전산 비전공 학부생들을 위한 프로그래밍 교수 전략

김준우*, 주지영**, 임광혁***

요약

컴퓨터 및 정보 기술에 대한 기본적인 이해는 현대인에게 필수적인 소양으로 인식되고 있다. 이에 따라 전산 비전공 대학생들도 프로그래밍과 같은 컴퓨터 관련 과목을 수강하는 경우가 많다. 그러나, 일반적으로 프로그래밍 강의가 문법 소개 위주로 진행되는 경우가 많아 수강 후에도 간단한 프로그램조차 작성하지 못하는 학생들이 많다. 초심자인 학생들은 흔히 반복문과 같은 흐름 제어 구문을 이해하고 응용하는 부분에서 어려움을 겪기 시작하며, 이는 과목에 대한 흥미를 떨어뜨리는 요인이 된다. 본 논문은 기본적인 흐름 제어 단계에서 학생들이 접하는 예제 소스 코드들에 숨겨져 있는 기본적인 원리들을 제시하고, 이를 이용한 프로그래밍 교수 및 학습 콘텐츠 설계 전략을 제안한다. 이러한 전략은 학습자들의 문제 해결 능력 및 논리적 사고력을 증진하는데 도움이 될 것으로 기대된다.

Instructional Strategy for Teaching Computer Programming to Non-Computer Science Major Undergraduates

Jun-Woo Kim*, Jee-Young Joo**, Kwang-Hyuk Im***

ABSTRACT

Understanding computer and information technology is very important in today's society. In this context, many non-computer science major undergraduates take computer programming language course in their school. However, elementary programming language classes generally concentrate on introducing syntax, and even writing a simple program is very difficult to many undergraduates who have completed such classes. Non-computer science majors and novice learners often have trouble in understanding and applying the flow of control such as iteration, which is a major barriers to learning programming. This paper identifies the basic principles hidden in the example source codes related with basic flow of control and proposes a strategy for programming language instruction and learning contents design. This strategy is expected to help the learners to develop problem-solving and logical thinking capabilities.

Key Words : Programming Language, Instructional Strategy, Learning Contents, Flow control, Iterative Statements

* 동아대학교 산업경영공학과(✉kjunwoo@dau.ac.kr)
** 한국과학기술원 과학영재교육원
*** 배재대학교 전자상거래학과
· 제1저자(First Author) : 김준우 · 교신저자(Correspondent Author) : 임광혁
· 접수일(2011년 2월 25일), 수정일(1차 : 2011년 3월 25일), 게재확정일(2011년 3월 31일)

Ⅰ. 서 론

정보화 시대를 맞아 일상생활의 여러 분야에서 컴퓨터 및 정보통신 기술의 활용은 이제 필수적인 것이 되어 가고 있다[1][2]. 이에 따라 일반 기업 사무환경에서도 여러 가지 컴퓨터 활용 능력이 중요시되고 있고, 많은 대학에서 학부생들이 기본적인 프로그래밍 또는 컴퓨터를 활용한 사무 능력 배양과 관련된 교과목을 수강한다. 나아가, 학과에 따라 관련 자격증 취득을 졸업 요건으로 명시하기도 한다[1][3]. 또한 이러한 경향은 경영학 관련 학과들처럼 전산 전공이 아닌 학과들에서도 나타나고 있다. 따라서 전공을 막론하고 학부생들이 프로그래밍이나 사무 자동화와 같은 컴퓨터 활용 능력을 배양하게 하는 것이 중요함을 알 수 있다.

컴퓨터 활용 능력 중에서도 프로그래밍은 학습자의 복합적인 문제해결능력을 요구하는 고차원적인 활동이다[4]. 하지만 초중고 과정에서 컴퓨터 활용 능력의 학습은 주로 H/W, S/W의 기능적인 조작이나 사용법에 초점을 맞추는 경우가 많고[4][5], 대학 신입생 수준의 학습자들 중 상당수가 대학에서 프로그래밍을 처음 접하면서 학습에 어려움을 겪는다. 또한 이러한 프로그래밍 초심자들은 프로그래밍은 까다롭고 이해하기 어려운 것이라는 막연한 선입견을 갖기도 한다[6]. 더구나 전산 비전공 학부생들의 경우, 프로그래밍 능력이 자신의 진로와 큰 상관이 없는 역량으로 여기는 경향이 있으며, 실제 프로그래밍 과목 수강 중 내용을 이해하는데 어려움을 겪다보니 과목 수강 후에도 간단한 프로그램조차 스스로 작성하지 못하는 경우도 많다.

프로그래밍을 학습하는 이유에 대해 살펴보면, 프로그래밍 학습은 일차적으로는 간단한 프로그램 개발 및 구현 능력 습득을 목표로 하나, 궁극적으로는 창의적인 문제해결력 및 논리적 사고력을 배양하는 데 초점이 맞추어져야 함이 지적되고 있다[1][4][5][7][8]. 특히 전산 비전공 대학생들을 대상으로 하는 프로그래밍 수업에서는 이러한 부가적인 목표가 중요하게 고려되어야 할 것이다. 하지만 기초 프로그래밍 교재나 수업들이 종종 특정 프로그래밍 언어의 문법 소개 위주로 구성되다보니 이러한 목표가 충분히 반영되지 않는 경우가 많고, 문법 위주 학습만으로 학습자가 실제 개발 능력을 습득하기는 현실적으로 어렵다[9].

물론 문법 소개와 함께 관련 예제 소스 코드와 그에 대한 설명들이 함께 제공되지만, 초심자들은 보통 이러한 소스 코드를 해석해 보기 급급하고, 이에 성공하더라도 예제와 동일한 기능을 직접 구현하도록 할 경우, 시작조차 하지 못하는 모습이 많이 관찰된다. 특히 반복문이나 재귀문, 프로그램 구조화 등 프로그램 흐름 제어가 필요한 부분에서 이러한 경향이 두드러진다[10][11]. 따라서 개별 문법 및 구문 하나하나보다 전체적으로 명령들이 어떻게 구성되어 프로그램의 기능을 수행하게 되는지를 파악하고 이해하는 것이 중요하다[10][12].

본 논문은 경영학 관련 전공 대학생들을 대상으로 기초 프로그래밍 과목을 강의한 경험을 토대로 전산 비전공 학부생들이 프로그래밍 과목을 접할 때 어떤 문제점이 있는지를 살펴보고, 이러한 학습자들을 위한 효과적인 교수전략을 개발하고자 한다. 구체적으로는 초심자들이 어려움을 많이 겪는 반복문 및 이후 개념들을 설명하기 위하여 기초 문법 학습 직후에 학습자들이 숙지해야 할 일반적인 프로그래밍 원리들을 제시하고, 이를 이용한 단계적인 프로그래밍 학습 모형을 제안한다. 이러한 내용들은 천안 소재 H대학 산업경영학부에서 비주얼베이직 언어 강의에 실제로 적용되었고, 기타 프로그래밍 언어 학습에도 도움이 될 것으로 기대된다.

