Journal of Engineering Education Research
Vol. 15, No. 3, pp. 66~71, May, 2012

기초적인 프로그래밍 교육을 위한 컴퓨터 언어의 특성 및 개발 환경의 임상적 비교 분석

강대기†
동서대학교 컴퓨터정보공학부

Clinical Comparative Analysis of Characteristics of Computer Programming Languages and their Development Environments for Basic Programming Education

Dae-Ki Kang†
Division of Computer & Information Engineering, Dongseo University

ABSTRACT

In this paper, we try to explore basic factors that defines easy-to-learn programming language and easy-to-learn development environments for novice students who have not been exposed to computer programming language education. For these purpose, we investigate and analyze computer programming languages that are widely used in industrial environments, and present the summary and analyzed results. From the experimental results, most novice programmers understand computer programming languages in terms of procedural programming languages rather than in terms of functional programming languages or object oriented programming languages. Furthermore, we have found that, for effective education of basic level programming languages, factors of development environments are much more important than factors of programming paradigms that the computer programming languages are based on.

Keywords: Computer programming language, Programming language education, Programming paradigm, Tangibility

I. 서 론

컴퓨터 프로그래밍을 위한 프로그래밍 언어는 여전히 대부분의 학생들에게 어려운 과목이다. 1950년대에 FORTRAN과 같은 디지털 컴퓨터를 위한 최초로 널리 쓰이는 프로그래밍 언어(Backus et al., 1957)가 등장한 이후, 수많은 프로그래밍 언어와 그를 배우기 위한 개발 환경이 등장하였다. 기본적으로 튜링 완전한(Turing complete) 프로그래밍 언어를 가지고 만들어진 하나의 컴퓨터 프로그램은 하나의 수학적 증명과 비슷하다(Sipser, 1996). 그 후로 60년이 넘는 세월이 지났으나, 여전히 많은 학생들은 프로그래밍 언어와 개발 환경을 이용하여 컴퓨터 프로그램을 구현하는 데 어려움을 겪고 있다.

본 논문에서는 컴퓨터 프로그래밍을 처음 접하는 학생들에게 배우기 쉬운 프로그래밍 언어 및 개발 환경을 결정하는 요소가 무엇인지 알아보고자 하는 목적을 가지고 있다. 이를 위해, 산업 현장에서 많이 쓰이고 있는 언어들에 대한 임상적 실험을 수행하고 그 결과를 분석해서 제시하고자 한다. 실험 결과, 초보자들이 이해하는 대부분의 컴퓨터 언어들은, 함수형 프로그래밍이나 객체지향 프로그래밍 보다는, 여전히 기본적으로 알골(Algol) (Backus et al., 1960)과 같은 언어를 계승한 절차적 프로그래밍 언어들이었으며, 컴퓨터 언어가 가지는 패러다임보다는 초보자들이 쉽게 익힐 수 있는 개발 환경에 대한 요소가, 효과적인 기초 프로그래밍 교육을 위해서는 더욱 중요함을 알 수 있었다.

Received April 9, 2012; Revised April 17, 2012
Accepted May 10, 2012
† Corresponding Author: dkkang@dongseo.ac.kr

II. 문제점

국내 대학에 컴퓨터 관련 학과가 생기고 수십 년이 넘게 프로그래밍 교육이 이루어져 왔으나, 여전히 컴퓨터 프로그래밍 학습에 많은 학생들이 어려움을 겪고 있다. 특히 컴퓨터 프로그래밍 수업의 경우, 대다수의 학생들이 프로그래밍을 배우는

데 어려움을 느끼고 있다.

이러한 경향은 이른바 “이공계의 위기”로 인해 컴퓨터 관련 학과에 우수한 학생들이 지원하지 않기 시작하면서 더더욱 심해져, 교과 과정을 보면, 과거 대학 2학년 1학기 정도에서 머물던 프로그래밍 언어 교육이 2학년 2학기를 거쳐 심지어 3학년 1학기까지 미치고 있다.

