흥미도, 자신의 컴퓨터 활용 능력과 컴퓨터과학을 가르치는 것에 대한 자신감에 대해서도 설문하였다.

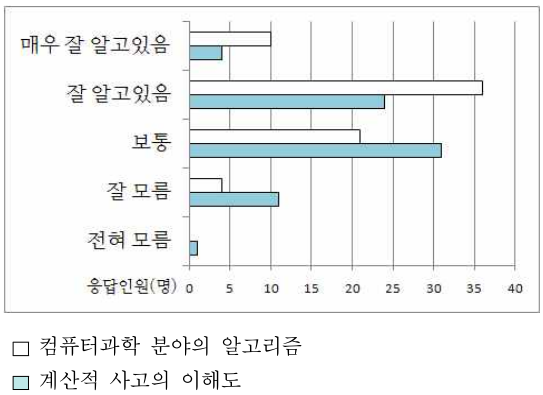


[그림 16] 흥미도, 활용능력, 교수 자신감 비교

[그림 16]의 비교 그래프에서 보듯이, 초등 예비교사들의 컴퓨터과학 수업에 대한 흥미도는 높은 편이다. 하지만 자신의 컴퓨터 활용 능력과 교수에 대한 자신감은 낮은 편이다. 이러한 상반된 결과의 원인은 초등 예비교사들이 컴퓨터과학 교육의 주요 학습 활동이 응용프로그램의 활용이라고 생각하고 있는 상황에서 자신의 활용 능력은 낮은 편이라고 생각하기 때문에 교수에 대한 자신감 또한 낮은 편으로 나타나고 있다고 판단된다.

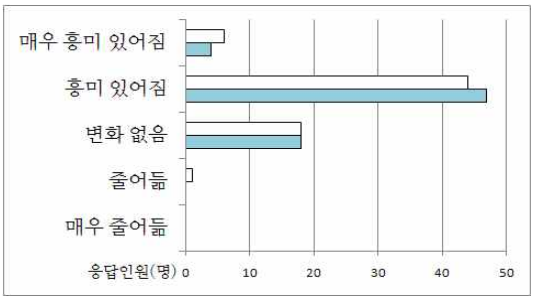
2. 사후 설문

한 학기동안 2시간 연차시의 수업을 15회(총 30차시 수업) 실시한 후 초등 예비교사들의 계산적 사고 및 컴퓨터과학의 다양한 알고리즘에 대한 이해도, 컴퓨터과학과 알고리즘 학습의 흥미도, 교수 자신감 변화 정도 등과 관련하여 사후 설문을 실시하였다.



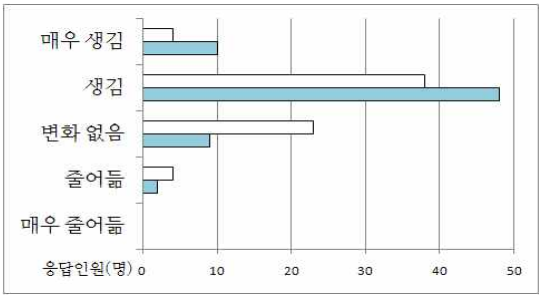
[그림 17] 계산적 사고와 알고리즘에 대한 이해도

[그림 17]의 설문 결과에서도 보듯이 많은 초등 예비교사들은 계산적 사고의 개념과 컴퓨터과학에서 활용되는 다양한 알고리즘에 대해 이해하게 되었다. 또한 이들에게 계산적 사고의 개념 이해가 상대적으로 더 어려워했음을 알 수 있었다.



[그림 18] 알고리즘 학습과 계산적 사고에 대한 흥미도 설문 결과

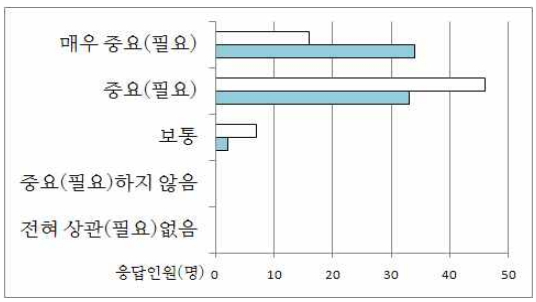
[그림 18]은 초등 예비교사들이 자가 진단한 계산적 사고와 알고리즘 교수·학습에 대한 흥미도의 변화에 관한 설문 결과이다.