Ⅱ. 관련 연구

프로그래밍을 시작하기 위해서는 먼저 특정 프로그

래밍 언어의 기본적인 문법을 익혀야 한다. 이로 인해 기초 프로그래밍 과정의 학습은 종종 특정 언어의 문법들을 소개하고 암기하는데 중점을 두게 된다. 하지만 문법 위주 학습은 초심자들의 흥미를 떨어뜨리고, 실제 개발 능력 습득으로 이어지기 어렵다[6][13]. 따라서 Ben-ari[6]가 지적하였듯이, 컴퓨터 프로그래밍 교육에서도 학습자들이 프로그래밍 방법에 대한 효과적인 내적 모델(mental model)을 습득하는 것이 중요하다. 즉, 개별 문법이나 구문 하나하나의 작성보다 전체적으로 명령들을 어떻게 구성하는지를 이해하는 것이 중요하다[10][12].

초심자들의 경우, 프로그래밍 학습 초기에 변수, 대입, 연산자 등 기본 문법과 관련된 오개념을 갖기도 하는 것으로 알려져 있다[6][14][15][16]. 하지만 본 논문의 저자들이 대학 학부생들을 관찰한 바로는 기초 문법 부분은 반복 숙달과 간단한 예제 연습만으로도 학습자들의 이해를 높일 수 있었다. 반면 기존의 연구들에서도 지적되었듯이, 초심자들은 조건문, 반복문, 재귀문 등 흐름 제어가 필요한 구문들을 이해하고 응용하는 부분에서 보다 큰 어려움을 겪는 경우가 많다[9][10][11][17]. 일반적으로 다양한 예제들을 접하면서 프로그래밍에 대한 이해를 높이려는 시도들도 존재했지만[18][19], 기본적으로는 반복문을 전후하여 나오는 흐름 제어 기능을 구현하는 과제에 직면하였을 경우, 어떻게 이 문제에 접근하여 해결해나갈 것인지에 대한 일반적인 전략을 학습자들에게 주지시켜 나가는 것이 중요할 것으로 보인다.

기존의 흐름 제어 프로그래밍 학습 관련 연구들은 일반적으로 재귀문 학습에 초점을 맞춘 경우가 많았다. Sinha and Vessey[11]는 그 이유를 재귀문이 일상생활에서의 사람의 작업 방법과 상이한 부분이 있어 초심자들이 이해하기 어렵기 때문이라고 설명한다. 재귀문 학습을 돕기 위한 기존 연구들에서는 반복문과 재귀문의 관계에 착안하여, 먼저 반복문을 익힌 다음 반복문을 활용하여 문제에 접근하고, 반복문으로 작성된 소스 코드를 재귀문으로 수정하는 과정 등을 통해 학습 효과를 높일 수 있다는 점이 제안되었다[10][20]. 이러한 연구들은 보통 반복문을 학습자가 이미 습득했음을 전제하지만, 저자들의 경험으로는 전산 비전공 학부생들의 경우, 반복문 및 그 전후에 나오는 배열이나 프로시저 등의 개념 역시 문법만 암기할 뿐, 어떤 경우에 왜 이러한 요소들이 사용되는지에 대해 깊이 이해하지 못하는 경우가 많다. 이러한 맥락에서 본 논문은 초심자들이 반복문 및 전후 문법들을 이해하는 것을 돕기 위한 교수 전략을 제안하고자 한다.

반복문을 토대로 재귀문을 학습하는 것처럼 기존 학습 내용과 새로운 내용의 연관성을 통해 지식을 확장해나가는 것은 일반적으로 효과적인 교수 전략으로 알려져 있다[21]. Merril[22] 역시 학습 효과 극대화를 위해서는 실제 문제 상황을 학습자가 직접 해결하는 기회를 부여하고, 이 때 기존에 학습자가 가진 지식과 새로운 지식이 결합되어 그 문제를 해결하는 상황을 제시하는 것이 필요하다는 점을 지적하였다.

프로그래밍 과목에서는 일반적으로 학습자들이 직접 프로그램을 작성해보거나, 실기 평가를 실시하는 경우가 많다. 하지만 새로운 문법이나 구문들이 기존의 문법과 어떤 관계가 있는지에 대한 일관된 설명이 부족하기 쉽다. 따라서 이러한 점을 보완하기 위하여 상대적으로 습득이 용이한 기초 문법에서 어떻게 흐름 제어 구문들이 도출되는지를 학습할 필요가 있다고 생각되며, 본 논문에서는 특히 반복문 및 그 전후 문법들과 기초 문법 간의 관련성에 대해 탐구하고, 이를 바탕으로 흐름 제어 구문 학습과 관련된 교수 전략 및 학습 콘텐츠 설계 방법을 제안하고자 한다.

III. 기초 문법 학습 직후의 프로그래밍 원리

프로그래밍을 처음 배우는 학습자들은 대부분 초기에 특정 프로그래밍 언어에 대한 개괄적인 안내에 이

어 기본적인 문법을 익힌다. 이 과정에서 “Hello, World!"로 대표되는 간단한 입출력 프로그램을 작성해보기도 하며, 이후에는 변수 개념 및 선언 방법, 값의 대입이나 활용에 이어 여러 연산자 등을 학습한다.

그러나 이후 조건문, 반복문 단계에서 어려움을 겪기 쉽고, 특히 반복문의 경우 문법 자체는 이해 또는 암기를 하더라도 구체적인 프로그램 작성 시 이들을 응용하지 못하는 경우가 많다. 또한, 반복문의 이해가 부족하면 이후의 내용들도 습득하기 어려워져, 학기 초에 비교적 의욕적으로 프로그래밍 학습을 시작하던 수강생들도 반복문 부분에서 이해를 포기하는 경우도 종종 관찰된다.

사실 기초 문법 이후에 나오는 반복문, 배열, 프로시저 등의 개념 활용에는 모두 비슷한 프로그래밍 원리들이 내재되어 있다고 볼 수 있으며, 이러한 원리들을 습득하는 것이 기초 문법 이후 내용을 보다 효과적으로 이해하는데 도움이 될 것으로 생각된다. 이에 따라 본 논문은 기초 문법 직후에 프로그래밍 초심자들에게 다음의 두 가지 원리를 숙지시킬 것을 제안한다.

3.1 작업 분할의 원리

첫 번째는 작업 분할의 원리이다. 이는 컴퓨터 프로그래밍에서는 복잡한 작업이나 개수가 많은 데이터들에 대한 작업을 단번에 처리하지 않고, 대부분 작은 단위 또는 데이터 1개씩에 대한 작업으로 분할하여 수행하도록 접근해야 한다는 것이다. 이는 초창기 문법 학습 이후의 내용이 담고 있는 중요한 것임에도 불구하고 프로그래밍 교육에서 강조되지 못하고 있다.

예를 들어 1에서 5까지의 정수 5개의 합을 변수 sum 에 저장하는 것을 생각해보자. 초기 문법 교육을 마친 학습자들은 이를 위해 다음과 같은 구문을 작성하기 쉽다.

$$sum = 1 + 2 + 3 + 4 + 5 \quad (1)$$

위 구문은 1~5의 숫자들을 모두 더한 합을 변수 sum 에 저장한다는 점에서 기능적으로는 목표에 부합하며, 실생활에서 우리가 쓰는 표현과도 유사하다. (1)의 방법은 5개의 숫자를 한꺼번에 모두 처리한다는 특징이 있다. 비유를 하자면, 1~5의 숫자 합을 구하기 위하여 그림 1의 (a)처럼 5개 숫자값을 한꺼번에 변수 sum 에 넣고 있는 것이다. 하지만 프로그래밍에서는 이런 식의 처리가 드물다. 대신 한 번에 한 개의 값을 처리하는 식으로 분할하여 접근하는 것이 먼저 필요한데, 그림 1의 (b)에서 이러한 원리를 볼 수 있다.