1. 프로그래밍 학습의 어려움

또한, 이러한 컴퓨터 프로그래밍 언어에 대한 학생들의 학습 경향을 보면, 어떤 학생은 매우 쉽게 프로그래밍을 배우는가 하면, 어떤 학생은 프로그래밍에 매우 어려움을 느끼고 있다. 이러한 경향을 보면, 한 가지 던질 수 있는 open problem은 과연 프로그래밍을 잘하는 데에는 유전적인 경향이 존재하는 가이다. 즉, 프로그래밍 DNA가 있는 것인가 하는 것이다. 현재, 널리 쓰이는 인기 있는 컴퓨터 프로그래밍 언어는 주로 알골 언어의 영향을 많이 받아온 것이 사실이다. 따라서, 이러한 질문은 학생들의 두뇌에서 알골과 같은 언어를 이해하는 내재된 능력이 있는가라는 질문으로까지도 미칠 수 있을 것이다.

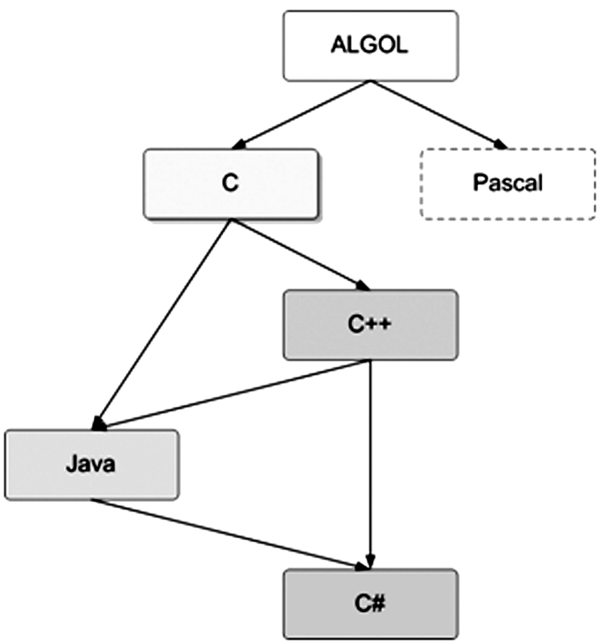

그림 1 인기 언어들의 계통도

2. 동기 부여의 문제

컴퓨터 프로그래밍 언어의 학습을 어렵게 하는 다른 요소로는, 대부분의 국내 대학이 취업 위주의 학문을 가르치게 된 요즘, 학생들은 취업에 도움이 될 거 같은 언어를 공부한다는 점이다.

그러다 보니, 학생들은 다음과 같은 질문을 교수에게 자주 하게 된다.

“어떤 언어를 공부해야 장래에 좋을까요?”

또한 대부분의 학생들은 경제적인 문제와 학비에 대한 부담으로, 아르바이트를 해서 학비를 버느라 공부에 투자할 시간이 과거에 비해 많이 부족해졌다. 그래서 컴퓨터 프로그래밍 언어를 배우는 학생들 중 컴퓨터 프로그래밍 언어에 대한 소양이 부족한 일부 학생들은 프로그래밍 언어의 학습 동기에 대해 다음과 같은 의견을 토로하고 있다.

“여러 언어를 공부하기 보다는 장래에 도움이 되는 언어 하나를 빨리 공부하고 싶다!”

또한 몇 년 전부터 스마트폰이 인기를 끌자, 안드로이드, iOS, 윈도우폰7과 같은 스마트폰 프로그래밍 강의가 인기를 끌기 시작하는 것도 이러한 학생들의 학습 동기 및 경향과 관련이 있다.

III. 관련 연구 및 역사점 흐름

이제 더 나아가, 본 섹션에서는, 컴퓨터 언어와 컴퓨터 공학의 유관성, 컴퓨터 언어의 발전 경향, 프로그래밍 언어 패러다임, 프로그래밍 언어 교육의 일반적 흐름, 그리고 쉬운 프로그래밍 언어 교육에의 도전 및 시도에 대해 알아보겠다.

