

Python 변환을 지원하는, 한글 프로그래밍 언어 KoBASIC의 구현



저자 1 201624543 이석원
저자 2 202155574 유수민
저자 3 202043154 김정한

지도교수 우균

목 차

1. 서론	1
1.1 연구 배경	1
1.2 기존 문제점	1
1.3 연구 목표	2
2. 연구 배경	3
2.1 배경 지식	3
2.1.1 한베이식	3
2.1.2 스몰베이직 확장	4
2.1.3 새싹	5
2.1.4 Apple II BASIC	6
2.1.5 Python	7
3. 연구 내용	8
3.1 개발 일정 및 역할	8
3.1.1 개발 일정	8
3.1.2 구성원별 역할	9
3.2 KoBASIC의 설계 구조	9
3.3 GUI	10
3.4 한글 BASIC 해석기	12
3.5 Python 변환기	15
3.6 Python 변환 웹 인터페이스 구현	23
4. 연구 결과 분석 및 평가	24
4.1 테스트 케이스 분석	24

4.1.1 Execution Commands	24
4.1.2 Number Format	26
4.1.3 Print Format	27
4.1.4 Variable Names	29
4.1.5 IF...THEN	30
4.1.6 FOR...NEXT	32
4.1.7 Arrays	34
4.1.8 GOSUB...RETURN	37
4.1.9 READ...DATA...RESTORE	38
4.1.10 Real, Integer and String Variables	40
4.1.11 Strings	41
4.1.12 Evaluating Expressions	42
5. 멘토 의견서 반영 및 시연계획	42
5.1 멘토 의견서 반영	42
6. 결론 및 향후 연구 방향	44
7. 참고 문헌	45

1. 서론

1.1 연구 배경

현대 사회에서 컴퓨터 프로그램은 생활과 산업의 필수적인 도구로 자리 잡았다. 각종 소프트웨어와 응용 프로그램은 다양한 목적에 맞춰 활용되며, 이를 구현하는 프로그래밍 언어의 중요성도 함께 커지고 있다. 프로그램의 핵심 도구인 프로그래밍 언어는 컴퓨터와 사용자가 소통하는 수단으로, 현재 대부분의 프로그래밍 언어는 영어를 기반으로 설계되어 있다. 특히 주요 프로그래밍 언어로 사용되는 C, Python, Java, JavaScript와 같은 언어들은 영어 문법과 어휘를 바탕으로 작동한다.

이러한 환경은 영어를 모국어로 사용하는 사람들에게는 자연스럽지만, 영어에 익숙하지 않은 비영어권 사용자들, 특히 초보 학습자들에게는 큰 장벽으로 작용할 수 있다. 프로그래밍의 논리적 사고와 문법을 배우는 과정에서 영어로 된 명령어와 키워드를 이해해야 한다는 점은 학습 난도를 더욱 높이는 요인이 될 수 있다. 프로그래밍 학습 초기 단계에서, 언어적 어려움이 기술적 개념을 습득하는 데 걸림돌이 되기도 하며, 이는 비영어권 학습자들이 프로그래밍에 대한 흥미를 잃을 수도 있다.

특히 초등 및 중등 교육 과정에서 프로그래밍 교육이 확대되고 있는 시점에서, 영어 문법과 단어에 익숙하지 않은 학습자들에게 프로그래밍 언어는 접근하기 어려운 과목으로 다가온다. 따라서 비영어권 사용자들을 위한 대안적 프로그래밍 언어의 개발과 연구가 필수적으로 요구된다. 비영어권 학습자들이 자국어어를 통해 쉽게 프로그래밍을 배울 수 있는 환경을 제공함으로써, 프로그래밍의 진입 장벽을 낮추고 교육의 기회를 넓힐 수 있다[1].

1.2 기존 문제점

기존의 프로그래밍 언어는 대부분 영어를 기반으로 만들어져 있으며, 영어권 사용자들에게는 유리한 학습 환경을 제공한다. 그러나 비영어권 사용자들은 언어 장벽으로 인해 여러 가지 문제를 겪고 있다. 첫 번째 문제는 프로그래밍 언어의 명령어와 키워드가 모두 영어로 되어 있어, 초보자가 이들 명령어를 학습하는 데에 어려움을 느낀다는 점이다. 프로그래밍 초심자들에게는 새로운 개념과 구문을 배우는 과정 자체도 도전적이지만, 영어로 이루어진 문

법과 구문은 추가적인 부담을 준다. 특히 어린 학생이나 영어가 익숙하지 않은 학습자들에게 이러한 문제는 매우 큰 걸림돌이 될 수 있다.

두 번째로, 비영어권 사용자가 영어 문법을 이해하지 못하는 경우, 단순한 오류를 해결하는 데에도 어려움을 겪을 수 있다. 예를 들어, 프로그래밍에서 자주 발생하는 구문 오류(syntax error)나 타이포그래피 오류(typographical error)는 영어에 익숙하지 않은 사용자에게 큰 혼란을 초래할 수 있다. 또한, 영어 기반 문서를 통해 문제 해결 방법을 찾는 것도 학습자들에게 큰 스트레스로 다가올 수 있다. 결과적으로 이러한 언어적 장벽은 비영어권 학습자들이 프로그래밍 학습을 포기하게 하거나, 학습의 속도를 더디게 만든다.

세 번째로, 기존 프로그래밍 교육 환경에서는 영어로 된 교재와 온라인 자료가 대부분을 차지하고 있다. 영어 자료는 그 양이 방대하고, 기술적 내용을 깊이 있게 다루는 경우가 많지만, 영어에 익숙하지 않은 학습자들에게는 정보에 접근하는 것 자체가 매우 어렵다. 이에 따라 비영어권 학습자들은 영어에 의존해야 하는 상황에서 큰 불편을 겪게 된다. 이러한 문제점들은 프로그래밍 교육의 접근성을 제한하고, 다양한 학습자의 요구를 충족시키지 못하고 있다.

1.3 연구 목표

이러한 문제들을 해결하고자, 본 연구는 한글을 사용하는 초보자들이 쉽게 접근할 수 있는 프로그래밍 언어인 KoBASIC을 개발하는 것을 목표로 한다. KoBASIC은 기존의 교육용 프로그래밍 언어인 BASIC을 기반으로 하여, 영어 대신 한글 명령어와 키워드를 사용하는 프로그래밍 언어이다. 이를 통해 학습자들은 영어로 된 프로그래밍 문법에 대한 부담을 덜고, 자국어인 한글로 프로그래밍 개념을 이해할 수 있다.

KoBASIC은 기본적으로 초보자들이 쉽게 이해할 수 있는 간단한 문법과 구조를 갖추고 있어, 처음 프로그래밍을 배우는 학습자들에게 적합하다[2]. 한글로 이루어진 명령어와 구문은 학습자가 프로그램의 동작 방식을 더욱 직관적으로 이해할 수 있도록 도와준다. 이는 프로그래밍 학습 과정에서 비영어권 학습자들이 언어 장벽을 느끼지 않고, 개념 자체에 집중할 수 있는 환경을 제공한다.

