예술로서의 컴퓨터 프로그래밍

도널드 크누스

1959년, *Communications of the ACM*이 창간되었을 때 ACM 편집위원회의 위원들은 이 정기간행물의 목적을 다음과 같이 서술하였습니다 [2]. "컴퓨터 프로그래밍이 컴퓨터 연구와 개발에 있어서 중요한 부분이 되려면, 프로그래밍은 아트에서 체계를 갖춘 학문으로 전환되어야 한다." 그러한 목표는 그 이후 몇 년 동안 계속 반복되는 주제였습니다. 예를 들어, 1970년에는 "프로그래밍 기법을 학문으로 변환하는 첫 걸음"이라는 기사도 있었습니다 [26]. 그러는 동안 우리는 우리 분야를 학문으로 만드는 데에 실제로 성공했는데, 그 방법은 너무도 쉬운 것이서, 단지 우리 분야를 "전산학"이라고 부르기로 결정한 것에 지나지 않았습니다.

이러한 의견들에는 "아트"로 분류되는 인간 활동 영역에는 바람직하지 않은 무언가가 있다는 생각이 내포되어 있습니다. 즉 아트가 어떤 실제적인 발전을 이루기도 전에 "학문"이 되어야 한다는 것입니다. 반면에, 나는 12년 이상 동안 "The *Art* of Computer Programming"이라는 일련의 책들을 집필해 오고 있습니다. 사람들은 나에게 그 책의 제목에 아트라는 단어를 사용했는지를 자주 묻습니다. 그리고 사실 어떤 사람들은 내가 그러한 제목을 선택한 것을 믿으려 하지도 않는 것 같습니다. 왜냐하면, 저의 그 책이 "The *Act* of Computer Programming"이라는 제목으로 참고 문헌으로 사용된 것을 본적이 있기 때문입니다.

이 강연에서 나는 왜 "아트"가 적절한 단어인지를 설명하려고 합니다. 어떤 것이 아트가 된다는 의미는 그것이 과학이 된다는 것과 대조할 때 어떤 의미를 갖는지를 논의할 것입니다. 그리고 아트라고 불리는 것들이 좋은 것인지 나쁜 것인지 설명할 것이고, 이 주제에 대해 적절한 관점을 갖는 것이 우리 모두가 지금 하고 있는 일을 질적으로 향상시키는 데 도움이 될 것임을 밝혀 볼 것입니다.

나의 책 제목에 대해 처음 질문을 받았던 것은 남부 캘리포니아에서 개최된, 지난 ACM 학회 기간이었던 1966년 이었습니다. 책이 아직 출판되기 전이었는데, 내 기억으로는 한 친구와 학회가 열린 호텔에서 점심을 먹고 있었습니다. 그 친구는 그 당시 내가 얼마나 나 자신을 내세우기를 좋아했는지를 알고 있었기 때문에, 그 책을 "An Introduction to Don Knuth"이라고 하는 것이 어떠냐고 물어봤습니다. 나는 그 반대로 그 친구의 이름을 따서 책 제목을 붙일 것이라고 했습니다: 그 친구 이름은 Art Evans 였습니다. (The Art of Computer Programming, 몸소 나타난 것이죠.)

이 이야기에서 우리는 "아트"라는 단어가 한 가지 이상의 뜻을 가지고 있다고 결론지을 수 있습니다. 사실, 이 단어의 가장 좋은 점은 그것이 여러 다른 의미로 쓰인다는 것이며, 그 각각은 컴퓨터 프로그래밍과 연관지어볼 때 아주 적절하다는 것입니다. 나는 이 강연을 준비하면서 도서관에 가서 수 년동안 "아트"라는 단어가 어떤 글에서 어떻게 사용되었는지를 찾아봤습니다. 그리고 쌓아 놓은 책 속에 파묻혀 매혹적인 나날을 보낸 뒤에 나는 "아트"라는 단어는 영어에서 가장 흥미로운 단어들 중 하나라는 결론에 이르게 되었습니다.

예술의 고전적 의미

라틴어 어원을 거슬러 가보면, "skill"이라는 뜻을 가진 *ars, artis* 를 찾게됩니

다. 그리고 여기에 해당하는 그리스 단어가 “technology”과 “technique”의 어원인 $\tau\acute{\epsilon}\chi\nu\eta$ 라는 것이 아마도 의미심장할 것입니다.

요즘에 누군가가 “아트”에 대해 이야기하면, 여러분은 가장 먼저 회화나 조각 같은 “미적 예술(미술, fine art)”을 생각하게 될 것입니다. 하지만, 20세기 이전에 이 단어는 상당히 다른 의미로 널리 사용되었습니다. “아트”의 예전 의미가 아직 여러 관용어구 속에 남아 있고, 특히 아트를 학문과 비교할 때 그러하므로, 앞으로의 몇 분 동안은 고전적인 의미의 아트에 대해 이야기해 보고자 합니다.

중세 시대 최초의 대학들은 다음 일곱 가지의 소위 “인문 교양 과목(liberal art)”들, 즉 문법, 수사, 논리, 대수, 기하, 음악, 천문을 가르치기 위해 설립되었습니다. 이것이 오늘날 인문 학부 교과과정과 상당히 다르다는 것과, 원래의 일곱 가지 중에서 적어도 세 가지는 전산학의 중요한 구성 요소라는 것에 주목하길 바랍니다. 그 당시의 “아트”는, 자연이나 인간의 본능으로 부터 나오는 활동과 대비되는 개념으로 인간의 지적 능력으로 만들어진 어떤 것을 의미했습니다. “교양” 과목들은 , 쟁기질과 같은 힘겨운 육체적 기술과 대비되는 개념에서 자유로왔습니다 ([6] 참고). 중세 시대 동안 “아트”라는 단어 그 자체는 일반적으로 논리를 의미했으며 [4], 그 논리는 삼단논법의 연구를 의미하는 것이었습니다.