1. 컴퓨터 언어와 컴퓨터 공학의 유관성

들어가기 전에 우선 한 가지 분명한 사실은, 특정 프로그래밍 언어를 공부하는 것은 컴퓨터 공학 또는 컴퓨터 과학이라는 학문의 전공과는 무관하다는 것이다. 컴퓨터 공학을 보면, 컴퓨터 소프트웨어와 컴퓨터 하드웨어에 대한 공부 및 연구를 같이 수행하며, 컴퓨터 과학에서는 컴퓨터 소프트웨어 및 컴퓨터에 대한 수학적인 분석도 중요시한다. 예를 들어, 컴퓨터 과학에서 중요시하는 알고리즘(algorithm) 및 컴퓨테이션 이론(theory of computation)에 대한 수업은, 컴퓨터 공학에서는 덜 중요하게 다루어질 수도 있다. 한 가지 안타까운 점은, 컴퓨테이션 이론은 매우 중요한 학문임에도, 수학적인 기본기가 있어야 하는 부분 때문에, 대부분의 국내 대학에서 다소 소홀하게 다루어지고 있다는 것이다.

그럼에도 대부분의 컴퓨터 전공 학부생들에게 컴퓨터 공부는 프로그래밍 언어 공부로 여겨지고 있다. 실제로 어떤 대학교 3학년 학생은 강의를 하는 교수에게 “교수님의 전공은 C++인가요?”라고 묻기도 했다. 이 3학년 학생은 그 당시 성적이 우수한

축에 드는 학생이었는데, 이것은 그만큼 대학에서 컴퓨터 공학의 학문 수준이 낮아졌다는 것을 반증하는 것이기도 하며, 그로 인해 학생들이 그만큼 컴퓨터 프로그래밍 언어 공부를 어려워한다는 것을 반증하는 것이기도 하다.

2. 컴퓨터 프로그래밍 언어

컴퓨터 프로그래밍 언어는 그 발전 양상에 따라 다음과 같이 여러 세대로 나누어진다(Sammet, 1996).

- 1세대 언어(1950년도 초반)
  - 기계어(machine language)
- 2세대 언어(1950년대)
  - 어셈블리어(assembly language)
- 3세대 언어(1950년대 후반)
  - FORTRAN, LISP, COBOL
  - Algol(현재 대부분의 프로그래밍 언어에 영향)
  - C, Prolog, Simula, ML, APL, PL/I
  - Java, Perl
- 4세대 언어
  - Structured Query Language(SQL)

1950년대 초반의 1세대 언어를 보면, 고수준의 추상화를 보이는 프로그래밍 언어는 존재하지 않았고, 기계나 CPU가 직접 이해할 수 있는 기계어(machine language)만이 존재하였다. 그러다가, 각각의 기계어를 단순한 영어 단어 및 관련 식과 일-대-일 대응이 되게 구성한 어셈블리어(assembly language)가 등장하였다. 1950년대 후반, 3세대 언어가 등장하면서, 다소의 추상화가 구현되기 시작하였는데, 이때에 등장한 언어로는 수치 해석을 위한 FORTRAN, 인공 지능을 위해 만들어진 LISP (McCarthy et al., 1960), 상업용 트랜잭션을 위해 만들어진 COBOL(Sammet, 1978) 등이 있다. 또한 이러한 언어들 이후에, 고수준의 추상화와 범용성을 두루 갖춘 알골 언어가 등장한다. 이 후, 인공 지능 분야에서는, Prolog 언어가 등장하였고, 알골 언어의 영향을 받아 C, Simula(Sklenar, 1997), Meta Language(ML), A Programming Language(APL), Programming Language One(PL/I) 등이 등장하였다. 인터넷의 등장과 인터넷 기반 애플리케이션들의 다양한 활용으로 Java 및 Perl과 같은 언어들도 등장하게 되었다. 마지막으로, 우리가 이러한 3세대 언어의 번성과 별도로 주목할 만한 것은 데이터베이스 분야에서 제 4세대 언어로 Structured Query Language(SQL)이 등장한 것이다.