본 연구의 궁극적인 목표는 KoBASIC을 통해 프로그래밍 교육의 접근성을 높이는 것이다. 한글 기반 프로그래밍 언어를 제공함으로써, 한국어 사용자들이 프로그래밍을 배우고 활용하는 데 있어 언어적인 어려움을 극복하고, 더욱 효율적으로 학습할 수 있도록 돕는 것이다. 또한, 이 언어를 활용한 교육 프로그램을 개발함으로써, 프로그래밍 교육이 보다 널리 퍼지고, 다양한 학습자가 쉽게 프로그래밍의 기초를 익힐 기회를 제공하는 것이 연구의 주요 목표이다.

```

10 INPUT "Please enter your name", A $
20 PRINT "Good day", A $
30 INPUT "How many stars do you want?"; S
35 S $ = ""
40 FOR I = 1 TO S
50 S $ = S $ + "*"
55 NEXT I
60 PRINT S $
70 INPUT "Do you want more stars?"; Q $
80 IF LEN (Q $) = 0 THEN GOTO 70
90 L $ = LEFT $ (Q $, 1)
100 IF 30 (L $ = "Y") OR (L $ = "y") THEN GOTO
110 PRINT "Goodbye";
120 FOR I = 1 TO 200
130 PRINT A $; "";
140 NEXT I
150 PRINT

```

그림 1. BASIC 예시

10	반복 가 = 1..3 목 1
20	조건 가 = 1 참이면 순회 200
30	순회 100
40	다음 가
50	끝
100	출력 " "
200	출력 " +-----+ "
300	복귀

그림 2. KoBASIC 예시

2. 연구 배경

2.1 배경 지식

2.1.1 한베이식

한베이식(HanBASIC)[3]은 1980년대 고려대학교에서 개발된 한국어 기반 프로그래밍 언어로, 영어에 익숙하지 않은 한국어 사용자들이 프로그래밍을 쉽게 배울 수 있도록 고안된 언어이다. 이는 당시 컴퓨터 교육의 초기 단계에서 프로그래밍의 언어적 장벽을 낮추기 위한 중요한 시도였으며, 프로그래밍을 처음 접하는 초등학생과 중학생을 대상으로 설계되었다.

1980년대는 정보화 시대의 도래와 함께 컴퓨터 교육의 필요성이 급격히 증가하던 시기였다. 그러나 대부분의 컴퓨터 프로그램과 프로그래밍 언어는 영어로 되어 있었기 때문에, 영어권이 아닌 사용자들은 프로그래밍을 배우는 데 많은 어려움을 겪고 있었다. 당시 한국에서도 컴퓨터 교육을 위한 프로그래밍 언어의 필요성이 대두되었지만, 영어에 대한 부담은 학습의 큰 장애물로 작용했다. 이러한 문제를 해결하기 위해 한글을 기반으로 한 프로그래밍 언어를

개발하는 것이 중요한 과제로 떠올랐다.

한베이식은 이 같은 필요성을 충족시키기 위해 고려대학교에서 개발된 언어로, 한국어 사용자들이 자신의 모국어로 프로그래밍을 학습할 수 있도록 한글 명령어와 한글 오류 메시지를 제공함으로써, 프로그래밍 학습의 진입 장벽을 낮추는 것을 목표로 했다. 이는 한국어를 모국어로 사용하는 사용자들에게 언어적인 부담 없이 프로그래밍에 입문할 기회를 제공했다.

한베이식의 가장 큰 특징은 모든 명령어와 오류 메시지가 한글로 제공된다는 점이다. 예를 들어, 영어 기반의 프로그래밍 언어에서 흔히 사용되는 PRINT 명령어는 한베이식에서 '출력'으로 표현되었다. 이를 통해, 한국어 사용자들은 영어로 된 프로그래밍 용어를 배우지 않고도 프로그래밍 언어의 기본 개념을 쉽게 습득할 수 있었다. 한베이식은 교육용 언어로 설계되었으며, 특히 초등학교 저학년 학생들을 대상으로 쉽게 이해하고 사용할 수 있도록 단순화된 문법과 구조를 갖추었다. 이러한 특성 덕분에 컴퓨터를 처음 접하는 어린 학생들에게도 적합한 학습 도구가 되었다. 한베이식은 한국어를 사용하는 사용자들이 모국어로 프로그래밍을 배울 수 있도록 한 최초의 시도 중 하나로, 한국어 기반 프로그래밍 언어의 개발 역사에서 중요한 의미를 지닌다.

2.1.2 스몰베이직 확장

스몰베이직[4]은 Microsoft에서 개발한 초보자용 프로그래밍 언어로, 간단한 문법과 직관적인 구조를 제공하여 프로그래밍 입문자들이 쉽게 학습할 수 있는 환경을 제공한다. 주로 Windows 환경에서 사용되며, 초등학생이나 프로그래밍을 처음 접하는 사용자들이 코딩의 기본 개념을 익히기에 적합한 도구로 설계되었다.

스몰베이직은 복잡한 구조나 개념 없이 간단한 명령어와 기본적인 프로그래밍 문법을 사용한다. 이를 통해 초보자들이 혼란 없이 프로그래밍의 기초를 배울 수 있다. 사용자는 스몰베이직에서 간단한 코드를 통해 그래픽 요소를 화면에 그리거나 게임과 같은 간단한 프로그램을 구현할 수 있다. 이러한 시각적 출력은 학습자의 흥미를 끌고, 결과를 빠르게 확인할 수 있는 장점을 제공한다. 스몰베이직은 주로 Windows에서 실행되며, Microsoft의 다양한 개발 도구 및 라이브러리와 호환성을 제공한다.

이 논문에서는 기존에 Windows에서만 실행되던 스몰베이직을 확장하여 Mac, Linux, 모바일 플랫폼에서도 실행할 수 있도록 개선한 점을 다룬다. 이는 프로그래밍 교육의 접근성을 높이기 위한 중요한 시도로, 다양한 운영체제에서 스몰베이직을 사용할 수 있게 함으로써 더 많은 사용자가 다양한 환경에서 쉽게 프로그래밍을 배울 수 있도록 한다. 특히, 모바일 플랫폼에서의 사용은 접근성을 극대화하는 중요한 요소로 작용한다.

그러나, 스몰베이직은 영어 기반의 프로그래밍 언어로 제공되기 때문에 비영어권 사용자들에게 여전히 언어적인 장벽이 존재한다. 명령어와 오류 메시지가 영어로 되어 있어, 영어에 익숙하지 않은 사용자들에게는 프로그램의 이해와 활용이 어려울 수 있다.

2.1.3 새싹

새싹[5]은 부산대학교에서 개발된 한글 기반 프로그래밍 언어로, C 언어와 유사한 문법을 제공하면서도 한국어 어순에 맞춘 프로그래밍 환경을 지원한다. 새싹은 한국어 사용자가 자연스러운 한글 문장을 통해 프로그래밍을 학습할 수 있도록 고안되었으며, 프로그래밍을 처음 접하는 초보자도 쉽게 접근할 수 있도록 설계되었다.