즉, sum 이 0인 상태부터 시작하고, 첫 번째 값으로 가서 sum 에 1을 더한다. 그 다음에는 두 번째 값으로 가서 1이 저장된 현재의 sum 에 2를 더하여 sum 의 값이 3이 된다. 마찬가지로 이후에는 현재 3이 저장된 sum 에 다음 숫자인 3을 더하여 총 6이 되고, 이런 식으로 현재까지의 sum 에 매번 다음 수 1개를 더하는 작업을 수행해나가면 최종적으로는 sum 에 1~5의 총합이 저장된다는 식이다. 그림 1의 (b)와 같은 방법을 실제 구문으로 작성하면 아래와 같아진다.

$$sum = 0 \quad (2.1)$$

$$sum = sum + 1 \quad (2.2)$$

$$sum = sum + 2 \quad (2.3)$$

$$sum = sum + 3 \quad (2.4)$$

$$sum = sum + 4 \quad (2.5)$$

$$sum = sum + 5 \quad (2.6)$$

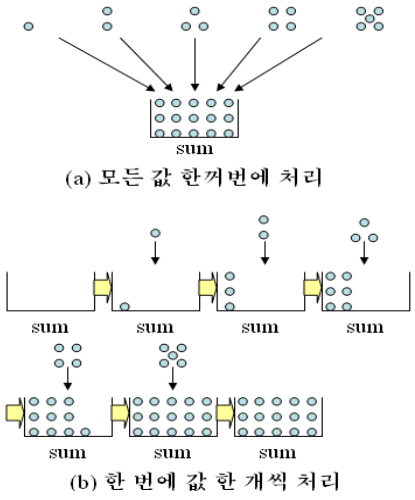

그림 1. 작업 분할의 원리
Fig. 1. The Principle of Task Division

대학 학부생 수준의 학습자들은 그림 1의 (b)와 같은 접근을 해도 최종적으로 변수 sum 에 원하던 결과가 저장된다는 것은 쉽게 받아들이는 편이다. 따라서 기본적인 문법 학습을 마친 학습자들은 (2)와 같은 구문도 무난히 이해하고 작성한다.

3.2 중복 작업 제거의 원리

위 (2)의 구문은 아직 실용적이지 못하다. 그 이유는 (2.2)~(2.6)에서 비슷한 작업이 여러 줄에 걸쳐 나오기 때문이다. 단순하게 생각해도 (2)는 (1)과 기능은 같지만 소스 코드의 길이가 길어 오히려 더 비효율적으로 보인다.

여기서 학습자들이 두 번째로 알아야 할 프로그래밍 원리는 중복 작업의 제거이다. 이는 비슷한 작업 내지 똑같은 작업을 하는 구문들을 일일이 여러 번 쓰는 것을 지양해야 한다는 것으로 역시 프로그래밍 학습에 있어 대단히 중요하나 초심자들은 이를 충분히 숙지하지 못하는 경우가 많다.

비슷한 소스 코드를 여러 번 작성하면 초기 개발 시 타이핑 양이 늘어나고 오타 등의 에러를 발생시킬 가능성도 커지며, 향후 프로그램 유지보수도 어렵다. 반복문은 바로 이러한 중복 작업 제거를 사용하는 방법 중 하나로 볼 수 있다.

위 (2.2)~(2.6)에 나오는 다섯 줄의 구문을 다시 살펴보면, 이들은 완전히 똑같은 구문들은 아니지만 그 형태나 기능이 대단히 비슷하며, 수학적인 표기법을 빌리자면 아래와 같이 일반화시켜 표현할 수 있다.

```
i = 1, 2, 3, 4, 5인 동안 각 i에 대하여
  sum = sum + i                                    (3)
```

대학 신입생들의 경우, 일반적으로 대학수학 등의 과목을 통해 위와 같은 표기법에 익숙한 경우가 많으므로, (2)를 (3)과 같은 형태로 나타낼 수 있다는 점에도 쉽게 동의한다. 그리고 (3)과 같은 형태를 표현하는 방법이 반복문으로, 이를 이용하면 (4)의 소스 코드를 얻게 된다.

```
For i = 1 To 5
  sum = sum + i
Next i                                             (4)
```

경험적으로 반복문의 문법만을 가르치고 (4)와 같은 소스 코드를 바로 제시하는 경우, 학생들이 이를 이해하는데 어려움이 많았다. 나아가, (4)의 작업이 1~5의 총합을 구하는 것이라는 것을 이해하더라도 반복문의 의미를 깊이 깨닫지 못하고, 이후 다른 예제에서 반복문을 응용하지 못하는 경우도 많다. 하지만 작업 분할 및 중복 작업 제거 원리를 설명하고, 위 (1)~(4)의 과정을 통해 설명했을 때는 학습자들이 반복문의 개념과 필요성을 이해하기가 더 용이하다.

3.3 프로그래밍 원리들의 필요성

여기서 학습자들은 왜 (4)와 같은 소스 코드가 (1), (2)에 비해 바람직한지를 알 필요가 있다.

먼저, (4)와 같은 소스 코드는 작성 및 관리가 용이하다. (1)의 경우, 만약, 1~5의 합이 아니라 1~100의 합을 구하는 것이 목적이었다면 이 소스 코드는 한 줄이 대단히 길어질 것이다. 설사 이를 모두 타이핑하는 데 성공했다고 하더라도 교수자가 다시 이 프로그램을 101~200의 합을 구하는 것으로 수정하라고 할 경우, 수정 작업이 만만치 않다. 이러한 점은 (2)의 경우도 마찬가지이다. 하지만 (4)와 같이 구현할 경우, 반복문의 카운터 변수 i 의 시작값과 종료값만 조정하면 이러한 수정을 쉽게 할 수 있다.

두 번째 장점은 프로그램 상에서 더해야 할 수들의 시작값과 종료값이 명확히 정해져 있지 않은 경우에 대한 처리이다. 만약 사용자가 두 개의 정수 a, b(a < b)를 입력하면 a~b 범위 정수들의 총합을 구하는 경우를 생각해보자. 소스 코드 단계에서는 a, b 값이 정해져 있지 않은 상태이므로, (1)과 같이 접근하면 아래처럼 중간 부분의 식을 완성할 수 없다.

$$sum = a + (a + 1) + (a + 2) + ... + b \quad (5)$$

이 경우에는 반드시 (4)와 같은 형태로 작성하고, 카운터 변수 i 의 시작값을 a, 종료값을 b 로 지정해주어야 구현이 가능하다.

IV. 반복문 이외 구문들과의 연계

이번에는 반복문을 전후하여 소개되는 다른 문법 요소들은 앞 장에서 제시한 프로그래밍 원리와 어떻게 연관지을 수 있는지 살펴보고자 한다.