3. 컴퓨터 프로그래밍 언어 패러다임

이렇게, 컴퓨터 프로그래밍 언어가 다양하게 발전하고, 고수준의 추상화를 구현할 수 있게 되면서, 프로그래밍 패러다임 또한 다양하게 발전하였다. 다음은 이러한 프로그래밍 패러다임의 일부로, 이 외에도 많은 프로그래밍 패러다임이 존재한다.

- 구조적 프로그래밍(structured programming) – 절차형 프로그래밍의 하위 개념으로, 순차/선택/반복의 세 가지 구조만 사용하며 특히 GOTO문을 최대한 피함(Rubin, 1987).
- 절차형(또는 명령형) 프로그래밍(procedural programming) – 프로시쥬어 호출의 개념을 바탕으로 하고 있는 프로그래밍 패러다임
- 선언형 프로그래밍(declarative programming) - 프로그램이 어떤 방법으로 해야 하는지를 기술하는 것이 아니라, 어떤 것을 해야 하는지를 기술하는 방식으로 구성되는 프로그래밍 기법
- 함수형 프로그래밍(functional programming) - 계산을 수학적 함수의 조합으로 생각하는 방식의 프로그래밍으로, 그 기원은 컴퓨터를 수학적 모델로 간주했던 재귀 함수(recursive function)이나 람다 계산법(lambda calculus)에 있음
- 객체지향 프로그래밍(object oriented programming) - 컴퓨터 프로그램을 명령어의 목록으로 보는 시각에서 벗어나 여러 개의 독립된 단위, 즉 “객체”들의 모임으로 파악하고자 하는 것. 이러한 객체들은 내부 데이터와 인터페이스로 구성됨.
- 애스펙트 지향 프로그래밍(aspect oriented programming) - 컴퓨터 프로그램을 애스펙트들로 파악하고자 하는 패러다임. 애스펙트는 기존의 프로그램 로직을 관심사(concern)이라 불리우는 응집도 높은 단위로 분리함. 기본적으로 비즈니스 로직과 시스템 서비스(감사 Audit, 로깅, 트랜잭션 관리 등)를 분리하는데 도움이 되며, 실제로 Java의 Spring 프레임워크에서 사용됨(Kiczales et al., 1997).

재미있는 점은 최근 언어들은 여러 패러다임을 동시에 가지고 있다는 것이다. 예를 들어 C++는 진화를 거듭하여, C 언어에서 상속받은 기존의 절차형 프로그래밍 패러다임에 객체지향 프로그래밍 패러다임을 가지고 있었다. 최근의 C++는 여기에 더해서, 템플릿(template)을 통한 일반화 프로그래밍(general programming) 패러다임과 템플릿 메타 프로그래밍(template metaprogramming)을 통한 함수형 프로그래밍(functional programming) 패러다임까지 가지고 있다(물론 순수한 함수형 프로그래밍 패러다임의 정확한 의미로 들어가면, 다소 논의의 여지는 있다.).

4. 프로그래밍 언어 교육의 일반적 흐름

이제, 프로그래밍 언어 교육의 일반적인 흐름을 알아보고자 한다. 주목할 점은 프로그래밍 언어 교육이 실제로는 이론 강의 위주로 흘러간다는 것이다. 예를 들어, 알골 언어와 비슷한 현재의 인기 있는 프로그래밍 언어들의 교육 흐름을 설명해 보겠다.

우선, 처음에는 해당 프로그래밍의 역사나 배경 이론이 나온다. 그리고 나서, 자료형을 설명하고, 연산자를 설명한 후에, 프로그램의 제어문을 설명한다. 다음으로는, 객체 지향의 경우, 클래스를 설명한다. 그리고 나서, 클래스 상속, 멤버, 접근 제한 등등을 설명한다. 여기에 추가로 이름 공간까지 언급되기도 한다. 만일 해당 언어가 어떤 언어이냐에 따라, 예를 들면, C의 함수 포인터나 그 함수 포인터의 복잡한 응용, C++의 멤버 함수 포인터, 펑크터(functor) 또는 템플리트 메타 프로그래밍(template metaprogramming), Java의 제네릭(generic)이나 클로저(closure), C#의 델리게이트 (delegate) 및 람다 식(lambda expression)으로의 응용까지 설명한다면, 프로그래밍에 재능이 없는 대부분의 학생들을 수업 시간에 확실히 잠재울 수 있을 정도이다.