새싹은 한글 어순에 맞춰 프로그래밍할 수 있도록 설계되었다. 이는 사용자들이 프로그래밍 명령어를 작성할 때 자연스러운 한국어 문장 구조로 코드를 작성할 수 있게 하여, 프로그래밍을 처음 배우는 학습자들이 문법적 부담 없이 쉽게 프로그래밍 개념을 익힐 수 있도록 돕는다. 이는 영어 기반의 C 언어와 같은 언어를 배울 때 겪는 언어적 장벽을 줄여준다. 새싹은 C 언어를 기반으로 한 문법을 제공한다. C 언어는 범용 프로그래밍 언어로 널리 사용되며, 이를 기반으로 한 새싹은 프로그래밍의 기본 개념과 구조를 깊이 있게 이해할 수 있는 기회를 제공한다. 그러나 C 언어 특유의 복잡한 문법은 초보자에게 다소 어려울 수 있으며, 이로 인해 새싹도 C 언어의 난해한 부분을 내포할 가능성이 있다. 새싹은 초보자를 위한 한글 프로그래밍 언어로, 한국어를 모국어로 사용하는 사용자들이 보다 쉽게 프로그래밍을 접할 수 있도록 설계되었다. 프로그래밍을 처음 배우는 학습자들이 C 언어와 같은 고급 언어의 기본 개념을 학습하기 전, 자연스럽게 프로그래밍의 기초를 다질 수 있도록 돕는

언어이다.

새싹은 C 언어를 기반으로 하고 있기 때문에, C 언어 특유의 복잡한 구조와 문법으로 인해 초보자가 완전히 이해하기에는 어려움이 있을 수 있다. 특히, 포인터와 같은 저수준 프로그래밍 개념은 프로그래밍 경험이 없는 학습자들에게 부담이 될 수 있다. 따라서 새싹은 초보자들이 프로그래밍을 시작할 때 기본 개념을 익히기 좋은 도구이지만, C 언어의 복잡성으로 인해 완전한 초보자에게는 다소 어렵게 느껴질 수 있는 부분도 존재한다.

2.1.4 AppleII BASIC

AppleII BASIC[6]은 AppleII 컴퓨터에 내장된 BASIC(Beginner's All-purpose Symbolic Instruction Code) 프로그래밍 언어로, Apple이 1970년대 후반에 출시한 개인용 컴퓨터 AppleII에서 주로 사용되었다. AppleII BASIC은 당시 개인용 컴퓨터 사용자들이 프로그래밍을 배우고 프로그램을 개발하는 데 사용된 주요 도구 중 하나였다.

AppleII BASIC은 Microsoft BASIC을 기반으로 하여, 기능과 성능이 개선된 버전이다. Microsoft BASIC은 1970년대에 다양한 컴퓨터 시스템에서 광범위하게 사용된 언어로, AppleII에서도 이를 기반으로 프로그래밍이 이루어졌다. AppleII BASIC은 수학적 연산, 문자열 처리, 그래픽 기능 등 다양한 기능을 제공했으며, 사용자들이 AppleII 컴퓨터에서 프로그램을 개발하는 데 주로 사용되었다. AppleII BASIC은 AppleII 컴퓨터에 기본 내장되어 있었기 때문에, 사용자가 별도의 설치 과정 없이 AppleII를 켜고 바로 프로그래밍을 시작할 수 있었다. 이는 당시 컴퓨터 사용자들이 쉽게 프로그램을 작성하고 실행할 수 있도록 도와주었고, 프로그래밍 학습을 위한 대중적인 도구로 자리 잡게 되었다. BASIC은 본래 프로그래밍 입문자를 위한 언어로 설계되었기 때문에, AppleII BASIC 역시 간단한 문법과 직관적인 사용법을 제공하였다. 프로그래밍에 처음 입문하는 사람들도 쉽게 배우고 사용할 수 있었으며, 이는 개인용 컴퓨터가 대중화되면서 많은 사용자들에게 유용한 학습 도구로 활용되었다.

이 논문에서는 AppleII BASIC을 기반으로 KoBASIC을 개발하였다. KoBASIC은 AppleII BASIC의 철학을 계승하면서도, 한글 명령어와 단순한 문법을 제공하여 한국어 사용자들이 쉽게 접근할 수 있는 프로그래밍 언어로 개발되었

다. AppleII BASIC이 프로그래밍 입문자에게 친숙한 환경을 제공한 것처럼, KoBASIC 역시 한국어 기반 프로그래밍 언어로서 초보자들이 프로그래밍을 배우는 데 있어서 언어적 장벽을 줄이고, 기본적인 개념을 쉽게 이해할 수 있도록 돕는다.

2.1.5 Python

Python[7]은 1991년에 개발된 인터프리터 기반 프로그래밍 언어로, 그 간결하고 읽기 쉬운 문법 덕분에 많은 개발자와 학습자들 사이에서 인기를 얻고 있다. Python은 다양한 분야에서 활용될 수 있는 다목적 언어로, 웹 개발, 데이터 분석, 인공지능, 머신러닝, 과학 컴퓨팅 등 광범위한 응용 프로그램을 지원한다.

Python의 문법은 자연어에 가까워서 코드의 가독성이 뛰어나며, 이는 특히 프로그래밍 입문자들이 쉽게 이해하고 배울 수 있도록 돕는다. 명령어와 구문이 간단하여 복잡한 문법을 배우지 않고도 기본적인 프로그래밍 개념을 빠르게 익힐 수 있다. Python은 다양한 라이브러리와 프레임워크를 제공하여, 개발자들이 특정 분야의 요구사항을 충족시키기 위해 필요한 기능을 손쉽게 구현할 수 있게 돕는다. 예를 들어, NumPy와 Pandas는 데이터 분석에, Django와 Flask는 웹 개발에 많이 사용된다. 이러한 풍부한 자원들은 Python을 다양한 프로젝트에 활용할 수 있는 유연성을 제공한다. 또, Python은 인터프리터 언어로, 코드가 실행될 때 즉시 해석되어 실행된다. 이는 개발자가 코드를 작성한 후 즉시 결과를 확인할 수 있게 해 주며, 빠른 프로토타이핑과 테스트를 가능하게 한다. 또한, 코드 수정 후 다시 컴파일할 필요 없이 실행할 수 있어 개발 과정이 더 효율적이다.

Python은 프로그래밍 교육의 표준 언어로 자리잡고 있으며, 많은 교육 기관과 온라인 학습 플랫폼에서 기본적인 프로그래밍 교육을 위해 Python을 채택하고 있다. 그 이유는 Python이 제공하는 간단한 문법 덕분에 초보자들이 코딩의 기본 원리를 쉽게 배울 수 있기 때문이다. 또한, Python은 코딩 교육을 위한 다양한 자료와 커뮤니티가 활성화되어 있어 학습자들이 자주 활용할 수 있는 리소스가 풍부하다.

3. 연구 내용

3.1 개발 일정 및 역할

3.1.1 개발 일정

BASIC 한글 지원 구현 작업은 성공적으로 완료되었으며, 한글 키워드 및 함수명을 추가하여 한국어로 프로그래밍할 수 있는 환경을 조성했다. 이후 Unicode 문자열 처리 기능 구현 작업도 완료되었다. KoBASIC 구현 작업은 예상보다 잘 진행되어 BASIC 문법을 기반으로 한 KoBASIC 언어의 기본 문법과 구조를 설계하고 구현하는 데 성공했다. KoBASIC 언어 함수의 Python 코드 변환 기능 구현 작업도 성공적으로 진행했다. Application 개발 및 배포 작업에서는 사용자 친화적인 GUI 인터페이스를 개발하였으며, 라인 번호 표시, 구문 강조, 자동 완성 등의 편의 기능을 추가하여 사용자가 편리하게 코드를 작성할 수 있도록 하였다. 또한, 라인 번호 리넘버링 기능을 추가하여 코드를 체계적이고 정돈된 방식으로 표시할 수 있게 하였다. KoBASIC과 Python 변환 기능의 품질 향상을 위해 데이터 학습을 진행했고, 사용자 피드백을 반영하여 인터페이스와 기능을 지속해서 개선하였다. 표 1은 진행된 추진 일정표이다.