학문 vs. 예술

“학문”이라는 단어는 여러 해 동안 “아트”과 거의 같은 의미로 사용되어 왔습니다. 예를 들어, 사람들은 일곱 가지 인문학들에 대해서도 말했는데, 그것은 앞에서 살펴본 일곱 가지 교양 과목들과 똑같은 것이었습니다 [1]. 13세기에 Duns Scotus는 논리학을 가리켜 “the Science of Sciences, and the Art of Arts”라고 했습니다 ([12, p. 34f] 참고). 문명이 발전하면서, 이 단어들은 점점 독자적인 의미를 갖게 되어서, “학문”은 지식을, “아트”는 지식의 적용을 나타내게 되었습니다. 그래서, 항해술은 천문학을 그 기반으로 하고 있습니다. 이러한 상황은 우리가 지금 “과학”과 “공학”을 구분하는 방식과 거의 정확히 같습니다.

19세기에 여러 저자들이 아트와 과학의 관계에 대해 글을 썼는데, 내가 아는 한 최상의 논의는 John Stuart Mill이 제시한 것이었습니다. 그는 1843년에 다음과 같이 말하였습니다 [28].

> 한 가지 아트의 토대를 형성하기 위해 여러 가지 학문들이 필요하기도 하다. 인간사의 복잡함은 이런 식이어서, 한 가지 일을 이루기 위해 여러 가지 일들의 본질과 특성을 아는 것이 필수적이기도 하다. ... 일반적으로 아트은 학문의 진리들로 이루어지며, 생각하는 데에 가장 적합한 순서가 아니라 실행하는 데에 가장 적합한 순서로 배열되어 이루어진다. 학문은 우리가 어떤 관점에 따라 우주의 일반적인 질서를 최대한 넓게 이해할 수 있도록 그 진리들을 통합하고 배열한다. 아트는 ... 서로 멀리 떨어져 있는 학문 분야의 각 부분들, 즉 실생활에서 긴급하게 요구되는 각 결과를 얻는 데 필요한 서로 다르고 이질적인 조건들의 생성에 관련된 진리들을 한 데 모은다.

“아트”의 의미에 대한 이런 내용들을 찾아보면서 나는 여러 저자들이 적어도 두 세기 동안 “아트”에서 학문으로 전환할 것을 요구해 왔다는 것을 알게 되었습니다. 예를 들어, 1784년에 쓰인 광물학 교과서의 서문에는 다음과 같은 글이 쓰여져

있습니다. [17]. “1780년 이전에는 광물학이 많은 사람들에게 아트 정도로는 이해되었어도 학문으로 간주되는 일은 거의 없었다.”

대부분의 사전들에 따르면 “학문”은 일반적인 “법칙”의 형태로 논리적으로 배열되고 체계화된 지식을 뜻합니다. 학문의 이점은, 그것을 통해 우리가 현상들을 개별 사례들에 대해 일일이 생각할 필요에서 벗어난다는 것입니다. 그래서 우리는 더 높은 수준의 개념들에만 신경을 써도 됩니다. John Ruskin이 1853년에 쓴 바와 같이 [32] “과학의 업적은 눈에 보이는 외관을 사실로, 인상을 실증으로 대체하는 것입니다.”

내가 찾아본 책의 저자들이 오늘날 글을 쓰게 된다면, 그들은 다음과 같이 특성을 구분하는 것에 동의할 것입니다: 학문은 우리가 잘 이해하고 있어서 컴퓨터에게도 가르칠 수 있는 지식이고, 우리가 어떤 것을 완전히 이해하지 못할 때 그것을 다루는 것이 아트이다. 해당 주제에 대한 우리의 지식의 깊이를 알고리듬이나 컴퓨터 프로그램이라는 개념을 통하여 아주 간편하게 검사할 수 있으므로, 아트에서 과학으로 가는 과정은 우리가 어떤 것을 자동화하는 방법을 알게 된다는 것과 같은 의미입니다.

인공 지능 분야가 그 동안 의미 있게 발전해 왔지만, 예측 가능한 미래에 컴퓨터가 할 수 있을 것과 보통 사람들이 할 수 있는 것 사이에는 여전히 큰 격차가 있습니다. 사람들이 말하고, 듣고, 창조할 때에, 물론 프로그래밍을 할 때에도 발휘되는 신비로운 통찰력은 아직도 과학의 범위 밖에 있습니다. 우리가 하는 거의 모든 것이 여전히 아트입니다.

이런 관점에서 컴퓨터 프로그래밍을 학문으로 만드는 것은 분명히 바람직한 것이며, 과연 우리는 오늘 강연 앞머리에 인용한 견해들이 발표된 이후 15년 동안 긴 길을 걸어 왔습니다. 15년 전에는 컴퓨터 프로그래밍에 대한 이해가 너무 부족해서, 프로그램의 정확성에 대한 증명을 생각조차 하지 못했습니다. 우리는 프로그램이 동작한다는 것을 알게 될 때까지 만지작거리기만 했습니다. 그 당시에 우리는 어떠한 엄격한 방법으로도 프로그램이 정확하다는 개념을 어떻게 표현할 것인지조차 알지 못했습니다. 최근 몇 년에야 프로그램을 만들고 이해하는 도구가 되는 추상화 과정에 대해 배우게 되었습니다. 그리고 프로그래밍에 대한 이 새로운 지식은 실제로 적용되면서 현재 크게 이익이 되고 있습니다. 비록 완전히 엄격하게 정확하다고 실제로 증명된 프로그램은 거의 없지만, 우리가 프로그램 구조의 원리들을 이해하기 시작하고 있기 때문입니다. 여기서 핵심은 오늘날 우리가 프로그램을 만들 때, 원한다면 언제든지 프로그램들의 정확성에 대해 형식을 갖춘 증명 방법을 원리적으로 구성할 수 있음을 안다는 것입니다. 그런 증명 방법이 어떻게 공식으로 표현되는지 우리가 이해하고 있기 때문입니다. 이러한 과학적 기반은, 직관이 정확성의 유일한 기반이었던 옛날에 우리가 만들었던 프로그램들보다 더욱 의미 있게 믿을만한 프로그램으로 귀결됩니다.