4.1 배열

중복 작업 제거가 필요한 이유 중의 하나는 프로그래밍에서 작업 분할 원리를 적용할 경우, 비슷하거나 동일한 작업이 여러 번 나오는 경우가 많기 때문이다. 기초 프로그래밍 수준의 과목에서 배우는 문법들 중, 배열이나 프로시저 등의 개념 역시 반복 제거 원리와 연계시켜 설명할 수 있다. 예를 들어, 앞 장에서 1~5의 총합을 sum 에 저장한 것과 달리, 이번에는 정수값을 저장한 변수 a, b, c, d, e 의 값들을 총합하여 sum 에 대입하는 경우를 생각해 보자. (1)과 같은 식으로 접근하면, 기초 문법 지식만으로도 아래와 같은 구현이 가능하다.

$$sum = a + b + c + d + e \quad (6)$$

작업 분할의 원리를 적용하면, 5개 변수를 한꺼번에 더하지 않고, 한 번에 하나씩만 더하도록 아래와 같은 소스 코드를 얻게 된다.

$$sum = 0 \quad (7.1)$$

$$sum = sum + a \quad (7.2)$$

$$sum = sum + b \quad (7.3)$$

$$sum = sum + c \quad (7.4)$$

$$sum = sum + d \quad (7.5)$$

$$sum = sum + e \quad (7.6)$$

그 다음에는 중복 제거 원리에 따라 비슷한 작업들인 (7.2)~(7.6)을 일반화시켜 나타내는 것이 필요하지만, 이번에는 (3)과 같이 표현하는 것이 불가능하다. (2.2)~(2.6)의 다섯줄들은 형태가 모두 같고, 단지 마지막 숫자값만 달라지지만, (7.2)~(7.6)의 경우에는 그렇지 않기 때문이다. 따라서 카운터 변수와 같은 개념을 이용하는 반복문으로 나타내기 어렵다.

배열은 동등한 의미를 갖는 여러 변수들을 묶어 한

개의 단위로 다루는 것이다. 위에서 a, b, c, d, e 들은 모두 총합을 계산하는데 사용된다는 동등한 의미가 있으므로, 이처럼 개별적인 변수로 관리하기보다 원소가 5개인 배열 a 로 관리하기로 하자. 그러면 이제 이 문제는 a(0), a(1), a(2), a(3), a(4)의 총합을 구하는 것이 되고, (7)은 아래처럼 수정된다.

```
sum = 0                                  (8.1)

sum = sum + a(0)                         (8.2)

sum = sum + a(1)                         (8.3)

sum = sum + a(2)                         (8.4)

sum = sum + a(3)                         (8.5)

sum = sum + a(4)                         (8.6)
```

(7.2)~(7.6)과 달리, (8.2)~(8.6)은 이제 (3)과 같이 일반화된 표현을 아래처럼 쉽게 얻을 수 있다.

```
i = 0, 1, 2, 3, 4 인 동안 각 i 에 대하여
  sum = sum + a(i)                       (9)
```

이를 반복문 처리하면 최종적으로 아래와 같다.

```
For i = 0 To 4
  sum = sum + a(i)
Next i                                   (10)
```

이러한 예제에서 보듯이 배열 역시 기본적인 개념은 여러 변수들을 한데 관리하는 것이지만, 배열을 사용하는 이유에는 작업 분할 및 중복 작업 제거의 원리들이 내재되어 있음을 알 수 있다.

4.2 프로시저

앞의 예제들 외에도 복잡하거나 많은 개수의 값을 다루는 작업의 경우에는 대부분 작업 분할과 중복 작업 제거의 원리들을 적용하여 단계적으로 구현 방법을 설명하는 것이 가능하다.

이번에는 정수 변수 a, b, c, d 중에서 가장 값이 큰 것을 골라 변수 max 에 저장하는 예제를 생각해보자. 실생활에서 구체적인 숫자 네 개 중 가장 큰 것을 고르는 작업은 어렵지 않다. 하지만 이러한 프로그래밍 과제를 부과했을 때, 초심자들은 종종 시작조차 하지 못하는 경우가 많다. 이 때 앞의 예제들처럼 먼저 조건문을 포함한 이전 문법 내용들로 기능을 어느 정도 작성해본 다음, 작업 분할과 중복 작업 제거 원리를 적용해 나가도록 유도하면 보다 효과적인 학습이 가능하다.

제일 먼저 네 변수들 중 가장 큰 것을 한 번에 골라내는 것을 생각해보자. 이는 조건문으로 해결할 수 있지만, 초심자들이 이조차도 구현하지 못하는 경우에는 좀 더 단순화된 기능을 구현하도록 유도할 수 있다. 즉, 네 변수가 아니라 두 변수 a, b 중 큰 것을 고르는 기능은 아래와 같이 쉽게 구현이 가능하다.

```
If a >= b Then
  max = a
Else
  max = b
End If                                   (11)
```

즉, a, b 둘 중, a 가 더 크면 max 에 a를, 그렇지 않다면 b를 저장하면 된다. 이렇게 단순화된 과제를 제시하는 것은 두 가지 측면에서 장점이 있는데, 첫 번째로는 전체 기능보다 학습자들이 접근하기 쉽다는 것이고, 두 번째로는 이렇게 단순화된 기능을 반복적으로 수행하여 결국 최초의 복잡한 기능이 달성되는 경우도 많다는 점이다. (11)의 소스 코드를 확장하면, a, b, c, d 중 가장 큰 것을 max 에 대입하는 기능은 최초에

아래처럼 구현할 수 있다.

```
If a>=b and a>=c and a>=d Then
  max = a
ElseIf b>=a and b>=c and b>=d Then
  max = b
ElseIf c>=a and c>=b and c>=d Then
  max = c
ElseIf d>=a and d>=b and d>=c Then
  max = d
End If                                    (12)
```

(12)의 소스 코드는 그림 1의 (a)처럼 모든 값을 한꺼번에 처리하는 경우이다. 하지만 이 역시 비교해야 할 값이 4개가 아니라 10개, 100개와 같이 늘어난다면 소스 코드가 대단히 길어지고 비효율적이다. 이 예제도 먼저 작업 분할의 원리에서부터 접근할 수 있다.

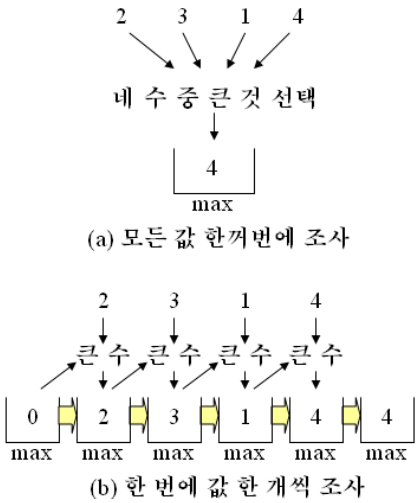

그림 2. 큰 수 고르는 예제의 작업분할
Fig. 2. An Example of Task Division

(12)의 작업은 다소 복잡한 대신 (11)처럼 두 수 중 큰 것을 고르는 것은 비교적 단순하다. 따라서 작업 분할의 원리에 따라 단순한 작업의 반복을 통해 전체 기능을 수행하는 소스 코드로 작성해볼 것을 유도하면, 그림 2의 (b)와 같은 처리를 생각해볼 수 있다. 즉, 먼저 max 의 값을 0으로 만들고 이번에는 네 변수의 값을 한 번에 한 개씩만 조사한다. 이에 따르면 처음에는 첫 번째 수인 2와 현재 max 의 값 0을 비교하여 2가 더 크므로, max 에 2를 대입한다. 그 다음에는 현재 max 와 두 번째 수 3을 비교하여 둘 중 큰 수인 3을 max에 대입하게 된다. 이런 식으로 맨 마지막 수인 4까지에 대하여 각각 비교적 간단한 작업인 두 수 중 큰 것을 선택하여 max 에 대입하는 것을 반복해나가면 최종적으로는 max 에 원하던 값인 4를 저장할 수 있다.