[그림 19] 컴퓨터과학의 흥미도, 교수 자신감 관련 설문 결과

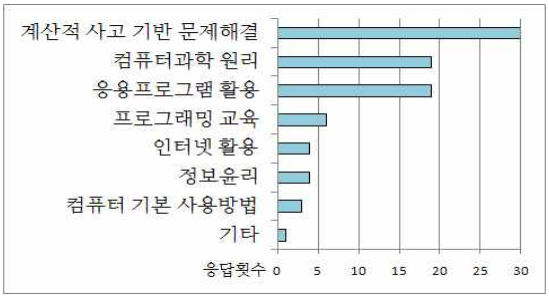
[그림 19]는 본 연구의 학습 프로그램을 이수한 초등 예비교사들의 84%가 컴퓨터과학 자체에 대한 흥미가 생겼다. 또, 전체 인원 중 60%가 컴퓨터과학 교수에 대한 자신감이 생겼음을 보여주고 있다.

[그림 20]은 대부분의 초등 예비교사들은 컴퓨터과학 교육을 위한 계산적 사고를 기반으로 하는 알고리즘 학습이 필요하다고 생각하게 되었으며, 이를 위해 교사에게도 논리적 사고력이 매우 중요하다고 생각하게 되었다. 이러한 전체적인 변화로 초등 예비교사들은 컴퓨터과학 교육에서



[그림 20] 논리적 사고력, 알고리즘 교육의 중요(필요)성

학생들이 배워야 한다고 생각하는 학습 내용들에 대한 생각이 바뀌었다.



[그림 21] 컴퓨터과학 교육에서 학생들이 배워야 한다고 생각하는 학습 내용들

[그림 21]에서 보듯이 동형의 사전 설문인 [그림 14]와 비교하였을 때, 초등 예비교사들은 계산적 사고 기반의 문제해결, 컴퓨터과학 원리, 프로그래밍 교육이 컴퓨터과학 교육에서 필요함을 새롭게 느끼게 된 것이다.

Ⅴ. 결론 및 제언

본 연구에서는 컴퓨터과학의 비전공자인 초등 예비교사들을 대상으로 하는 계산적 사고 기반의 알고리즘 학습 프로그램을 설계하고 적용하였다. 이는 미래의 세대들을 책임질 교사들이 계산적 사고의 개념을 이해하지 못하고 이에 대한 교육의 필요성을 느끼지 못한다면 컴퓨터과학 교육뿐 아니라 미래의 인재를 육성할 교육에 대한 비전을 찾지 못할 것이라는 판단 때문이었다.

본 학습 프로그램의 설계에서는 컴퓨터과학의 다양한 알고리즘을 주제로 선정하고 이에 대한 학습 방법에 있어서는 계산적 사고의 추상화, 자동화의 개념을 충분히 이해시키고자 계산 모델과 휴먼 컴퓨터, 디지털 컴퓨터를 충분히 활용하고자 했다.

이러한 학습 프로그램을 투입한 결과, 많은 초등 예비교사들이 계산적 사고와 알고리즘 학습에 흥미가 모두 높아졌고 컴퓨터과학 자체에 대한 흥미도와 교수에 대한 자신감까지 높아졌다. 이는 초등 예비교사들이 생각하고 있는 컴퓨터과학

교육이 소프트웨어 활용 교육이 아닌 원리와 논리 중심의 문제해결력을 신장하기 위한 교육임을 새롭게 인지하게 된 것이다. 또한, 이러한 교육을 위해서 교사 스스로가 알고리즘 학습과 논리적 사고력을 신장시킬 필요성이 있음을 인지하는 수준까지 끌어올렸다는 점이 더욱 고무적이다.

21세기의 모든 이들이 갖추어야 할 사고 기술이 있다면 그 교육에 대한 몫은 교사일 것이다. 그런 이유로 새로운 패러다임에 대한 교육은 교사에게 먼저 시행되어야 할 필요가 있다. 교사를 양성하고 교사 연수를 담당하는 많은 교육 기관에서는 이에 대한 노력을 더욱 해야 할 것이며, 컴퓨터과학 교육을 전공하는 많은 이들은 학습자의 수준에 맞는 실질적인 학습 내용을 개발하는 데에 많은 노력을 해야 할 것이라고 본다.

참고 문헌

고 건(2007). 컴퓨터공학의 위기 극복 방안, 정보과학회지 25(8), 72~75.

교육과학기술부.(2011). 2009 개정 교육과정에 따른 중학교 선택 교과 교육과정[별책 18], 교육과학기술부 고시 제2011-361호.

김병수・김종훈(2011). 기하학 문제해결 능력 향상을 위한 STEAM 교육 기반의 종이 접기 애플리케이션 설계 및 개발, 한국지식정보기술학회 6(6), 103~110.

김병수・김종훈(2012). 한국정보올림피아드 초등부 경시부문 문제해결을 통한 알고리즘 교재 개발 및 적용. 정보교육학회논문지 16(1), 11~20.

김자미·이원규(2010). 교과교육의 측면에서 본 정보교과의 정체성에 대한 고찰, 정보교육학회논문지 14(2), 219~227.