5. 쉬운 프로그래밍 교육의 도전

이러한 경향에도 불구하고, 쉬운 프로그래밍 교육을 하고자 하는 시도는 계속 이루어져 왔다. 그 가장 전형적인 예로는 Small Talk가 있다(Goldberg, 1983). Small Talk는 Alan Kay, Dan Ingalls, Adele Goldberg 등이 개발한 컴퓨터 프로그래밍 언어이다. 이들은 고안해 낸 Small Talk 언어를, 애플, 월트 디즈니

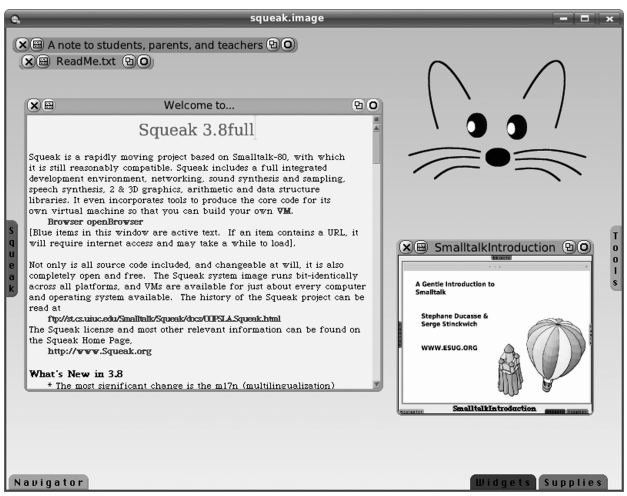

그림 2 Squeak 시스템

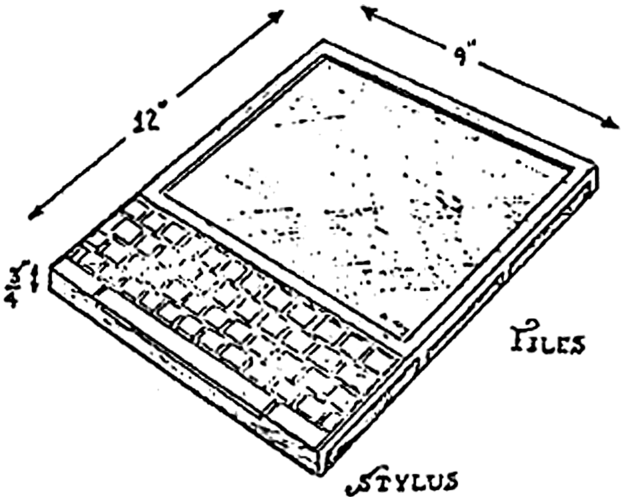

그림 3 Dynabook 개념(앨런 케이, 1972)

등의 도움을 받아, Squeak이라는 이름의 시스템으로 구현하였다(Ingalls et al., 1997). 그림 2에서 보듯이, 이것은 아이들이 타겟 사용자인 배우기 쉬우면서도 Small Talk의 모든 기능이 구현된 개발 환경이다.

또한 이들은 그림 3과 같이 Dynabook concept를 제안하는데, 이는 랩탑, 또는 태블릿 PC의 초기 개념으로, 최근 애플의 아이패드 시스템과 매우 유사함을 알 수 있다(Richards, 2008).

IV. C++ 언어와 C# 언어 학습 결과 비교

이제 본 연구에서 임상적 실험을 통해 분석한 C++언어와 C# 언어에 대한 학습 결과를 비교 분석해 보고자 한다.