그림 2의 (b)에 도달하였으면, 이제 이러한 개념을 소스 코드로 구현할 차례이다. (11)을 이용하면 이제 네 수 중 큰 수를 뽑아 max 에 저장하는 작업은 다음과 같이 수정된다.

```
max = 0                                   (13.1)

If max < a Then                           (13.2)
  max = a
Else
  max = max
End If

If max < b Then                           (13.3)
  max = b
Else
  max = max
End If

If max < c Then                           (13.4)
  max = c
Else
  max = max
```

```
End If

If max < d Then                                    (13.5)
  max = d
Else
  max = max
End If
```

이렇게 주어진 문제를 분석하고, 작업 분할의 원리를 적용하는 단계에서는 일반적으로 기초 문법 정도를 이용하여 소스 코드를 작성 가능하며, 학습자들은 기존 지식을 통해 새로운 문제에 접근할 수 있다. 또 이 단계의 소스 코드는 (13.2)~(13.5)처럼 비슷하거나 같은 작업이 반복적으로 등장한다는 특징이 있다.

이제 중복 작업 제거를 통해 (13)을 보다 간결하게 나타내기 위하여 먼저 프로시저의 개념을 적용해 보기로 한다. 비주얼베이직에서 프로시저는 자바, C언어의 메소드, 함수 등에 대응되며 기능 하나를 독립적으로 만들 때 사용한다. 프로시저를 사용하는 이유 역시 중복 작업 제거 원리를 통해 설명할 수 있다.

(13.2)~(13.5)는 각각 두 수 중 큰 것을 골라 변수 max 에 저장하는 작업으로 이해할 수 있다. 따라서 자주 쓰이는 이 기능을 별도 프로시저로 구현하는 것이 좋을 것으로 보인다. 비주얼베이직에서는 다음과 같이 정수 n1, n2를 매개변수로 입력받아 둘 중 큰 쪽을 반환하는 함수 프로시저 getMax()를 작성할 수 있다.

```
Private Function getMax(n1 as Integer, n2 as Integer) as Integer

  If n1 > n2 Then
    getMax = n1
  Else
    getMax = n2
  End If
End Function                                       (14)
```

이제 (14)의 함수 프로시저를 이용하면 (13)은 아래처럼 간소화될 수 있다.

```
max = 0                                            (15.1)

max = getMax(max, a)                               (15.2)

max = getMax(max, b)                               (15.3)

max = getMax(max, c)                               (15.4)

max = getMax(max, d)                               (15.5)
```

(13)과 (15)는 정확히 같은 기능을 하지만, (14)는 필요할 때마다 두 수 중 큰 수를 골라 max에 대입하는 작업이 세세하게 작성된다는 점에서 비효율적이다. 그리고 프로시저를 통해 (15)처럼 중복 작업을 상당히 제거할 수 있다. 하지만 (15.2)~(15.5)는 여전히 비슷한 작업들의 연속이며, 아직 중복 작업 제거 원리가 완전히 실현되지는 않았다고 볼 수 있다. 따라서 앞 의 예제들처럼 배열과 반복문을 적용하여 중복 작업을 제거해 보기로 한다.

먼저 a, b, c, d처럼 개별적인 여러 변수들에 저장된 값을 사용하면 (15.2)~(15.5)를 간편히 처리하기 어렵다. 따라서 개별 변수 대신 4개의 원소를 갖는 1차원 배열 a를 사용하면 아래와 같이 수정할 수 있다.

```
max = 0                                            (16.1)

max = getMax(max, a(0))                            (16.2)

max = getMax(max, a(1))                            (16.3)
```

```
max = getMax(max, a(2))                    (16.4)

max = getMax(max, a(3))                    (16.5)
```

이에 더하여 반복문을 적용하면, 비로소 아래와 같이 중복 작업 제거의 원리가 완전히 실현되고, 더 이상은 비슷한 작업을 여러 번 작성하는 일이 없음을 알 수 있다.

```
max = 0
For i = 0 To 3
  max = getMax(max, a(i))
Next i                                     (17)
```

Ⅴ. 흐름 제어를 위한 프로그래밍 교수 전략

앞에서 나온 네 수 중 가장 큰 것을 고르는 예제의 경우, 가장 모범적인 구현은 (17)이다. 초심자들에게 처음부터 이 소스 코드를 제시하고, 도식화 등을 통해 (17)이 갖는 의미를 풀어서 해설하면 학습자들이 이러한 소스 코드가 목표로 했던 기능을 달성한다는 것을 이해할 수는 있다. 하지만 여전히 상당수의 학습자들은 이러한 내용을 직접 구현하기 어려워한다. 이러한 상황에서는 체계적이고 논리적인 문제 해결을 위한 사고력을 증진한다는 궁극적인 프로그래밍 학습 목표가 달성되기 어렵다.

이러한 점을 보완하기 위해 본 논문은 작업 분할 및 중복 작업 제거 원리를 학습자들에게 소개하고, 이들에 기초하여 예제 소스 코드 구현 과정을 제시할 것을 제안한다. 학습자들은 이 과정을 통해 주어진 문제를 분할해보고 여러 구문들의 일반화된 표현을 만들면서 자연스럽게 문제 상황의 분석 및 체계적인 해결에 대한 연습을 하게 될 것으로 기대된다. 또한 스스로 (17)과 같은 최종적인 소스 코드를 직접 작성하지 못하는 학습자들도 (11)~(16)과 같이 그 이전 단계를 구현해보게 하면서 기존에 배운 내용들을 능동적으로 활용하게 하고 단계적으로 최종적인 형태로 유도할 수 있다.

이러한 단계적인 흐름 제어 프로그래밍 교수 전략은 그림 3에 요약되어 있다. 먼저 0단계에서는 학습자들에게 처음 예제가 제시되어 소스 코드 작성을 시작하게 된다. 종종 초심자들은 제시된 기능을 어떻게 구현해야 하는지 접근하기 힘들어 하고, 편집기를 열어둔 채 타이핑을 시작하지 못하는 경우도 있다. 이러한 학생들에게 바로 완성된 소스 코드를 소개하는 경우에는 학습자들이 이전에 배운 기초 문법을 능동적으로 활용해 보지 못한 채 수동적으로 교수자가 제시하는 모범 구현 사례를 접하게 된다. 따라서 0단계에 머무르는 학습자들에게는 현재 기능을 단순한 구문들로만 작성하여 1단계를 달성하도록 유도하는 것이 필요하다. (1), (5), (6), (12)의 경우가 이에 해당하며, 이 단계에서 (5)처럼 실제 실행이 불가능한 구문이 도출될 수도 있다. 또한 (12)처럼 목표 기능이 복잡한 경우에는 (11)처럼 보다 단순한 경우를 상정하여 이 단계를 진행하게 하는 것도 도움이 된다.