“자동화 프로그래밍”이라는 분야는 오늘날 인공 지능 연구의 주요 영역들 중 하나입니다. 그것을 지지하는 사람들은—프로그래밍이 지나간 시절의 유물일 뿐이라는 의미에서—“유물로서 컴퓨터 프로그래밍”이라는 제목의 강연을 하고 싶어 할 것입니다. 그들의 목표는 문제의 명세만 주어지면 우리보다 프로그램을 더 잘 만드는 기계를 창조하는 것이기 때문입니다. 개인적으로 나는 그런 목표가 완전히 달성될 것이라고는 생각하지 않지만, 그들의 연구는 정말로 아주 중요하다고

생각합니다. 우리가 프로그래밍에 대해 배우는 모든 것들이 우리의 예술성을 향상시키는 데 도움이 될 것이기 때문입니다. 이런 의미에서 우리는 모든 아트를 학문으로 전환하기 위해 계속 노력해야 할 것입니다. 그 과정에서 우리는 아트를 발전시키는 것입니다.

학문과 예술

이상의 논의에 따르면 컴퓨터 프로그래밍은 이제까지 학문인 동시에 아트이고, 이 두 가지 측면은 매우 상호보완적이라는 것을 알 수 있습니다. 분명 그 질문에 대해 고찰하는 저자들 대부분은 그들이 다루는 주제가 무엇이든 그것이 학문이면서 아트이라는 동일한 결론에 이르렀습니다 ([25] 참고). 1893년에 쓰인 기초 사진술에 대한 책을 발견했는데, 거기에는 이렇게 적혀 있었습니다. "사진을 현상하는 것은 아트이면서 학문이다" [13]. 사실, 내가 "아트"와 "학문"이라는 단어를 연구하기 위해 사전을 처음 집어 들었을 때, 우연히 편집자의 서문이 눈에 띄었는데, 이렇게 시작하고 있었습니다. "사전의 제작은 과학이면서 아트이다." Funk와 Wagnall의 사전 [27]은 단어들에 대한 자료를 공들여 축적하고 분류하는 것은 과학의 특성을 갖지만, 각 단어의 정의를 잘 선택하여 진술하는 것은 경제적이고 정확하게 글을 쓰는 능력이 요구된다고 말하고 있습니다. "아트가 없는 학문은 효과가 없을 것이며, 학이이 없는 아트은 부정확할 수밖에 없다."

이 강연을 준비하면서 나는 다른 사람들이 자신 책 제목에 "아트"와 "과학"을 어떻게 사용해 왔는지 알아보기 위해 스탠포드 도서관의 색인 카드를 조사했습니다. 그 결과는 상당히 흥미로왔습니다.

예를 들어, "*The Art of Playing the Piano*"이라는 제목의 책을 두 권 [5, 15] 찾았고, 다른 책들은 "*The Science of Pianoforte Technique*" [10], "*The Science of Pianoforte Practice*" [30] 등이었습니다. "*The Art of Piano Playing: A Scientific Approach*" 이라는 책도 있었습니다 [22].

그리고 "*The Gentle Art of Mathematics*"이라는 제목의 멋진 작은 책도 [31] 발견했습니다. 이 책은 나를 다소 슬프게 했는데, 나로서는 도저히 컴퓨터 프로그래밍을 "gentle art"라고 표현할 수 없을 것 같기 때문입니다.

나는 여러 해 전부터 "*The Art of Computation*"이라는 책을 알고 있었습니다. 이 책은 C. Frusher Howard라는 사람이 1879년 샌프란시스코에서 출판한 것입니다 [14]. 이 책은 실용적인 상업 계산에 대한 것으로 1890년까지 여러 판이 나오면서 40만 부 이상 팔렸습니다. 나는 그 서문을 재미있게 읽었습니다. 거기에서 하워드의 철학과 그 제목의 의도가 나와 상당히 다르다는 것을 볼 수 있었기 때문입니다. 그는 이렇게 썼습니다. "숫자의 학문에 대한 지식은 비교적 덜 중요하지만, 계산의 아트에 대한 기능은 절대적으로 필요하다."

몇 가지 책들은 제목에 과학과 아트을 모두 가지고 있습니다. 주목할 만한 것으로 "*The Science of Being and Art of Living*"이라는 Maharishi Mahesh Yogi의 책 [24]이 있습니다. "*The Art of Scientific Discovery*"[11]이라는 책도 있는데, 이 책은 위대한 과학의 발견들이 어떻게 이루어졌는지 분석하고 있습니다.

"아트"이라는 단어를 고전적인 의미로 사용하는 것이 매우 많습니다. 실제로 나의 책의 제목을 선택했을 때, 나는 아트을 이런 의미로 우선 생각하지는 않았고, 현재의 의미를 더 많이 생각하고 있었습니다. 아마도 이렇게 검색하면서 발견한

가장 흥미로운 것은 Robert E. Mueller가 비교적 최근에 쓴 "*The Science of Art*"라는 책이었습니다 [29]. 지금까지 언급한 모든 책들 중에서 뮐러의 책은 내가 오늘 연설의 중심 주제로 삼고자 하는 것을 가장 가깝게 표현하고 있습니다. 우리가 이제 이해하고 있는 바와 같은 실제 예술성의 측면에서 표현한 것입니다. 그는 이렇게 말합니다. "예술가의 상상적인 견해는 과학자에게는 죽음이라고 생각된 적이 있다. 그리고 과학의 논리는 아트적인 상상의 날개를 펴는 모든 가능성을 죽이는 저주처럼 보였다." 그는 이어서 과학과 아트을 종합함으로써 얻게 되는 이득에 대해 탐구하고 있습니다.

과학적인 접근 방법은 일반적으로 논리적, 체계적, 비인간적, 침착한, 이성적 등의 단어들로 특징짓는데 반해, 예술적인 접근 방법은 미적, 창의적, 인도주의적, 열망하는, 비이성적 등의 단어들로 특징짓습니다. 외견상 서로 모순된 이 두 가지 접근 방법 모두 컴퓨터 프로그래밍에 대하여 큰 가치가 있다고 생각됩니다.