1. 실험 환경 설명

우선 C++ 언어 학습 환경을 보도록 하겠다. C++ 언어 강의 수강 인원은 대학교 2학년 및 3학년으로 구성된 50명~60명으로, 강의 기간은 일 년으로 두 학기였다. 첫 학기는 기초를 가르치고, 다음 학기는 심화 과정을 가르쳤다. 교재는 “C++ 기초 플러스”(Prata, 2006)라는 책으로, 이 책은 1000 페이지가 넘지만, 자연스럽게 말하듯 매우 쉽고 상세하게 설명되어 있는 것이 특징이다. 실습 환경은 개개인의 학생이 Microsoft Visual Studio에서 프로그램을 짜면서 실습을 통해 설명할 수 있게 하였다.

결과적으로, 그럼에도 불구하고 C++라는 프로그래밍 언어의 기본적인 난해함으로 인해 많은 학생들이 어려움을 겪었다. 구체적으로 다중 상속, 캐스팅 연산자, 템플릿, 일반화 프로그래밍 등에서 많은 학생들이 수업을 따라가지 못하고 낙오되었

다. 최종적인 챕터까지 제대로 이해한 학생은 단 한명이었는데, 이 학생은 후에 명문대 대학원에 진학한 학생으로, 상대적으로 다른 학생들과 비교해 볼 때, 특출난 학생, 즉 outlier로 볼 수 있다.

C# 언어의 경우를 보면, 강의 수강 인원은 30명 정도였다. 강의 기간은 한 학기였고, 교재는 "C# 입문: 이론과 실습"(오세만, 2005)이었다. 책의 내용 자체는 쉬운 것만으로 국한하였으나, "C++ 기초 플러스"에 비해 그다지 쉽고 상세하게 서술된 형태는 아니다. 실습 환경을 말하자면, 역시 Microsoft Visual Studio에서 프로그램을 짜면서 실습을 통해 설명하였다.

결과적으로, 많은 학생들이 프로그래밍을 재미있게 여기게 되었는데, 주된 이유 중 하나는 Microsoft Visual Studio에 C# 프로그래밍 언어가 직관적으로 구현되어 있기 때문이다. 학기가 끝나도, 5~10명 가량의 학생들이 C# 언어에 큰 흥미를 느끼고 방학 동안에도 공부하겠다고 하였다.

2. 차이점 분석

이러한 차이가 나타난 이유는 무엇인지 분석하여 보았다. 본 연구에선 이에 대해 세 가지 요소를 주장하고자 한다.

- 만질 수 있는가?(Tangible): 디자인 용어
  - 특징 하나하나가 GUI와 같이 사용자에게 편리한 형태로 직접 확인 가능한가?
- 언어의 특성이 개발자 편의인가, 아니면 사용자 편의인가?
  - 언어의 기본적인 특성이 개발자의 철학을 반영한다면, 그 철학인 언어를 구성하는 시스템에 대한 철학인가 아니면 사용자에 대한 철학인가?
- 기존 언어에 대한 지식을 요구하는가? 그렇다면 어느 정도인가?
  - 기존 언어에 대한 지식이 불가피하다면, 어떤 언어의 지식을 어느 정도 따르는가?

3. 차이점 - 만질 수 있는가?

우리는 본 연구에서 "만질 수 있는가?!(Tangible?!)"라는 용어에 대해 기본적으로 배워야 할 하나의 특성을 바로 확인할 수 있는가의 여부를 나타내는 것으로 정의하였다.

C++ 언어의 각 특징은 프로그램에서 직접 보여줘야 하는 반면, C# 언어는 Microsoft Visual Studio와 연동된 특징들을 많이 가지고 있다. 더 구체적으로 설명하자면, C++에서는 특징 하나를 설명하기 위해 이론을 설명하고 프로그램을 작성해서 결과를 보여 주어야 한다. 그러나, C#에서는 예를 들어 프러퍼티는 프러퍼티 윈도우를 Visual Studio가 직접 가지고 있으며, 이벤트도 이벤트 윈도우에서 프로그래머가 직접 고를 수 있다.