작업 번호	개발 구분	5월				6월				7월				8월			
		1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
1	BASIC 한글 지원 구현																
2	Unicode 문자열 처리 기능 구현																
3	KoBASIC 구현																
4	KoBASIC 언어 함수 Python 코드 변환 함수 구현																
5	Application 개발 및 배포																
6	오류 구문 추가																

표 1. 상세 개발 일정

3.1.2 구성원별 역할

이 름	구성원별 역할
이석원	<ul style="list-style-type: none"> - BASIC 한글 지원 구현: BASIC 언어의 한글 지원을 위한 핵심 기능 개발 - 한글 변수명과 함수명 처리: Unicode 문자열 처리 기능을 활용하여 한글 변수명과 함수명이 올바르게 처리될 수 있도록 개발 - 한글 입출력 개발: 키보드 입력과 출력을 위한 문자 인코딩 문제를 고려하여 한글 입출력 기능을 구현
유수민	<ul style="list-style-type: none"> - KoBASIC 문법 및 함수 구현: KoBASIC 언어의 문법 규칙을 정의하고, 주요 함수들을 구현 - 오류 문구 테스트 및 검사: 오류 발생 시 사용자에게 적절한 피드백을 제공하기 위한 오류 메시지 테스트 및 검증 - 모델 테스트 및 분석: 구현된 KoBASIC의 기능을 테스트하고 분석하여 성능 및 오류를 확인
김정환	<ul style="list-style-type: none"> - KoBASIC 언어 함수 Python 코드 변환 함수 구현: KoBASIC 언어에서 Python 코드로의 변환을 위한 함수 개발 - Python 변환 웹 인터페이스 구현: KoBASIC을 Python 코드로 변환하는 웹 인터페이스를 설계하고 개발

표 2. 구성원별 상세 역할

3.2 KoBASIC의 설계 구조

지금까지 수행한 과제의 큰 틀은 다음과 같다. 우선 사용자를 통해서 코드를 입력받으면 한글 BASIC 해석기를 통해 코드를 해석한다. 이후 추가로 해당 코드를 Python 변환기를 통해 Python 코드로 변환하여 해당 코드를 출력하는 과정을 가진다.

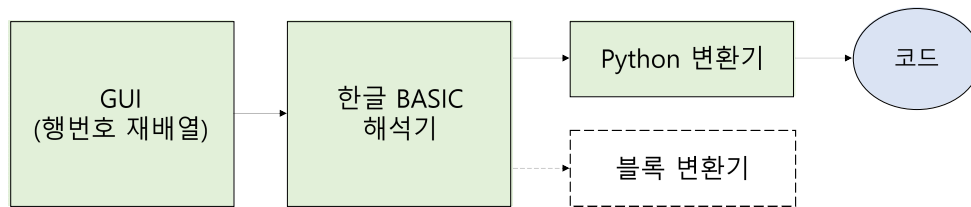


그림 3. KoBASIC 실행 구조

3.3 GUI

KoBASIC의 한글 지원은 C++과 Qt의 기능을 이용하여 구현한다. 한글 변수명과 함수명을 처리하기 위해 Unicode 문자열 처리 기능을 사용하였으며, 키보드 입력과 출력을 위한 문자 인코딩 문제를 고려하여 한글 입출력을 지원하도록 개발하였다. 또한, 한글 주석 기능을 구현하여 코드의 가독성을 높이도록 하였다. 아래는 KoBASIC으로 작성한 간단한 정렬 프로그램과 그 결과를 나타낸 것이다.

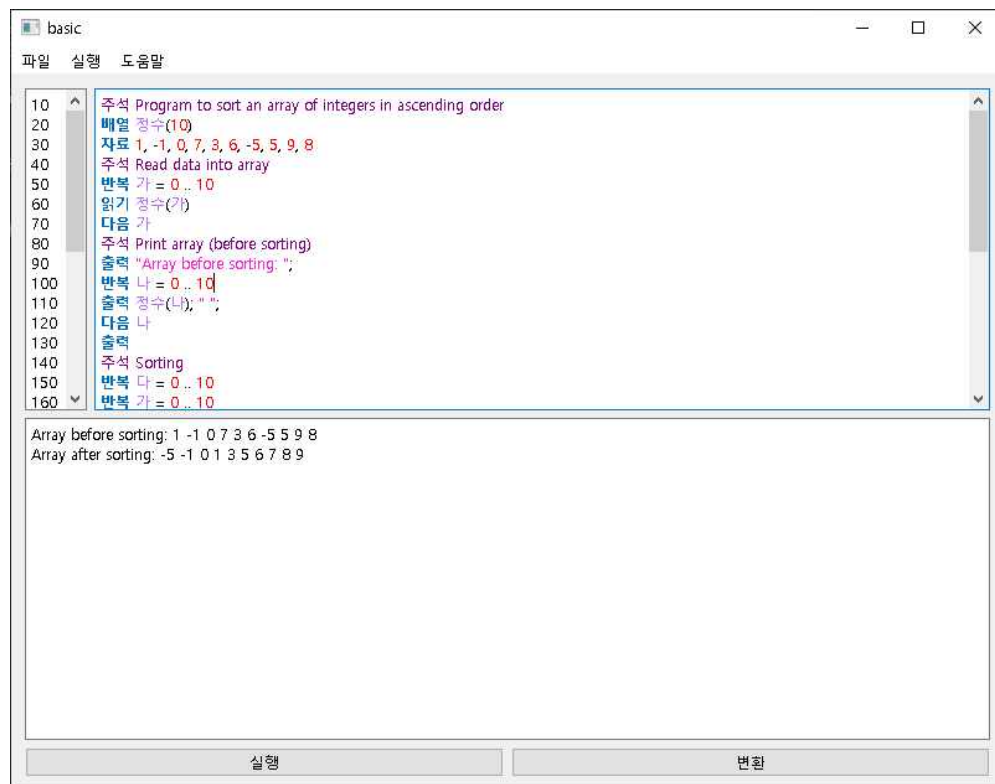


그림 4. KoBASIC을 이용한 정렬 예제

KoBASIC UI 상단에는 코드 작성 부분이 있으며, 하단에는 출력 부분이 있다. 코드 작성은 한글로 할 수 있으며, 코드의 가독성을 위해 문법 강조 기능을 적용하였다. 예를 들어, "PRINT"를 "출력"으로, "IF"를 "조건"으로 사용할 수 있게 하였다.