Emma Lehmer는 1956년에 코딩이 "엄격한 학문이자 흥미로운 예술"인 것을 알게 되었다고 썼습니다 [23]. H. S. M. Coxeter는 1957년에 자신이 종종 "과학자보다는 예술가인 것처럼" 느낀다고 말했습니다 [7]. 이것은 C. P. Snow가 교육받은 사람들이 "두 문화"로 양극화되어 가는 것에 경고의 목소리를 내기 시작한 것과 때를 같이합니다 [34, 35]. 그는 우리가 진정한 진보를 이루고자 한다면 과학적이고 아트적인 가치를 결합할 필요가 있음을 지적합니다.

예술 작품

긴 강의를 들으면서 청중 속에 앉아 있을 때는 이 시간쯤 되면 집중력이 약해지기 시작하곤 합니다. 여러분도 "학문"과 "예술"에 대한 이 장황한 이야기에 조금 지쳐 가고 있지 않습니까? 그래도 이 강연의 나머지 부분을 주의하여 들어 주시기를 간절히 바랍니다. 이제 내가 가장 가슴 깊이 뼈저리게 느낀 부분이 나오기 때문입니다.

예술로서 컴퓨터 프로그래밍에 대해 이야기할 때 나는 우선 그것을 미학적인 의미에서 예술 형식으로 생각합니다. 교육자이자 저자로서 내 작품의 최우선 목표는 사람들이 *아름다운 프로그램*을 만드는 방법을 배우도록 돕는 것입니다. 최근에 내 책들이 실제로 코넬대학의 미술 도서관에 비치되었다는 것을 알게 되어 특별히 기뻤던 이유가 이것 때문입니다 [32]. (하지만, 그 세 권이 아무도 손을 대지 않은 채 책꽂이에 얌전하게 꽂혀 있는 것 같은데, 혹시 사서들이 내 책 제목을 문자 그대로 해석하는 실수를 한 것이 아닌가 걱정이 되기도 합니다.)

우리가 프로그램을 마련하는 것은 시나 음악을 창작하는 것과 같을 것입니다. Andrei Ershov가 말했듯이 [9] 프로그래밍은 우리에게 지적이면서 감성적인 만족을 줍니다. 그것은 복잡함을 지배하여 일관된 법칙들의 시스템을 구축하는 진정한 성취 결과이기 때문입니다.

더군다나 다른 사람들의 프로그램을 읽을 때 우리는 그 중에 진정한 예술 작품이 있음을 깨달을 수 있습니다. 나는 1958년 Stan Poley의 `SOAP II` 어셈블리 프로그램을 읽었을 때 느꼈던 그 엄청난 전율을 아직도 기억합니다. 여러분은 아마 내가 정신이 나갔다고 생각할 수도 있고, 그 이후로 스타일이 분명 많이 달라지기는 했지만, 그 때 그 프로그램은, 특히 그 당시 공부했던 다른 코드들에서 보였던 서툰 코딩과 비교해 볼 때, 시스템 프로그램이 얼마나 우아할 수 있는지

보여 주었다는 점에서 나에게는 큰 의미가 있습니다. 어셈블리 언어를 가지고도 아름다운 프로그램을 만들 수 있다는 사실은 내가 프로그래밍에 사로잡히게 된 첫 번째 이유입니다.

어떤 프로그램은 우아하고, 어떤 것은 절묘하고, 어떤 것은 찬란합니다. 나의 주장은 웅대한 프로그램, 숭고한 프로그램, 진실로 장엄한 프로그램을 만드는 것이 가능하다는 것입니다!

취향과 스타일

마침내 프로그래밍에서 스타일이라는 개념을 아야기 할 수 있게 되어서 기쁩니다. 나는 여러분 대부분이 Kernighan과 Plauger의 "*Elements of Programming Style*" 이라는 제목의 훌륭한 작은 책 [16]을 본 적이 있을 것이라고 믿고 싶습니다. 이와 연관되어 우리 모두가 중요하게 기억해야 할 것은 가장 좋은 한 가지 스타일이란 존재하지 않는다는 사실입니다. 모든 사람들은 자기가 선호하는 것들이 있으므로, 사람들을 어떤 부자연스러운 틀에 끼워 맞추려는 것은 잘못입니다. 우리는 종종 이런 말을 듣습니다. "나는 예술은 하나도 모르지만, 내가 뭘 좋아하는지는 안다." 중요한 것은 여러분이 지금 사용하고 있는 스타일을 정말로 좋아하고 있다는 사실입니다. 그것은 자신을 표현하기 위해 선택한 가장 좋은 방법일 것입니다.

Edsger Dijkstra는 그가 쓴 *Short Introduction to the Art of Programming* 의 서문에서 이 점을 강조했습니다 [8].

> 이 책의 목적은 좋은 취향과 스타일의 중요성에 대해 전달하는 것이다. (하지만) 제시된 스타일의 특정 요소들은 단지 "스타일"이라는 것에서 어떤 이점을 얻을 수 있는지 설명하기 위한 것이다. 이 점에서 나는 음악학교의 작곡 교사와 같은 부류의 사람이 된 것 같기도 하다. 그는 자기 학생들에게 특정한 교향곡을 작곡하는 방법을 가르치는 것이 아니다. 그는 학생들이 자신만의 스타일을 발견하도록 도와야 하며, 이것이 의미하는 바가 무엇인지 설명해야 한다. (나의 지금까지 "프로그래밍의 예술"에 대한 연설을 해 온 것도 여기에서 유추할 수 있다.)

이제 우리 자신에게 질문해야 합니다. 좋은 스타일은 무엇이고, 나쁜 스타일은 무엇인가? 다른 사람들의 작품을 비평할 때 이 점에 대해 너무 완고해져서는 안 됩니다. 19세기 초의 철학자 Jeremy Bentham은 이것에 대해 이렇게 말했습니다 [3, Bk. 3, Ch. 1].