1단계에서 머무른 학생들에게는 이제 2단계로 진행하기 위하여 현재 묘사한 작업들을 보다 단순한 작업들의 반복으로 분할할 것을 유도한다. (2), (7), (13)의 소스 코드들이 이 단계에 해당하며, 때로는 그림 1, 그림 2와 같은 도식화도 학습자의 이해를 도울 수 있을 것이다.

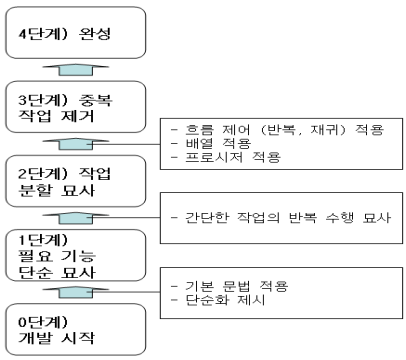

그림 3. 기본 흐름 제어 교수 전략
Fig. 3. Instructional Strategy for Flow Control Statements

2단계가 달성되면 소스 코드에는 비슷하거나 동일한 작업이 여러 번 작성되어 있을 것이다. 이러한 경우에는 현재 소스 코드에서 중복이 발생하는 부분을 찾고, 이를 효과적으로 묘사하도록 유도하는 것이 필요하다. 이를 위해 흐름 제어 구문이나 배열, 프로시저 등의 개념을 활용할 수 있고, 이를 통해중복 작업을 제거한 3단계로 갈 수 있다. 이후에는 최종적인 마무리를 거쳐 예제에서 요구하는 기능이 완성된다.

VI. 실제 적용 및 설문 조사 결과

본 논문에서 제안한 프로그래밍 원리 및 교수 전략은 충남 천안 소개 H대학 산업경영학부 학생들에게 프로그래밍 기초 과목을 강의하는데 적용되었다. 여기서는 2010년 가을 학기에 신입생 2개 분반에서 실시한 설문 조사 결과를 통해 그 유용성을 논의해보고자 한다. 2개 분반 각각 45명씩 총 수강 인원은 90명이고, 그 중 설문조사 응답자는 82명이다.

6.1 전산 비전공 학부생들과 프로그래밍 과목

설문조사의 전반부는 먼저 산업경영학부 학생들이 대학 입학 전에 프로그래밍에 대한 지식이 어느 정도 있는지, 그리고 프로그래밍 과목에 대한 인식은 어떠한지를 조사하기 위한 문항들로 이루어져 있고, 그 결과는 표 1~표6에 정리되어 있다.

먼저, 표 1에서 볼 수 있듯이 산업경영학부 학생들은 인문계 문과 고등학교에서 진학하는 비율이 75% 정도로 높고, 이로 인해 수학, 과학, 전산 쪽 과목들에 비해 인문 계통 과목을 선호하는 경향이 있다. 표 2에는 산업경영학부 개설 과목들 및 최근 학생들의 진로를 감안하여 향후 희망 직종을 조사한 것으로, IT/컴퓨터 관련 분야에 대한 관심이 저조한 것을 볼 수 있다.

표 3~표 6에는 프로그래밍 과목을 수강하기 전, 프로그래밍 과목과 관련된 사전 지식이나 선입견 같은 것을 갖고 있었는지에 대한 응답 결과이다.

표 1. 수강생 출신 고등학교 분류
Table 1. High school categories of students

항목	응답자 (명)
1. 인문계 문과	62
2. 인문계 이과	12
3. 실업계	8 (상업: 5, 공업: 2, 기타: 1)

표 2. 장래 희망 직종 분야 (3개까지 선택)
Table 2. Preferred occupation (up to 3)

항목	응답자 (명)
1. 인사/노무	47
2. 회계	21
3. 마케팅/세일즈	49
4. IT/컴퓨터 관련	8
5. 품질/공정관리	8
6. 직업 교육	12
7. 금융	28
8. 공무원	24
9. 기타 일반 사무직	14
10. 창업	14
11. 기타	13 (예술, 조리, 디자인 등)

표 3. 이전에 프로그래밍을 배운 경험 유무
Table 3. Prior knowledge on computer programming

항목	응답자(명)
1. 전혀 없다.	73
2. 배운 적 있으나 잘 모른다.	5
3. 기초적인 문법 정도는 알고 있었다.	4
4. 약간의 프로그래밍 능력을 가지고 있었다.	0
5. 간단한 프로그램 작성에 자신이 있었다.	0

표 3을 보면, 대부분의 응답자 (89%) 가 프로그래밍을 이전에 전혀 배운 적이 없다고 응답했고, 배운 적은 있으나 잘 모른다는 응답까지 합하면 약 95%의 수강생들이 사전 지식이 거의 없는 상태에서 기초 프로그래밍 과목을 수강한다는 것을 알 수 있다. 표 4에서는 프로그래밍을 이전에 접해본 적이 있다면 어떤 경로를 통해서였는지에 대한 응답 결과이다.

표 4. 프로그래밍을 이전에 학습했던 경로
Table 4. How did you study programming language before this course?

항목	응답자(명)
1. 독학	0
2. 초, 중, 고 정규 과목	6
3. 초, 중, 고 선택 과목 또는 특별 활동	1
4. 학원	0
5. 기타	1

이렇게 프로그래밍을 본격적으로 학습해 본 적이 없는 수강생이 대부분이지만, 많은 산업경영학부 학생들은 프로그래밍을 비롯한 전산 관련 과목이 굉장히 어렵다는 선입견을 갖는다. 표 5에서도 2010년 가을 프로그래밍 기초 과목 수강생 중, 응답자의 66% 가량이 프로그래밍을 배우기도 전에 이미 어려울 것이라는 인상을 갖고 있었음을 알 수 있다.

표 5. 프로그래밍 과목 수강 전, 프로그래밍은 어렵다는 인상이 있었는지
Table 5. Did you think that computer programming is difficult before this course?

전혀 아니다	아니다	보통	그렇다	매우 그렇다
1	9	10	29	25

표 6. 프로그래밍이 어렵다는 인상이 있었다면 그 이유는 무엇인지
Table 6. Why did you think that computer programming is difficult before this course?

항목	응답자 (명)
1. 친구나 선후배의 경험담을 통해	10
2. 과거 프로그래밍을 접해본 경험 때문에	2
3. 잘은 모르지만 막연한 이미지 때문에	29
4. 기타	12

이러한 선입견은 표 6에서 볼 수 있듯이 주로 막연한 이미지나 주위의 경험담 등으로 인해 생기는 것으로 조사되었다. 표 6의 응답 항목 중, 기타를 선택한 응답자들은 주로 '자신이 컴퓨터를 잘 모르고, 프로그래밍이 생소한 분야'라는 부가적인 이유를 작성하였다.