C#에서는 객체를 표현하고 '.'을 누르면 관련 메쏘드나 멤버들이 자동으로 디스플레이되며, 그 결과는 바로 실행 프로그램의 GUI 와 연동된다. 이러한 점이 "만질 수 있다"로 표현될 수 있는 것이다. Microsoft Visual Studio 의 도움을 받는 이러한 C#을 이클립스의 도움을 받는 Java와 비교해 보면, 비슷할 수도 있으나, 여전히 그 편의성과 언어의 특징이 개발 환경과 연동된 수준은 C#이 더 높은데, 그 이유는 Microsoft Visual Studio가 전세계에서 가장 소프트웨어를 잘 만들고, 오랜 기간 만들어온 마이크로소프트 사의 주력 제품 중 하나이기 때문인 것으로 보인다.

4. 차이점 -언어의 특성이 개발자 편의인가, 아니면 사용자 편의인가?

C++ 언어에서 하나 하나의 특성은 그것이 적용되고 구현되는 시스템의 깊은 부분에 대한 이해를 요구한다. 예를 들어, 가상 함수, try-catch, 템플릿의 상세화, 구체화 등등이 그러하다.

가상 함수는 C#, Java, Python 등의 다른 객체 지향 언어에서는 당연히 지원되는 것인데, C++는 그 창조자이며 개발자인 비야네 스트로스트롭 교수의 고집으로 인해 여전히 virtual 이라는 키워드로 지정해야 한다. 가상 함수는 가상 테이블과 가상 포인터를 유지해야 하는데, 비야네 스트로스트롭에 의하면, 성능에 영향을 주는데도 불필요한 특성은 기본적으로 지원하지 않도록 해야 하기 때문이다.

try-catch 블록에서 finally 구문은 역시 C#, Java, Python 등의 대부분의 객체 지향 언어에서 구현되어 있으나, 역시 비야네 스트로스트롭 교수의 자원 획득은 초기화(Resource Acquisition Is Initialization; RAII) 원칙에 따라 C++에서 finally 구문은 없다. 또한, 템플릿의 상세화나 구체화에 대한 규칙도 매우 복잡하다.

5. 차이점 - 기존 언어에 대한 지식을 요구하는가? 그렇다면 어느 정도인가?

C++는 C 언어에 대한 지식을 기본적으로 요구하는데, 그러다 보니, 문법적으로 일부 비상식적인 부분을 제외하고는 C 언어의 난해한 특징을 대부분 물려받았다. 반면, C#은 Java 언어에 카피한 것으로 Java 언어에 대한 지식을 기본적으로 요구한다. 따라서, Java 언어에서 장점들을 많이 물려 받았는데, 이는 C/C++에 비해 상대적으로 쉬운 문법과 다양하게 제공되는 라이브러리 클래스들로 대변된다.

참고로, 역설적으로 쉬운 Java 언어의 장점으로 인해, 미국에선 Java 프로그래머가 C/C++ 프로그래머보다 훨씬 많고, 그로 인해 연봉이 더 적은 현상도 발생하고 있다.

V. 결 론

언어를 구현하는 시스템적 관점에서 C++는 흥미로운 언어임이 분명하다. 예를 들어, 자원 획득을 초기화에 수행해야 한다는 개발자인 비야네 스트로스트룹의 개인적인 고집 때문에 try catch 블록에서 finally를 구현하지 않고 있는 점이나, 성능 문제 때문에 가상 함수를 디폴트로 하지 않아서 필요한 경우 virtual 키워드를 사용해야 한다는 점 등은 C++ 언어가 상대적으로 독특한 언어임을 알 수 있게 해 준다. 그러나, 언어의 문법이나 이론, 또는 교육학적 관점에서는 이러한 특성은 C++이 그다지 좋지 않은 언어임을 반증하고 있는 것이다.

C# 을 가르칠 때 흥미로운 점은, 기본적으로 컴퓨터 언어에 대한 소양이 있으나, 잘 드러나지 않던 학생들이 즉각적으로 반응하고 결과를 볼 수 있다는 점에서 C# 언어를 매우 흥미로워 한다는 점이었다. 즉, 프로그래밍에 잠재된 소질은 있으나, 어려운 용어와 초기의 진입 장벽으로 어려워하던 학생들이 C#의 상대적으로 쉽게 사용할 수 있도록 잘 설계된 개발 환경으로 인해, 프로그래밍에 눈뜨게 된 좋은 예였다고 보여진다.