> 우아함이나 취향에 대해 비평하는 사람들은 자신을 인류에 기여하는 사람이라고 생각한다. 하지만 실제로 그들은 인류의 즐거움을 방해하는 사람일 뿐이다. ... 언젠가 미래에 쓸모 있을 것들을 결합하여 실제로 즐거움이 생기게 하는 일들에 대한 취향이 아니라면, 원래 좋은 것이라고 할 만한 취향은 없다. 해로워지기 쉬운 일들에 대한 취향이 아니라면, 원래 나쁜 것이라고 특징지을 만한 취향도 없다.

우리에게 다른 사람의 취향을 "교정"할 특권이 있어서 그것을 행사한다면, 그 사람이 전적으로 누려야 하는 즐거움을 나도 모르는 사이에 부정하게 될 수도 있습니다. 내가 프로그래머들이 하고 있는 여러 가지 것들에 대해 비난하지 않는

것은 이 때문입니다. 비록 나 자신은 그들처럼 하는 것을 전혀 좋아하지 않지만 말입니다. 중요한 것은 그들이 스스로 아름답다고 느끼는 것을 만들고 있다는 사실입니다.

방금 인용한 글에서 Bentham은 다른 어떤 것들보다 더 훌륭한 미학의 원리들에 대해 우리에게 조언해 주고 있습니다. 즉, 결과의 "유용성"이 그것입니다. 우리는 아름다움에 대해 나름대로 기준을 세울 자유가 있지만, 우리가 아름답다고 생각하는 것을 다른 사람들도 유용하다고 생각한다면 특히 더 멋질 것입니다. 나는 컴퓨터 프로그램 만드는 것을 좋아한다고 고백할 수밖에 없습니다. 그리고 어떤 의미에서 가장 좋은 일을 하는 프로그램 만드는 것을 특히 더 좋아합니다.

물론 어떤 프로그램이 "좋은" 것이냐는 점에 대해서는 여러 의미가 있습니다. 첫째로, 정확하게 동작하는 프로그램이 특히 좋습니다. 둘째로, 어떤 환경에 맞춰야 할 때 고치는 것이 어렵지 않은 프로그램이 좋을 경우도 있습니다. 이 두 가지 목표는 거기에 쓰인 언어를 아는 사람이 그 프로그램을 쉽게 읽고 이해할 수 있을 때 달성됩니다.

어떤 완성된 프로그램을 좋다고 할 수 있는 또 다른 중요한 점은 사용자와 적절하게 상호작용하는 것입니다. 특히 자료 입력에서 사람의 실수를 복구할 때 그렇습니다. 의미 있는 오류 메시지를 작성하거나 오류가 생기지 않게 유연한 입력 형식을 설계하는 것은 진정한 아트입니다.

프로그램 품질의 또 다른 중요한 측면은 컴퓨터의 자원들이 실제로 사용되는 효율입니다. 유감스럽지만 오늘날 많은 사람들이 프로그램 효율에 대해 형편없다고 비난하고 있습니다. 이것은 우리가 지금 좋다는 것을 인정받을 수 있는 유일한 기준이 효율밖에 없는 시대에 살고 있으며, 과거의 프로그래머들이 효율에 너무 열중하여 필요 없이 복잡한 코드를 만드는 경향이 있었기 때문입니다. 이렇게 불필요한 복잡함의 결과로 디버그와 유지보수의 어려움 때문에 최종적인 효율은 낮아지고 말았습니다.

진짜 문제는 프로그래머들이 엉뚱한 시간과 엉뚱한 곳의 효율을 위해 너무 많은 시간을 낭비해 왔다는 것입니다. 미숙한 최적화는 프로그래밍에서 모든 (혹은 적어도 대부분의) 악의 근원입니다.

우리는 사소한 것은 따지면서 큰 것은 놓쳐서도 안 되겠고, 총 실행 시간과 공간의 이득이나 손실이 몇 퍼센트인지만 가지고 효율을 생각해서도 안 됩니다. 차를 살 때 많은 사람들은 50달러나 100달러의 가격 차이는 안중에도 없지만, 50센트짜리 품목을 겨우 25센트에 사기 위해 특정한 가게를 일부러 찾아가기도 합니다. 내가 말하고자 하는 바는, 효율이 필요한 시간과 공간이 따로 있다는 것입니다. 나는 구조적 프로그래밍에 대한 논문에서 효율성의 적절한 역할에 대해 논의했는데, 이것은 *Computing Surveys* 최근호에 실립니다 [21].

열악한 조건에서의 성취감

미학적인 만족에 관하여 알게된 다소 진기한 한 가지 사실은, 우리가 어떤 일을 한정된 도구만으로 성취했을 때 즐거움이 더욱 더 커진다는 것입니다. 예를 들어, 내가 개인적으로 가장 즐겁고 자랑스럽게 생각하는 프로그램은 메모리가 4096워드밖에 없던 초창기 미니컴퓨터에서 쓰기 위해 만들었던 컴파일러입니다. (여기서

1워드는 16비트입니다.) 이렇게 극심한 제한 속에서 무엇인가를 성취하는 것은 사람에게 거장이 된 듯한 느낌이 들게 합니다.

이와 비슷한 현상이 다른 상황에서도 벌어집니다. 예를 들어, 사람들이 자신의 폭스바겐에 반하는 일은 있어도, (아마도 훨씬 더 잘 달리는) 자신의 링컨 콘티넨털에 반하는 일은 거의 없습니다. 내가 프로그램을 배울 때, 펀치 카드 단 한 장에 들어가는 프로그램으로 최대한 많은 일을 하게 하는 놀이가 유행했습니다. APL 열광자들이 "한 줄짜리(one-liner)"를 즐기는 것도 이것과 같은 현상일 것입니다. 오늘날 프로그래밍을 가르칠 때, 미니컴퓨터를 "만져 보는" 경험을 할 수 있는 과정이 되어야 학생의 마음을 붙잡을 수 있다는 사실도 호기심을 끌기에 충분합니다. 화려한 운영 체제와 언어를 갖춘 우리의 대형 기계들이, 최소한 처음 볼 때는, 프로그래밍에 대한 애착이 생기게 하지 않는다는 것은 사실로 보입니다.