김정아・김병수・이지훤・김종훈(2011). 융합형 인재 양성을 위한 IT 기반 STEAM 교수·학습 방안 연구. 수산해양교육연구 23(3), 445~460.

김태훈·김종훈(2012). 물리학습을 위한 STEAM 기반의 안드로이드 앱 개발, 수산해양교육연구 24(1), 25~33.

김형철(2011). 컴퓨터과학 교육용 계산 원리 학습 도구의 기능요소 고찰, 제주대학교 교육대학원. 석사 학위논문.

오경숙(2009). 순서도를 활용한 알고리즘 교육 시스템 설계 및 구현, 순천대학교 대학원 컴퓨터과학과, 석사학위논문.

이은혜(2007). 순환학습모형과 추상화단계를 적용한 알고리즘 시각화 교수학습방법, 고려대학교 교육대학원 컴퓨터교육전공, 석사학위논문.

현동림·김은길·짐종훈(2011). 안드로이드 기반 사고 공유 마인드맵 애플리케이션 구현, 수산해양교육연구 23(2), 234~243

현동림·송경철·김은길·김종훈(2011). 안드로이드 기반 비만 관리 애플리케이션 개발 - BMI 및 운동량 산출을 중심으로, 수산해양교육연구 23(4), 568~581.

Carlisle, M. C., Wilson, T. A., Humphries, J. W., & Hadfield, S. M.(2004). RAPTOR: Introducing progamming to non-majors with flowcharts. Journal of Computing Sciences in colleges, 19(4), 52~60.

Carlisle, M. C., Wilson, T. A., Humphries, J. W., & Hadfield, S. M.(2005). RAPTOR: a visual programming environment for teaching algorithmic problem solving. Proceedings of the 36th SIGCSE technical symposium, 37(1), 176~180.

Carter, L.(2006). Why students with an apparent aptitude for computer science don't choose to major in computer science. Proceedings of the 37th SIGCSE technical symposium on Computer science education, 38(1), 27~31.

Committe for the Workshops on Computational Thinking. (2011). Report of a workshop of pedagogical aspects of computational thinking, Washington DC: National Academies Press.

Denning, P. J.(2003). Great principles of computing, Communications of the ACM, 46(11), 15~20.

Denning, P. J.(2007). Computing is a natural science, Communications of the ACM, 50(7), 13~18.

Denning, P. J.(2009). The profession of IT: Byeond computational thinking, Communications of the ACM, 52(6), 28~30.

Denning, P. J.(2010). Ubiquity sumposium 'What is computation?': Computing and computation, Ubiquity 2010.

Fowler, A., & Cusack, B.(2011). Kodu game lab: improving the motivation for learning programming concepts. Proceedings of the 6th International Conference on Foundations of Digital Games, 238~240.

Klawe, M., & Shneiderman B.(2005). Crisis and opportunity in computer science. Communications of the ACM, 48(11), 27~28.

Kramer, J.(2007). Is abstraction the key to computing?. Communications of the ACM, 50(4). 36~42.

MacLaurin, M. B.(2011). The design of kodu: a tiny visual programming language for children on the Xbox 360. Proceedings of the 38th annual ACM SIGPLAN-SIGACT symposium, 241~246.

Microsoft research(2012). Kodu. http:// research.microsoft.com /en-us/projects/kodu

Moran, C.(2011). Microsoft's Kodu takes kids from game players to game makers, http://redmond.patch.com/articles/microsofts-kodu-takes-kids-from-game-players-to-game-makers-2

National Research Council(2010). Report of a workshop on the scope and nature of computational thinking, Washington DC: The National Academies Press.

Newell, A., Perlis, A. J., & Simon, H. A.(1967). What is Computer Science?. Science, 157, 1373~1374.

Nwana, H. S.(1997). Is computer science education in crisis?. ACM Computing Surveys, 29(4), 322~324.

Stolee, K. T., & Fristoe, T.(2011). Expressing computer science concepts through Kodu game lab. Proceedings of the 42nd ACM technical symposium on Computer science education, 99~104.

The CSTA Standars Task force(2011). CSTA K-12 Computer Science Standards, http://csta.acm.org/Curriculum/sub/CurrFiles /CSTA_ K-12_CSS.pdf

Wilson, T., Carlisle M. C., Jumphries J., & Moore, J.(2012). Welcome to the RAPTOR homepage. http://raptor.martincarlisle.com

Wing, J. M.(2006). Computational thinking. Communications of the ACM, 49(3). 33~35.

Wing, J. M.(2008). Computational thinking and thinking about computing. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 366(1881), 3717~3725.

• 논문접수일 : 2012년 07월 07일
• 심사완료일 : 1차 - 2012년 07월 21일
• 게재확정일 : 2012년 07월 29일