장래 연구로는 이러한 정성적 분석을 토대로 현재 다소 미흡한 정량적 결과를 더욱 많이 얻어서 본 가설을 확인하고 세련시키고자 한다.

본 연구는 2011년도 동서대학교 학술연구조성비 지원 과제와 지식경제부 및 부산광역시에서 지원하는 동서대학교 유비쿼터스 어플라이언스 지역혁신센터 사업에서 지원받았습니다(과제번호. B0008352).

참고문헌

1. Backus, J. W.; H. Stern, I. Ziller, R. A. Hughes, R. Nutt, R. J. Beeber, S. Best, R. Goldberg, L. M. Haibt, H. L. Herrick, R. A. Nelson, D. Sayre, P. B. Sheridan (1957). "The FORTRAN Automatic Coding System". Western joint computer conference: Techniques for reliability (Los Angeles, California: Institute of Radio Engineers, American Institute of Electrical Engineers, ACM): 188–198. doi:10.1145/1455567.1455599.
2. Backus, J. W.; Bauer, F. L.; Green, J.; Katz, C.; McCarthy, J.; Perlis, A. J.; Rutishauser, H.; Samelson, K. et al. (May 1960). Naur, Peter. ed. Report on the Algorithmic Language ALGOL 60.
3. Goldberg, A. (1983). Smalltalk-80: The Interactive Programming Environment. Addison-Wesley. ISBN 0201113724.
4. Ingalls, D.; Kaehler, T.; Maloney, J.; Wallace, S.; Kay, A. (1997). "Back to the Future: the story of Squeak, a practical Smalltalk written in itself". ACM Digital Library.
5. Kiczales, G.; Lamping, J; Mendhekar, A.; Maeda, C.; Lopes, C.; Loingtier, J.-M.; Irwin J. (1997). "Aspect-Oriented Programming". Proceedings of the European Conference on Object-Oriented Programming, vol.1241. pp. 220–242.
6. McCarthy, J.; Brayton, R.; Edwards, D.; Fox, P.; Hodes, L.; Luckham, D.; Maling, K.; Park, D. et al. (March 1960). LISP I Programmers Manual. Boston, Massachusetts: Artificial Intelligence Group, M.I.T. Computation Center and Research Laboratory Accessed May 11, 2010.
7. Richards, Mike (2008). "Why the iPhone makes 2008 seem like 1968 all over again". Open2.
8. Rubin, F. (1987). "'GOTO Considered Harmful' Considered Harmful". Communications of the ACM 30 (3): 195–196. doi:10.1145/214748.315722.
9. Sammet, J. (1978). "The Early History of COBOL". ACM SIGPLAN Notices (Association for Computing Machinery, Inc.) 13(8): 121–161.
10. Sammet, J.E. (1996). "From HOPL to HOPL-II (1978-1993): 15 years of programming language development". History of programming languages--II. New York: ACM. pp. 18.
11. Sklenar, J., Introduction to OOP in Simula - based on the 1997 seminar "30 Years of Object Oriented Programming (OOP)"
12. Sipser, M (1996). Introduction to the Theory of Computation, PWS Pub. Co.; 1 edition.

강대기 (Kang, Dae-Ki)
1992년: 한양대학교 전자계산학과 졸업
1994년: 서강대학교 전자계산학과(이학 석사)
1994년~1999년: 한국전자통신연구원(연구원)
2006년: Iowa State University(PhD in Computer Science)
2007년2월~2007년8월: 국가보안기술연구소(선임연구원)
2007년9월~현재: 동서대학교 컴퓨터정보공학부 조교수
관심분야: 기계학습, 관계학습, 통계적그래피컬모델, 온톨로지학습, 침입탐지, 웹방화벽, 웹마이닝, 컴퓨터비전
Phone: +82-51-320-1724
E-mail: dkkang@dongseo.ac.kr