프로그래머들이 자신의 일에서 기쁨을 더 많이 누리기 위해 이 원리를 어떻게 적용할 것인지는 명확하지 않습니다. 분명히 프로그래머들은 새 기계가 예전 것보다 메모리가 절반밖에 되지 않을 것 같다는 상사의 갑작스러운 통보를 받으면 신음 소리를 낼 것입니다. 아무리 가장 헌신하는 "프로그래밍 아티스트"라도 그런 예상을 환영할 것이라고 생각되지 않습니다. 아무도 이유 없이 도구를 빼앗기는 것을 좋아하지 않을 것이기 때문입니다. 이와 다른 유형의 사례는 이 상황을 명료하게 하는 데 도움이 될 것입니다. 영화 제작자들은 1920년대에 유성 영화의 도입에 강하게 저항했습니다. 그들은 소리 없이도 말을 전달할 수 있었던 예전 방식에 대해 자부심을 가지고 있었기 때문입니다. 이와 유사하게, 진정한 프로그래밍 아트가는 더 강력한 장비의 도입에 분개할지 모릅니다. 오늘날의 대형 저장 장치들은 예전의 테이프 정렬 기법의 아름다움을 상당 부분 망쳐 놓는 경향이 있습니다. 하지만 오늘날의 영화 제작자들은 무성 영화로 되돌아가고 싶어 하지 않습니다. 그들이 게을러서가 아니라, 향상된 기술을 사용해도 아름다운 영화를 만드는 것이 가능하다는 것을 알고 있기 때문입니다. 아트의 형식은 변했지만, 아트성을 발휘할 여지는 여전히 많습니다.

그들은 어떻게 자신의 능력을 발전시켰을까요? 몇 년간의 뛰어난 영화 제작자들을 보면, 보통 비교적 원시적인 상황들 속에서, 그리고 종종 영화 산업의 규모가 한정된 다른 나라들에서 자신의 예술을 배워 온 것 같습니다. 최근 몇 년 동안 우리가 프로그래밍에 대해 배운 아주 중요한 것들도 초대형 컴퓨터들에 접근할 수 없었던 사람들에게서 나온 것으로 보입니다. 내가 보기에 이 이야기의 교훈은 우리가 교육을 할 때 자원이 제한되어 있다는 생각을 이용해야 한다는 것입니다. 우리는 모두 인위적인 제한 때문에 우리의 능력을 극한으로 밀고 가야 하게 될 때, 임시로 "장난감" 프로그램을 만들어 도움을 받습니다. 우리는 호사스러움에 파묻혀 살기만 해서는 안 됩니다. 그로 인해 둔감해지기 쉽기 때문입니다. 조그만 문제들에도 모든 능력을 다하여 달려드는 예술을 통해 우리는 실제 문제들을 해결하기 위한 재능을 갈고 닦을 수 있을 것이며, 그 경험으로 우리는 제한이 덜한 장비를 가지고 과제를 달성함으로써 더 큰 즐거움을 누리게 될 수 있을 것입니다.

이와 유사한 점에서, "예술 자체를 위한 예술"에 대해 뒷걸음질하지 말아야 합니다. 우리는 단지 재미를 위한 프로그램에 대해 죄의식을 느낄 필요가 없습니다. 나는 한 구문 으로 된 ALGOL 프로그램을 만들면서 무척 재미있어 한 적이 있는데, 그것은 내적(innerproduct) 대신에 m번째 소수를 계산하는 이상한 방식으로 내적

프로시저를 호출하는 것이었습니다 [19]. 몇 년 전에 스탠포드의 학생들이 자기 자체를 인쇄하는, 즉 프로그램의 출력 내용이 그 자체의 소스 텍스트와 동일한, 가장 짧은 FORTRAN 프로그램을 찾아내고 흥분했던 적이 있습니다. 이와 같은 문제를 다른 여러 언어들로도 궁리했습니다. 나는 그들이 여기에 매달렸던 것이 시간 낭비였다고 생각하지 않습니다. 앞에서 인용한 Jeremy Bentham도 그런 놀이의 "유용성"에 대해 부정하지 않을 것입니다 [3, Bk. 3, Ch. 1]. 그는 이렇게 썼습니다. "그 반대로, 유용성에 대해 이 이상 명백한 것은 없다. 유용하다는 특성이 즐거움의 근원인 어떤 것에서 기인한 것이 아니라면, 무엇에서 기인한 것이라는 말인가?"

아름다운 도구의 제공

현대 예술의 한 가지 특성은 창의성을 강조한다는 것입니다. 요즈음 많은 예술가들은 아름다운 것들을 창조하는 것에 대해 전혀 상관하지 않는 것처럼 보입니다. 참신한 생각들만 중요하게 여깁니다. 컴퓨터 프로그래밍이 이런 의미에서 현대 예술과 같아져야 한다고 권하지는 않겠지만, 확실히 이런 관찰을 통해 중요해 보이는 한 가지 생각에 이르게 되었습니다. 때때로 우리는 맥빠지게 지루한 프로그래밍 과제를 맡게 됩니다. 그런 과제는 창의성을 발휘할 여지가 전혀 없습니다. 그런 상황에서는 누군가가 나에게 와서 이렇게 말할 것 같습니다. "이래도 프로그래밍이 아름답습니까? 우아하고 매력적인 프로그램을 만드는 일을 즐겨야 한다는 선생님의 주장은 아주 그럴 듯하지만, 내가 어떻게 이런 쓰레기들로 부터 예술 작품으로 만들 수 있겠습니까?"