이러한 결과들에서 볼 수 있듯이, 최근 여러 대학에서 컴퓨터 및 IT 관련 교과목을 개설하고 있지만 전산비전공 학부생들은 프로그래밍을 이전에 접해본 적이 없는 경우가 많고, 프로그래밍은 대단히 어려운 것이라는 인상을 갖기 쉽다. 또한 실제 프로그래밍 과목 수강 중에도 내용을 이해하고 응용하는데 어려움을 많이 겪는다.

6.2 교수 전략의 효과성

설문 조사의 후반부는 본 논문에서 소개한 교수 전략의 유용성을 검증해 보기 위한 문항들로 구성되었고, 각각의 응답 결과는 표 7~표 11에 요약되어 있다.

표 7은 2010년 가을 학기 프로그래밍 기초 과목의 주요 내용들 중, 맨 처음 접했을 때 어려웠던 항목을 선택하는 문항의 응답 결과이다. 이 과목은 표 7의 7가지 항목들을 순서대로 강의하였으며, 본 논문에서 제안하는 프로그래밍 원리와 교수 전략은 이 중 네 번째 항목인 반복문부터 여섯 번째 프로시저 부분을 설명하는데 적용되었다. 표에서 볼 수 있듯이 수강생들은 앞 부분의 기초 문법 단계보다는 반복문을 비롯한 흐름 제어 부분에서 어려움을 많이 겪는다. 참고로 표 7의 마지막 항목인 실전 예제는 한 학기 동안 배운 내용들을 종합하여 실제로 어느 정도 유용한 프로그램을 작성해보는 단계로, 여기서도 흐름 제어 구문들이 사용되며, 수강생들이 느끼는 난이도가 비교적 높았던 것으로 조사되었다.

프로그래밍 수강하기 전의 학생들은 막연하게 프로그래밍은 어렵다고 생각하기 쉽다. 반면 프로그래밍을 수강한 후, 프로그래밍이 어렵게 느껴지기 쉬운 이유에 대해 설문한 결과는 표 8과 같다. 대부분의 응답자 (79%)가 이 항목에서는 2번 항목을 선택하였다. 여기서도 프로그래밍 과목에서 단순히 문법을 나열식으로 소개하고 암기하는데 초점을 맞추는 학습보다 특

정 기능을 어떻게 구현해 나갈 것인지에 대한 논리적인 분석 능력을 학습하는 것이 필요함을 알 수 있다.

표 7. 한 학기 동안 처음 접했을 때 가장 어려웠던 부분 (3개까지 선택)
Table 7. The most difficult part of this course (up to 3)

항목	응답자 (명)
1. 기본 문법	9
2. 컨트롤과 이벤트	17
3. 조건문	29
4. 반복문	59
5. 배열과 사용자 정의 자료형	33
6. 프로시저	46
7. 실전 예제	42

표 8. 프로그래밍이 어렵게 느껴지기 쉬운 이유는 무엇이라고 생각하는가
Table 8. The reason why many people have trouble in learning computer programming

항목	응답자 (명)
1. 기본 문법의 암기	2
2. 요구하는 기능의 논리적 표현	65
3. 수학적인 지식의 부족	5
4. 에러 원인 찾기 어려움	11

표 9. 이번 학기 프로그래밍 수강 후 느껴지는 프로그래밍의 난이도
Table 9. Perceived difficulty of computer programming after this course

항목	응답자 (명)
1. 쉬움	4
2. 생각보다 어렵지 않음	39
3. 보통 (타 과목과 유사)	26
4. 어려움	13

표 10. 작업 분할의 원리가 프로그래밍 이해에 도움이 되는지
Table 10. Do you think that the principle of task division is useful?

전혀 아니다	아니다	보통	그렇다	매우 그렇다
2	0	24	49	8

표 11. 중복 작업 제거의 원리가 프로그래밍 이해에 도움이 되는지
Table 11. Do you think that the principle of duplication removal is useful?

전혀 아니다	아니다	보통	그렇다	매우 그렇다
0	1	23	48	11

끝으로 본 논문에서 제안한 교수 전략을 적용한 2010년 가을 학기 프로그래밍 기초 과목을 수강하면서 느꼈던 학습 난이도 및 교수 전략의 유용성에 대한 응답 결과를 표 9~표 11에 정리하였다.

표 9에서는 응답자의 52% 가량이 생각했던 것보다 프로그래밍 과목 수강 및 학습이 쉬웠다(쉬움 또는 생각보다 어렵지 않음으로 응답)고 응답하였음을 알 수 있다. 또한 어렵다는 응답은 16% 정도에 그쳤다. 일반적으로 학생들이 프로그래밍 과목을 부담스러워하는 경향에 비추어 볼 때, 이는 본 논문의 교수 전략이 어느 정도 효과적이었을 입증하는 결과로 보인다.

또한 표 10~표 11에서 작업 분할 및 중복 작업 제거 원리를 숙지하는 것이 프로그래밍 이해에 도움이 된다(그렇다 또는 매우 그렇다로 응답)는 응답이 각각 70% 정도로 조사되어, 본 논문의 교수 전략이 수강생들의 교과목 이해에 상당히 기여했음을 알 수 있다.

VII. 결론 및 추후 연구과제

컴퓨터 및 IT관련 기술 교육이 대학에서도 강조되는 추세에 있으나, 초심자들은 프로그래밍 학습에 부담을 느끼는 경우가 많다. 더구나 전산 비전공 학부생들은 자신이 진로와 프로그래밍이 그다지 큰 관련이 없다고 여기는 경향까지 있다. 따라서 여러 가지 문법 소개에 초점을 맞추기보다 효과적인 교수 전략을 통해 학습자의 이해를 돕는 것이 필요하다.

본 논문은 초심자들이 기초 프로그래밍 학습 과정

중에서도 반복문을 비롯한 흐름 제어 부분에서 어려움을 많이 겪는다는 점을 지적하고, 이 부분을 위한 교수 전략을 제안하였다. 특히 기초 프로그래밍 과목에서 소개되는 구문들 중, 반복문, 배열, 프로시저 등의 여러 구문들이 실제로는 작업 분할 및 중복 작업 제거의 원리는 일반적인 개념을 실현하는 데 관련이 있음을 밝히고, 이러한 원리에 의거하여 단계적으로 프로그램 소스 코드를 완성하는 것을 유도함으로써 최초 소스 코드 작성을 스스로 하지 못하는 학습자들도 가급적이면 기존의 지식을 유기적으로 활용하여 효과적으로 새로운 개념을 배울 수 있도록 하는 교수 전략을 제안하였다.

이러한 내용들은 실제 경영학 관련 전공 학부생들을 대상으로 한 프로그래밍 기초 과목 강의에 적용되었으며, 설문 조사 결과 초심자들의 프로그래밍 이해에 상당한 도움이 되었던 것으로 보인다. 하지만 본 논문에서는 설문 대상이 80여명으로 제한되어 있고, 동일한 수업을 들은 수강생을 대상으로 하여 여러 교수 전략의 비교 분석이 어려웠다는 한계가 있었다. 이에 따라 향후 보다 다양한 전공의 학생들을 대상으로 실제 학업 성취도의 향상 여부를 정량적으로 검증하는 연구가 필요할 것으로 생각된다.