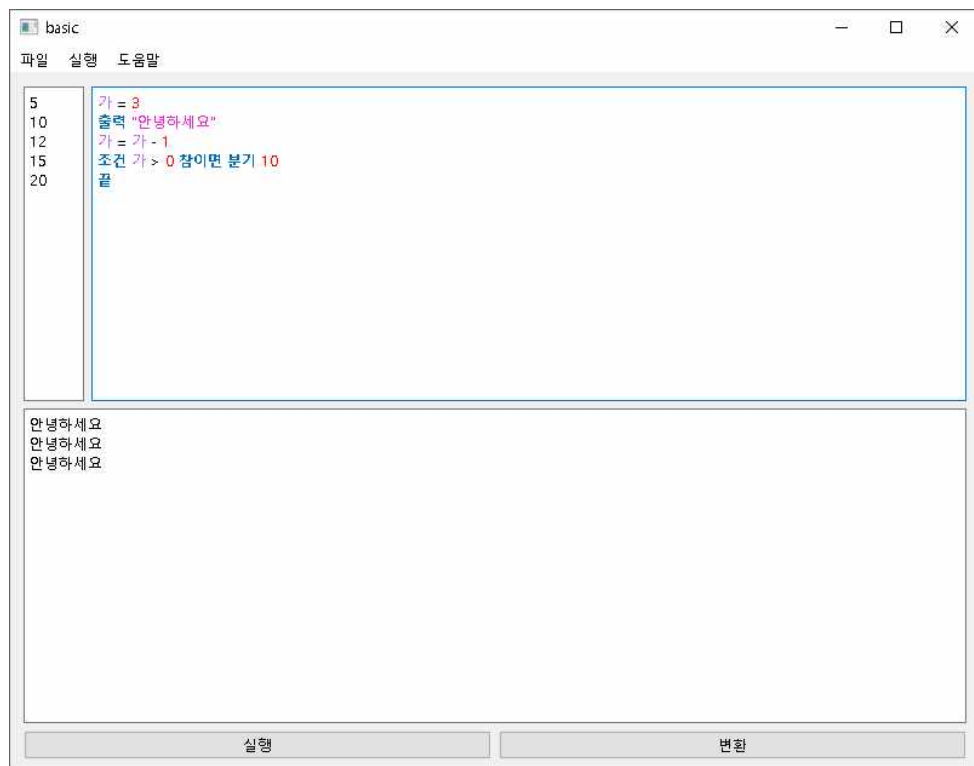


그림 5. 한글 키워드 명령어

또 라인 번호를 따로 분리하여서 리넘버링 하고자 하면 순서대로 정렬해주는 기능을 추가로 구현하였다. 이를 통해 라인 번호를 10단위로 정렬하여 코드를 더 체계적이고 정돈된 방식으로 표시할 수 있도록 하였다. 코드 작성 후 실행 버튼을 누르면 실행 결과가 아래에 출력된다.



그림 6. Re-numbering 전



그림 7. Re-numbering 후

Qt의 구현 중 어려움이 있었던 부분은 다음과 같다. KoBASIC에서 제공하는 함수 중 기존의 Input 함수는 Qt에서 사용될 때 블로킹이 발생하는 문제를 가지고 있다. 이 블로킹 문제는 프로그램이 사용자 입력을 기다리는 동안 UI가 멈추거나 응답하지 않게 되는 상황을 일으킨다. 이는 Qt의 이벤트 루프가 입력 대기 동안 차단되기 때문에 발생하며, 결과적으로 전체 애플리케이션의 반응성이 떨어진다. 이러한 문제는 특히 GUI 애플리케이션에서 치명적일 수 있으며, 사용자 경험을 크게 해칠 수 있으므로 수정이 필요하다.

위 문제의 원인을 살펴보면 다음과 같다. Input 함수는 기본적으로 프로그램이 사용자의 입력을 받을 때까지 대기하는 구조이다. 이 대기 과정에서 프로그램의 실행이 멈추고, 이벤트 루프가 차단되기 때문에 GUI가 응답하지 않게 된다. Qt는 이벤트 기반 프레임워크로, UI의 반응성을 유지하려면 이벤트 루프가 지속적으로 실행되어야 한다. 하지만, Input 함수가 실행될 때 프로그램의 흐름이 멈추고 이벤트 루프도 함께 차단된다. 따라서, 사용자는 입력 대기 중에 GUI가 멈추는 문제가 발생한다고 볼 수 있다. 이 문제를 해결하기 위해 Qt의 비동기 처리 기능을 활용하여 Input 함수가 호출될 때도 UI가 계속 반응할 수 있도록 개선할 필요가 있다.

3.4 한글 BASIC 해석기

KoBASIC은 BASIC의 문법은 유지하되, 키워드와 변수명 등을 한글로 제공하

는 것을 목적으로 한다. 한글 키워드를 사용하면 한글을 모태로 하는 사용자가 자연스럽게 프로그래밍할 수 있다. 또한, 주석 등을 한글로 작성할 수 있게 하여 코드의 이해와 설명을 돕는다. 아래 그림은 KoBASIC의 실행 구조를 자세히 나타낸다.

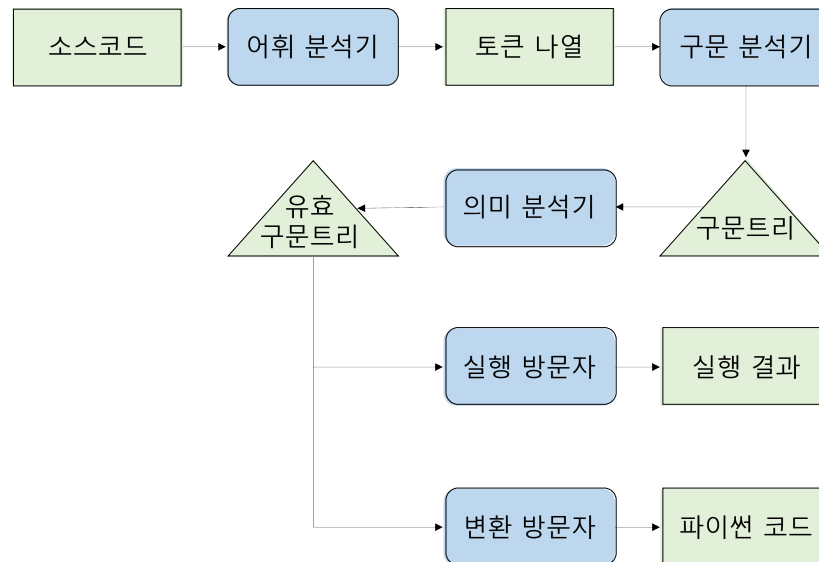


그림 8. KoBASIC 실행 구조

KoBASIC의 구현도 전통적인 컴파일러 및 인터프리터 구현 과정을 따른다. 어휘분석 단계에서는 소스 코드를 토큰으로 분해한다. 토큰은 코드의 기본 구성 요소로, 변수, 연산자, 예약어 등과 같은 언어 요소를 나타낸다. 구문분석은 어휘분석에서 생성된 토큰들의 구조와 관계를 파악한다. 이 단계에서는 문법 규칙을 준수하는지 확인하고, 추상 구문 트리를 생성한다.

son과 bro의 관계는 이 구문 트리의 노드들 간의 관계를 나타내는 데 사용된다. 각각의 노드는 다른 노드와 연결될 수 있으며, 이 관계는 특정 트리 구조를 통해 프로그램의 구문을 나타낸다. 구체적으로 son은 현재 노드의 자식 노드를 가리킨다. 즉, 하위 구조를 나타내는 노드를 참조한다. 예를 들어, 어떤 구문 요소의 하위 항목들이 있을 때, 이를 son을 통해 연결할 수 있다. bro는 현재 노드의 형제 노드를 가리킨다. 같은 레벨에 있는 다른 구문 요소들을 참조한다. 예를 들어, 여러 개의 명령어가 같은 레벨에 나열되어 있을 때 이들을 bro를 통해 연결한다. 이 구조를 통해 프로그램의 구문을 트리 형태로 표현할