어쩌면, 사실입니다. 모든 프로그래밍 과제들이 재미있는 것은 아닙니다. 똑같은 식탁을 매일 닦아야 하는 "일상에 틀에 박힌 가정주부"를 생각해 보십시오. 거기에는 어떤 상황에서도 창의성이나 예술성의 여지가 없습니다. 하지만 그런 경우라 해도 크게 개선할 수 있는 방법이 있습니다. 우리가 아름다운 것을 가지고 일할 수 있다면, 판에 박힌 일을 하는 것도 즐거울 수 있습니다. 예를 들어, 매일 매일 부엌 식탁을 닦는 것도 정말로 즐길 수 있습니다. 그 식탁이 어떤 고급 원목으로 만들어졌고 아름답게 디자인되었다면 말입니다.

때때로 우리는 교향곡을 작곡하는 것 대신에 연주하라는 주문을 받습니다. 비록 작곡가에 의해서 작곡된 그대로 연주하느라 우리의 자유가 억압될 지라도 훌륭한 음악을 연주하는 것은 대단히 즐겁습니다. 프로그래머는 때때로 예술가가 되기 보다는 기능공이 되라는 요구를 받을 때가 있습니다. 그렇더라도 좋은 도구와 재료들이 주어진다면 그러한 기능공의 일 역시 매우 즐거울 것입니다.

그러므로 나는 시스템 프로그래머들에게, 그리고 우리 모두가 사용할 시스템을 만들어내는 기계 설계자들에게 내 생각을 이야기하면서 마무리하고자 합니다. 우리가, 특히 판에 박힌 일들을 할 때, 고생스럽게 사용할 수밖에 없는 도구보다는 사용하기가 즐거운 도구를 만들어 주십시오. 우리가 프로그램을 만들 때 우리를 더 즐겁게 하여 더 좋은 프로그램을 만들 수 있도록 북돋워 주는 도구를 만들어 주십시오.

내가 대학 입학생들에게 첫 번째로 말해야 하는 것이 "슬래시 슬래시 JOB은 아무개이다"를 펀치 카드에 찍는 방법일 때, 그들에게 프로그래밍이 아름답다고 납득시키는 것은 매우 어려울 것입니다. 작업 제어 언어라 해도 엄격하게 기능 위주로 하는 것보다 사용하기에 즐겁게 설계할 수 있을 것입니다.

컴퓨터 하드웨어 설계자들은 기계들을 더욱 더 사용하기에 즐겁게 만들 수 있습니다. 예를 들어 간단한 수학 법칙들을 만족하는 실수 계산을 제공하는 것 등이 그것입니다. 현재 대부분의 기계에서 이용할 수 있는 도구들로는 엄격한 오류 분석 작업이 맥빠질 정도로 어렵지만, 연산들을 적절하게 설계한다면 수치해석 전문가들이 정확성을 보증하는 더 좋은 서브루틴을 제공하는 데에 도움이 될 것입니다 ([20, p. 204] 참고).

소프트웨어 설계자들이 무엇을 할 수 있는지도 생각해 봅시다. 시스템 사용자의 사기를 높이는 가장 좋은 방법 중 하나는 사용자가 상호작용할 수 있는 루틴을 제공하는 것입니다. 우리는 시스템을 지나치게 자동화하여 동작들이 항상 막후에서만 일어나게 해서는 안 됩니다. 프로그래머인 사용자에게는 그들의 창의성을 유용한 방향으로 발휘할 수 있는 기회를 주어야 합니다. 모든 프로그래머들의 공통점 중 하나는 기계들을 가지고 일하는 것을 즐긴다는 것입니다. 그러므로 그들을 반복문 속으로 들어갈 수 있게 해 줍시다. 어떤 과제들은 기계가 잘 하지만, 다른 것들은 인간의 직관으로 더 잘 합니다. 적절히 설계된 시스템에서는 올바른 균형점을 찾을 수 있을 것입니다. (나는 여러 해 동안 방향을 잘못 잡은 자동화를 피하기 위해 노력해 왔습니다. [18] 참고)

프로그램 측정 도구들은 적절한 좋은 사례가 됩니다. 여러 해 동안 프로그래머들은 계산에 드는 실제 비용이 자기 프로그램의 어느 부분에 몰려 있는지 모르는 채로 있었습니다. 경험에 따르면 거의 모든 사람들이 자기 프로그램의 실제 병목 현상에 대해 엉뚱한 생각을 하고 있습니다. 프로그래머에게 그가 만든 코드의 각 줄에 대한 비용 내역이 전혀 주어지지 않은 상황에서, 효율을 위한 시도들이 종종 실패하고 마는 것은 놀랄 일이 아닙니다. 이런 일은, 의식주의 개별 항목들에 얼마나 많은 비용이 드는지 모르는 채로 균형 잡힌 예산을 계획해 보려는 신혼부부의 상황과 비슷합니다. 이제까지 우리가 프로그래머들에게 지급해 온 것은 최적화 컴파일러뿐입니다. 이 컴파일러는 해석하는 프로그램에 대해 뭔가 신비한 일을 하지만, 그것이 무엇인지는 전혀 설명하지 않습니다. 다행히도 마침내 이제 우리는 어떤 지식을 사용자가 당연히 가질 수 있게 해 주는 시스템의 출현을 보고 있습니다. 이 시스템은 프로그램의 계측과 실제 비용에 대한 적절한 반응을 자동으로 제공합니다. 이 실험적인 시스템은 상당히 큰 성공을 거둬 왔습니다. 이것을 통해 측정 가능한 개선 효과가 있기 때문이며, 특히 그것을 사용하는 것이 재미있기 때문입니다. 그래서 나는 이러한 시스템을 사용하는 것이 표준적인 운영 과정이 되는 것은 시간 문제일 뿐이라고 확신합니다. *Computing Survey*의 내 논문[21]에서 여기에 대해 좀 더 자세히 논의하고 있으며, 적절한 대화형 루틴으로 사용자인 프로그래머들의 만족을 높일 수 있는 다른 방법들에 대한 몇 가지 생각을 제시하고 있습니다.