부가적으로 저자들은 향후에도 실제 강의를 통해 이러한 교수 전략에 부합하는 프로그래밍 예제 및 관련 콘텐츠를 다양하게 개발할 예정으로 있다. 또한, 이러한 프로그래밍 교수 전략을 토대로 학습자 스스로 프로그래밍 연습을 하며 소스 코드를 작성하는 과정을 지원할 수 있는 코스웨어나 각 학습자들에게 개인화된 맞춤형 튜터링 서비스를 제공하는 지능형 튜터링 시스템 개발을 목표로 연구를 계속하고자 한다.

참고문헌

[1] 김갑수, "초등학생들의 창의력과 논리력 향상을 위한 프로그래밍 언어 교수전략에 관한 연구," *정보교육학회 논문지*, 제14권, 제1호, pp.88-97, 2010.

[2] 김종훈, 김종진, 고정림, "컴퓨터 활용 능력 및 학습 태도 향상을 위한 창의적 교수법 연구," *한국콘텐츠학회 논문지*, 제6권, 제7호, pp.99-110, 2006.

[3] 석현태, "컴퓨터공학 전공자의 전공 관련 자격증 획득률 제고를 위한 교과과정 보완 연구," *한국콘텐츠 학회/한국통신학회 추계종합학술대회 논문집*, 제1권, 제2호, pp.43-47, 2003.

[4] 이화현, 임연욱, 이옥화, "문제해결을 위한 ICT 활용 능력 분석: 문맥 속에서 대학생의 기능 및 논리 능력 학습하기", 한국컴퓨터교육학회 논문지, 제9권, 제3호, pp.85-96, 2006.

[5] 박영미, 유관희, "초중등학교에서 수학교육체계와 연계된 컴퓨터 프로그래밍 교육과정과 교수방법," *한국콘텐츠학회 논문지*, 제8권, 제1호, pp.116-127, 2008.

[6] Ben-ari, M., "Constructivism in Computer Science Education," *Journal of Computers in Mathematics and Science Teaching*, Vol.20, No.1, pp.45-73, 2001.

[7] 교육인적자원부, 교육혁신과 인적자원 개발을 위한 교육정보화 종합 발전 방안, 정부간행물등록번호 11-1340400-000004-01, 2001.

[8] Ismail, M.N., Azilah, N., and Naufal, U., "Instructional Strategy in the Teaching of Computer Programming: A Need Assessment Analyses," The Turkish Online Journal of Education Technology, Vol.9, No.2, pp.125-131, 2010.

[9] Rudder, A., Bernard, M., and Mohammed, S., "Teaching Programming Using Visualization," *Proceedings of the 6th conference on IASTED International Conference Web-Based Education*, Vol.2, pp.487-492, 2007.

[10] Kessler, C.M., and Anderson, J.R., "Learning Flow of Control: Recursive and Iterative Procedures," *Human-Computer Interaction*, Vol.2, pp.135-166, 1986.

[11] Sinha, A.P., and Vessey, I., "Cognitive Fit: An Empirical Study of Recursion and Iteration," *IEEE Transactions on Software Engineering*, Vol.18, No.5, pp-368-379, 1992.

[12] Kelleher, C., and Pausch, R., "Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice

Programmers," *ACM Computing Surveys*, Vol.37, No.2, pp.83-137, 2005.

[13] Taylor, J., "Analysing Novices Analysing Prolog: What Stories Do Novices Tell Themselves About Prolog," *Instruction Science*, Vol.19, pp.283-309, 1990.

[14] Sleeman, D., Putnam, R.T., Baxter, J.A., and Kuspa, L., "An Introductory Pascal Class: A Case Study of Student Errors," In R.E. Mayer(Ed.), *Teaching and Learning Computer Programming*, Hillsdale, NT., pp.237-257, 1998.

[15] Samurçay, R., "The Concept of Variable in Programming: Its Meaning and Use in Problem-Solving by Novice Programmers," In E. Soloway & J.C. Spohrer (Ed.), *Studying the Novice Programmers*, Hillsdale, NJ., pp.161-178, 1989.

[16] Haberman, B., and Ben-David Kolikant, Y., "Activating 'Black Boxes' Instead of Opening 'Zippers' - a Method of Teaching Novices," *Proceedings of the 6th Annual conference on Innovation and Technology in Computer Science Education*, Canterbury, UK., 2001.

[17] Salomon, G., and Perkins, D.N., "Transfer of Cognitive Skills from Programming: When and How?," *Journal of Educational Computing Research*, Vol.3, pp.149-169, 1987.

[18] Fritz, C., and Gil., Y., "Towards the Integration of Programming by Demonstration and Programming by Instruction using Golog: Extended Version with Proofs," *Proceedings of AAAI Workshop on Plan, Activity, and Intent Recognition*, 2010.

[19] Fritz, C., and Gil., Y., "A Formal Framework for Combining Natural Instruction and Demonstration for End-User Programming," *Proceedings of 16th International Conference on Intelligent User Interfaces*, 2011.

[20] Anzai, Y., and Uesato, Y., "Learning Recursive Procedures by Middleschool Children," *Proceedings of the 4th Annual conference on the Cognitive Science Society*, pp.100-102, 1982.

[21] Smith III, J.P., diSessa, A.A., and Roschelle, J., "Misconceptions Re-conceived: A Constructivist Analysis of Knowledge in Transition," *The Journal of the Learning Sciences*, Vol.3, No.2, pp.115-163, 1993.

[22] Merrill, M.D., "First Principles of Instruction," *Educational Technology Research and Development*, Vol.50, No.3, pp.43-59, 2002.

감사의 글

이 논문은 동아대학교 교내연구비 지원에 의하여 연구되었음.

저자소개

김준우(Jun-Woo Kim)

2001년 한국과학기술원 산업공학과 졸업(공학사)
2003년 한국과학기술원 산업공학과 졸업(공학석사)
2009년 한국과학기술원 산업 및 시스템공학과 졸업(공학박사)
2009년~2010년 한국기술교육대학교 산업경영학부 대우교수
2011년~현재 동아대학교 산업경영공학과 조교수
※ 관심분야 : 데이터마이닝, 지능형 시스템, 서비스 관리, Operations Reseach, e-러닝

주지영(Jee-Young Joo)

2001년 충남대학교 물리학화 졸업(이학사)
2004년 한국교원대학교 대학원 공통과학교육과 졸업(교육학석사)
2004년~2010년 (주) 브레인벨리 선임연구원
2010년~현재 한국과학기술원 과학영재교육원 연구원
※ 관심분야 : 과학영재교육, e-러닝, 과학사

임광혁(Kwang-Hyuk Im)

1995년 한국과학기술원 전산학과 졸업(공학사)
2000년 한국과학기술원 산업공학과 졸업(공학석사)
2006년 한국과학기술원 산업공학과 졸업(공학박사)
2006년~2008년 삼성전자 반도체연구소 책임연구원
2008년~현재 배재대학교 전자상거래학과 조교수
※ 관심분야 : 지식서비스, 경영정보시스템, 데이터마이닝, 전자상거래, 고객관계관리, 공급사슬관리