수 있으며, 이를 바탕으로 프로그램의 의미를 분석하고 실행할 수 있는 구조를 구현할 수 있다. 구문 트리의 생성과 탐색 과정에서 이러한 연결 관계가 중요한 역할을 하게 된다. 아래의 예시를 트리 형태로 표현하면 다음과 같다.

```
10 print "hello"
20 if n > 0 then goto 100
```

그림 9. 트리 구조 예시를 위한 코드

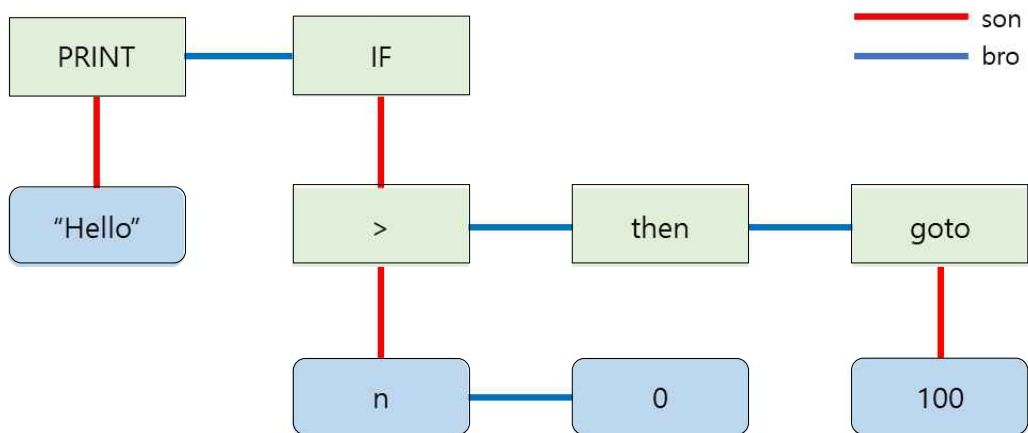



그림 10. 위 예시의 트리 구조

의미분석은 프로그램의 의미를 이해하고 검증하는 단계이다. 변수의 선언과 사용 등의 검사가 이루어지며, 코드의 의미적 일관성을 보장한다.

3.5 Python 변환기

KoBASIC을 Python으로 바꾸는 기능은 PyCodeGen 클래스를 통해 구현하였다. 클래스 내부 공개 API인 compile 메서드에서는 KoBASIC 코드를 입력받아 구문 분석한 후 결과로 도출된 추상구문 트리를 클래스 내부에 구현되어 있는 genPyCode 메서드에 전달한다. genPyCode 메서드가 다시 클래스 내부에 정의된 Python 문법의 코드를 만들어내는 여러 메서드들을 재귀적으로 호출하면서 주입된 추상 구문 트리에 맞는 Python 코드를 만들어 반환하고 이후 compile 메서드에서 반환된 Python 코드 문자열을 반환하면서 Python 변환이 끝나게 된다.

- Compile 메서드



```
1  compile(kobCode) {
2    const lines = kobCode
3      .trim()
4      .split("\n")
5      .filter((e) => e);
6    const ast = lines.map((l) => kobParser.parse(l));
7    this.lineNums = [];
8    ast.forEach((l) => this.lineNums.push(Number(l.number)));
9    const pyCode = this.genPyCode(ast);
10   return pyCode;
11 }
```

그림 11. Compile method

1. 줄 단위로 분리된 KoBASIC 문자열을 추상 구문 트리로 구문 분석한다.
2. 각각의 라인 넘버를 배열에 저장한다. 이는 이후 Python 변환할 때 라인 넘버가 필요한 기능들(goto, gosub 등)에 필요한 정보를 제공하기 위해서이다.
3. 클래스 내부의 genPyCode 메서드에 추상 구문 트리 포인터를 전달하여 파이썬 코드 변환 및 생성을 수행한다.
4. 변환된 Python 코드를 반환한다.

- genPyCode 메서드

변환된 Python 코드는 크게 top, mid, tail 파트로 나뉜다.

top은 main 함수를 정의하는 부분이다. main 함수에서는 mid에 정의되는 각 줄에 해당하는 함수(stmt10, stmt30 등)를 순차적으로 호출한다. 만약 특정 줄에서 End 등이 호출되어 프로그램이 종료되는 상황이 있으면, 해당 줄의 함수에서 예외를 발생시키고 main 함수에서 예외를 잡아서 이후 줄에 해당하는 함수를 실행하지 않고 프로그램을 종료할 수 있다.

mid는 주어진 코드에 나타난 모든 줄을 함수로 만든다. 이 기능을 지원하기 위해 클래스 내부의 genStmt 메서드로 각 줄의 구문 분석된 토큰을 넘긴다.

tail은 BASIC과 Python 간 호환되지 않는 문법을 지원하기 위한 내장 함수들이 정의되는 부분이다. 필요한 모든 내장 함수를 조건적으로 포함시킨 후, 맨 마지막에 main 함수를 실행하여 프로그램을 동작하도록 한다.

- getStmt 메서드




```
1  genStmt(l) {
2    this.currentLine = Number(l.number);
3    if (this.forStack.length > 0) {
4      this.midList[
5        this.forStack[this.forStack.length - 1]
6      ] += '\n      stmt${this.currentLine}()';
7      this.linesInFor.push(this.currentLine);
8    }
9    let header = `def stmt${this.currentLine}():\n`;
10   let body = l.stmts.map((exp) => this.gen(exp)).join("\n");
11   let indented_body = body
12     .split("\n")
13     .filter((e) => e)
14     .map((l) => `    ${l}`)
15     .join("\n");
16   this.midList.push(header + indented_body);
17 }
```

그림 12. getStmt method

genStmt 메서드는 BASIC의 한 문장을 받아 Python 함수로 변환하는 역할을 한다. this.currentLine은 현재 처리 중인 문장의 번호를 저장한다. BASIC 언어는 각 문장에 번호가 붙는 경우가 많기 때문에, 이를 currentLine에 저장하여 관리한다. forStack은 for 루프가 중첩되었을 때, 그 구조를 관리하기 위한 스택이다. 만약 forStack에 데이터가 있으면, 현재 변환 중인 for 루프의 마지막 부분에 해당 문장의 함수 호출(stmt{currentLine}())을 추가한다. 그 후, 변환된 BASIC 문장에 대해 Python 함수의 헤더를 생성한다. 예를 들어, stmt{currentLine}이라는 함수 정의가 생성된다. body는 BASIC 문장의 각각의 표현식을 Python 코드로 변환하여 저장하는 부분이다. 각 표현식은 gen(exp) 메서드를 통해 변환된다. 마지막으로, 변환된 문장들을 적절한 들여쓰기를 통해 Python 함수의 본문으로 만들어서 midList 배열에 추가한다.

즉, 이 메서드는 BASIC의 한 문장을 Python 함수로 변환하고, 그 변환된 코드를 관리하는 리스트(midList)에 추가하는 기능을 한다.