언어 설계자들도 좋은 스타일을 유도하는 언어를 제공할 의무가 있습니다. 우리 모두는 스타일이 그것을 표현하는 데 사용되는 언어에 크게 영향을 받는다는 것을 알고 있기 때문입니다. 구조적 프로그래밍에 대한 현재의 격동적인 관심에 의해, 지금 사용하고 있는 언어들 중에는 프로그램과 자료 구조를 다루기에 정말 이상적인 것이 하나도 없지만, 이상적인 언어는 어떠해야 하는지도 명확하지 않다는 사실을 알게 되었습니다. 그러므로 나는 다음 몇 년 동안 언어 설계에 대한 면밀한 실험들이 많이 수행되기를 고대합니다.

요약

요약하면, 우리는 컴퓨터 프로그래밍이 예술이라는 것을 깨닫게 되었습니다. 그것이 축적된 지식을 이 세상에 적용하기 때문이며, 기술과 독창력을 요구하기 때문이며, 특히 아름다운 것들을 만들어내기 때문입니다. 어렴풋이나마 자신을 예술가라고 생각하는 프로그래머는 자기가 하는 일을 즐길 것이며, 그것을 더 잘 해낼 것입니다. 그래서 우리는 컴퓨터 학회에서 강연하는 사람들이 *예술의 최상의* 수준에 대해 이야기하는 것을 기뻐할 수 있습니다.

참고문헌

[1] Nathan Bailey, *Tile Universal Etymological English Dictionary* (London: T. Cos, 1727). See “Art,” “Liberal,” and “Science.”

[2] Walter F. Bauer, Mario L. Juncosa, and Alan J. Perlis, “ACM publication policies and plans,” *J. ACM* **6** (Apr. 1959), 121–122.

[3] Jeremy Bentham, *The Rationale of Reward*, translated from *Théorie des pehws et des récompenses*, 1811, by Richard Smith, (London: J. & H. L. Hunt, 1825).

[4] *The Century Dictionary and Cyclopedia* **1** (New York: The Century Co., 1889).

[5] Muzio Clementi, *The Art of Playing the Piano*, translated from *L'art de jouer le pianoforte* by Max Vogrich (New York: Schirmer, 1898).

[6] Sidney Colvin, “Art.” *Encyclopaedia Britannica*, eds 9, 11, 12, 13 (1875-1926).

[7] H. S. M. Coxeter, “*Convocation address*,” *Proc. 4th Canadian Math. Congress* (1957), 8–10.

[8] Edsger W. Dijkstra, *EWD316: A Short Introduction to the Art of Programming*, T. H. Eindhoven, The Netherlands, Aug. 1971.

[9] Andrei P. Ershov, “Aesthetics and the human factor in programming.” *Comm. ACM* **15**, 7 (July 1972), 501–505.

[10] Thomas Fielden, *The Science of Pianoforte Technique*. Macmillan, London, 1927.

[11] George Gore, *The Art of Scientific Discovery*. Longmans, Green, London, 1878.

[12] William Hamilton, *Lectures on Logic* **1** Win. Blackwood, Edinburgh, 1874.

[13] John A. Hodges, Elementary Photography: *The “Amateur Photographer” Library* **7** London, 1893. Sixth ed, revised and enlarged, 1907, p. 58.

[14] Howard, C. Frusher. Howard's *Art of Computation* and golden rule for equation of payments for schools, business colleges and self-culture ... C. F. Howard, San Francisco, 1879.

[15] Hummel, J.N. *The Art of Playing the Piano Forte*. Boosey, London, 1827.

[16] Kernighan B.W., and Plauger, P.J. *The Elements of Programming Style*. McGraw-Hill, New York, 1974.

[17] Kirwan, Richard. *Elements of Mineralogy*. Elmsly, London, 1784.

[18] Knuth, Donald E. *Minimizing drum latency time*. J. ACM 8 (Apr. 1961), 119–150.

[19] Knuth, Donald E., and Merner, J.N. ALGOL 60 confidential. *Comm. ACM* **4** (June 1961), 268–272.

[20] Knuth, Donald E. *Seminumerical Algorithms: The Art of Computer Programming* **2** Addison-Wesley, Reading, Mass., 1969.

[21] Knuth, Donald E. *Structured programming with go to statements*. Computing Surveys 6 (Dec. 1974), pages in makeup.

[22] Kochevitsky, George. *The Art of Piano Playing: A Scientific Approach*. Summy-Birchard, Evanston, II1., 1967.

[23] Lehmer, Emma. Number theory on the SWAC. *Proc. Syrup. Applied Math.* **6**, Amer. Math. Soc. (1956), 103–108.

[24] Mahesh Yogi, Maharishi. *The Science of Being and Art of Living*. Allen & Unwin, London, 1963.

[25] Malevinsky, Moses L. *The Science of Playwriting*. Brentano's, New York, 1925.

[26] Manna, Zohar, and Pnueli, Amir. Formalization of properties of functional programs. *J. ACM* **17** (July 1970), 555–569.

[27] Marckwardt, Albert H, Preface to *Funk and Wagnall's Standard College Dictionary*. Harcourt, Brace & World, New York, 1963, vii.

[28] Mill, John Stuart. *A System Of Logic, Ratiocinative and Inductive*. London, 1843. The quotations are from the introduction, §2, and from Book 6, Chap. 11 (12 in later editions), §5.

[29] Mueller, Robert E. *The Science of Art*. John Day, New York, 1967.

[30] Parsons, Albert Ross. *The Science of Pianoforte Practice*. Schirmer, New York, 1886.

[31] Pedoe, Daniel. *The Gentle Art of Mathematics*. English U. Press, London, 1953.

[32] Ruskin, John. *The Stones of Venice* **3**. London, 1853.

[33] Salton, G.A. Personal communication, June 21, 1974.

[34] Snow, C.P. "The two cultures." *The New Statesman and Nation* **52** (Oct. 6, 1956), 413–414.

[35] Snow, C.P. *The Two Cultures: and a Second Look*. Cambridge University Press, 1964.