- gen 메서드



```

1  gen(exp) {
2      if (this[exp.type] == null) {
3          throw `Unexpected expression: "${exp.type}.";
4      }
5      return this[exp.type](exp);
6  }
```

그림 13. gen method

gen 메서드는 BASIC 문장에 포함된 개별 표현식을 Python 표현식으로 변환하는 역할을 한다. 이 메서드는 exp.type에 따라 표현식의 타입을 확인하고, 그 타입에 맞는 변환을 수행한다. 만약 해당 타입에 맞는 변환 메서드가 없으면 에러를 던진다. 예를 들어, FOR 타입의 표현식이면, this.FOR(exp)를 호출하여 변환 작업을 수행한다. gen 메서드는 BASIC 표현식을 Python 코드로 변환하는 핵심 함수로, 각 표현식의 타입에 따라 적절한 변환 메서드를 호출하

여 처리한다.

- 타입별 변환 메서드



```
1  // 원자 타입 (NUMBER, STRING)
2  NUMBER(exp) {
3    return `${exp.value}`;
4  }
5
6  STRING(exp) {
7    return `${exp.value}`;
8  }
9
10 // 변수명
11 VARIABLE_NAME(exp) {
12   if (!exp["args"]) {
13     return `vars["${exp.name}"]`;
14   } else {
15     return `vars["${exp.name}"]${exp.args
16       .map((a) => `${this.gen(a)}`)
17       .join("")}`;
18   }
19 }
20
21 // 배열 선언 DIM
22 DIM(exp) {
23   let left = `vars["${exp.var}"] = `;
24   let right = `[0] * ${this.gen(exp.dimension.pop())}`;
25   let popped = "";
26   while ((popped = exp.dimension.pop())) {
27     right = `${right}] * ${this.gen(popped)}`;
28   }
29   return left + right + "\n";
30 }
31
32 // 할당 문법 (LET, ASSIGN)
33 LET(exp) {
34   return `${this.gen(exp.var)} = ${this.gen(exp.expr)}\n`;
35 }
```

그림 14. 타입별 변환 method

타입별 변환 메서드란 특정 타입을 가지고 있는 KoBASIC 토큰을 그에 걸맞는 파이썬 코드로 변환하는 함수이다. 더 이상 쪼개질 수 없는 타입(원자 타입)이 아닌 경우, 해당 토큰 내부에 또 다른 토큰이 있다는 것이므로 타입 변환 메서드 내부에서 gen 함수를 다시 호출하고 내부 토큰을 전달하는 방식으로 구현하였다. 이 방식으로 주어진 모든 토큰을 순회하며 파이썬 코드를 생성할 수 있게 된다.

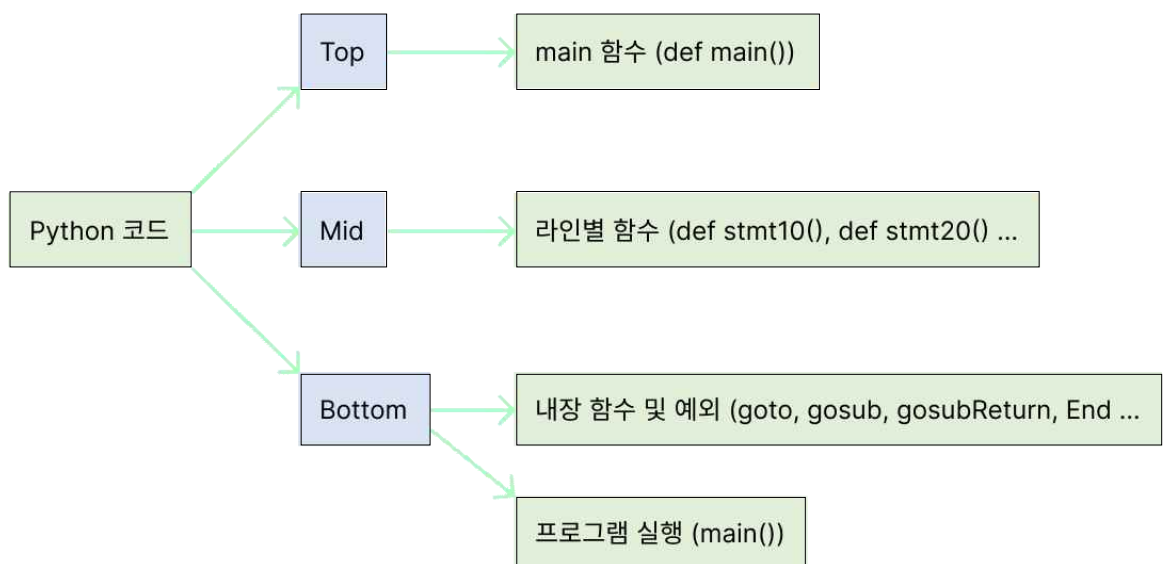



그림 15. Python 변환 구조

KoBASIC을 변환한 Python 코드는 위와 같은 구조로 작성되었다. 크게 Top, Mid, Bottom으로 나뉜다. 먼저 맨 위에 main 함수를 선언한다. main 함수는 실행될 때 Mid에서 나타나는 라인별 함수를 순차적으로 실행하도록 구성되어 있다. Mid에서는 KoBASIC으로 작성한 각 줄을 Python 함수로 번역하여 순차적으로 선언하고 있다. 마지막 Bottom에서는 Python에는 존재하지 않지만, BASIC에는 존재하는 Gosub, Goto 등을 Python에서 구현하기 위해 추가적으로 구현한 함수와 Input의 자료형을 결정하기 위한 함수 등이 담겨 있다. 또한 프로그램과 서브루틴의 종료를 나타내기 위해 구현한 End, GosubReturn 등의 예외 클래스가 있다. 이렇게 미리 정의한 코드들을 적절하게 포함한 후 맨 끝에서 main 함수를 호출하여 프로그램을 실행한다. KoBASIC 이중반복문 예제

를 Python으로 변환한 결과 코드가 아래에 나와 있다. 이를 보면 더 정확하게 이해할 수 있다.



```

1  def main():
2      try:
3          stmt10()
4      except End:
5          return
6
7  vars = {}
8
9  def stmt10():
10     for vars["I"] in range(1, 4, 1):
11         stmt20()
12         stmt50()
13
14  def stmt20():
15     for vars["J"] in range(1, 4, 2):
16         stmt30()
17         stmt40()
18
19

```

그림 16. KoBASIC 이중반복문 예제를 Python으로 변환한 코드

```

18
19  def stmt30():
20     print("I = ", end="")
21     print(vars["I"], end="")
22     print(", J = ", end="")
23     print(vars["J"], end="")
24     print(", I * J = ", end="")
25     print((vars["I"] * vars["J"]), end="")
26     print()
27
28  def stmt40():
29     pass
30
31  def stmt50():
32     pass
33
34  # 내장 함수 및 예외 Inner Function and Exception
35
36  class End(Exception):
37     pass
38
39
40  # 프로그램 실행
41
42  main()

```

그림 17. KoBASIC 이중반복문 예제를 Python으로 변환한 코드

KoBASIC을 Python으로 변환할 때, 주요한 과제는 BASIC에는 존재하지만, Python에는 존재하지 않는 Goto, Gosub 문법을 구현하는 것이었다. Goto는 아래 그림과 같이, KoBASIC에서 Goto를 사용했을 때, 변환 Python 코드에 자동으로 포함되게 되어 있는 내장 함수 Goto를 통해 구현하고 있다.

```
93     def goto(from_line, to_line):
94         if from_line > to_line:
95             linenums_goto_use = linenums[
96                 linenums.index(to_line) : linenums.index(from_line) + 1
97             ]
98             for linenum in linenums_goto_use:
99                 lines[linenum]()
100         else:
101             linenums_goto_use = linenums[linenums.index(to_line) :]
102             for linenum in linenums_goto_use:
103                 lines[linenum]()
104         raise End()
```

그림 18. Goto 문법의 Python 구현 코드

Goto 함수는 함수를 호출하는 줄 번호와 이동하는 줄 번호를 각각 from_line, to_line 인수로 받는다. 클래스 내부적으로, linenums 리스트에는 줄 번호들을, lines 딕셔너리에는 각 줄의 번호를 key로 저장하고 각 줄의 로직을 나타내는 함수를 value로 담도록 되어 있다. Goto는 바로 이 linenums 리스트에서 from_line과 to_line을 기준으로 적당한 범위를 slice한 다음, 선택된 줄 번호들에 대한 함수를 lines 딕셔너리에서 찾아서 호출한다. Goto를 호출하면 새로운 call stack이 열리는데 호출한 줄 번호 이후의 줄 번호로 이동하는 경우에는 call stack을 닫을 뿐만 아니라 Goto 함수의 call stack을 열었던 main 함수의 call stack도 닫아야 한다. 그러기 위해서 이 경우에는 End() 예외를 발생시켜 프로그램이 종료되도록 구현하였다.

End 예외는 아래와 같이 선언되어 있다.

```
121     class End(Exception):
122         pass
```

그림 19. End 예외 코드

또한 main 함수에서는 End 예외가 발생할 경우, Return을 호출하여 함수 실행을 즉시 종료하도록 하고 있다. BASIC에는 함수 선언의 개념은 없고, Gosub를 통해 subroutine을 구현하도록 하고 있다. Gosub는 Return이 나올 때까지 Goto와 동일하게 동작하다가, Return을 만나면 Gosub를 호출한 라인으로 복귀한다는 특징이 있다. 이러한 특징을 반영하기 위해서, Goto와 마찬가지로 Gosub 내장 함수를 구현하여 이를 각 줄 번호 함수에서 호출하는 방식으로 작동하게 하였다. Gosub 함수의 구현사항은 아래와 같다.

```
110     def gosub(from_line, to_line):
111         try:
112             goto(from_line, to_line)
113         except:
114             return
```

그림 20. Gosub 문법의 Python 구현 코드

Gosub 함수는 내부적으로 Goto 함수를 통해 구현된 syntax sugar이다. Gosub의 복귀를 일으키는 Return은 예외로 구현되었다. Return 예외 클래스는 다음과 같다.

```
116     class GosubReturn(Exception):
117         pass
```

그림 21. Return 예외 클래스의 Python 구현 코드

Gosub 함수의 call stack이 실행되는 중에, Return을 만나 GosubReturn이 발생하면, Gosub 함수의 except문이 이러한 예외를 받아 즉시 Gosub의 call stack을 종료하게 된다. 그러면 Gosub를 호출한 곳에서 다시 실행 흐름이 이어지게 되므로, BASIC의 Gosub 작동 방식을 구현한 결과가 된다.

3.6 Python 변환 웹 인터페이스 구현

KoBASIC은 현장에서 활용된다면 교육 목적으로 사용될 가능성이 크다. 따라서 사용자들이 쉽게 사용할 수 있도록 접근성을 높이는 것이 중요한 목표 중 하나이다. 이러한 필요성에 따라 추가적으로 KoBASIC 코드를 Python 코드로 변환해주는 웹 인터페이스를 구현했다.



그림 22. KoBASIC WebApp에서 이중반복문 코드 실행 결과

초기에는 C++로 파서와 코드 변환기를 구현하여 서버에서 파싱, 변환, 실행 등 핵심 기능을 구현하고자 하였다. 하지만 이렇게 구현할 경우, Input을 만날 때마다 서버와 클라이언트의 소통이 필요하므로 Input()을 처리할 수 있도록 만드는 과정이 까다롭고, 서버를 운영하는 데 드는 비용이 장기적인 관점에서 부담스러울 수 있다.

이러한 문제점을 해결하기 위해서 파싱, Python 변환, Python 실행 모두를 브라우저에서 처리하는 것으로 방향성을 바꾸었다. 새로운 방향성에 따라 파서, 코드 제너레이터를 포함한 트랜스파일러를 자바스크립트로 구현하였다. 그리고 이 자바스크립트 코드를 웹 클라이언트에 모두 포함해서 브라우저에서 파싱 및 Python 코드 변환이 이루어질 수 있도록 했다. 브라우저에서 Python을 실행하는 것은 Web Assembly 기반의 Pyodide 라이브러리를 사용하였다. 이를 통해, Input()을 문제없이 실행할 수 있게 되었고 github pages와 같은 정적 웹 무료 호스팅 서비스를 활용해 서버 비용의 부담도 줄일 수 있게 되었다. 현재 배포된 서비스는 <https://kimjeonghan0625.github.io/kobasic/>에 있다.

4. 연구 결과 분석 및 평가

4.1 테스트 케이스 분석

우리는 KoBASIC이 실용적이고 교육적인 프로그래밍 언어로서 충분히 활용 가능한지 여부를 평가하기 위해 테스트 케이스 분석을 진행하였다. 이를 위해 Applesoft BASIC Programming Reference Manual에 수록된 다양한 예제 코드를 KoBASIC에서 실행하며, 그 성능과 호환성을 검증하였다.

Applesoft BASIC은 1977년에 Apple에서 발표한 인터프리터 언어로, 당시 AppleII 컴퓨터에서 주로 사용되던 BASIC 언어의 변형 중 하나였다. Applesoft BASIC은 프로그래밍 입문자들이 학습하기 쉽도록 설계된 언어로서, 다양한 명령어와 함수, 제어 구조를 제공한다. 이 언어를 기반으로 한 예제 코드는 기본적인 입출력, 조건문, 반복문, 배열 처리 등 프로그래밍의 기초적인 개념을 포함하고 있어, KoBASIC의 기능을 전반적으로 테스트하는 데 적합한 자료라고 판단하였다.

4.1.1 Execution Commands

PRINT와 같은 명령어는 즉시 실행 명령어(Immediate-Execution Commands)라고 한다. 그 이외에도 지연 실행 명령어(Deferred-Execution Commands)라는 종류의 명령어가 있다. 모든 지연 실행 명령어는 줄 번호(line number)로 시작하는데, Apple BASIC에서는 줄 번호는 0에서 63999까지의 정수로 이루어져 있다.

지연 실행 명령어의 일련의 명령들은 프로그램(program)이라고 부른다. 지연 실행 명령어는 즉시 실행되지 않고, 명령어를 저장하고 이후 RUN을 입력하면, 가장 작은 줄 번호부터 시작해서, 차례대로 다음 명령어를 실행한다. 프로그램이 완전히 실행될 때까지 계속해서 다음 명령어를 실행한다.

AppleSoft BASIC 매뉴얼의 설명을 바탕으로 KoBASIC에서도 동일한 방식으로 지연 실행 명령어가 잘 동작함을 확인하였다. KoBASIC은 AppleSoft BASIC의 구조와 개념을 기반으로 개발되었기 때문에, 줄 번호를 사용한 명령어 저장과 실행이 문제없이 이루어진다. 아래의 예시는 KoBASIC이 지연 실행 명령어를 정확하게 처리하고, AppleSoft BASIC과 유사하게 줄 번호를 통해 프로그램을 순차적으로 실행할 수 있음을 증명한다.

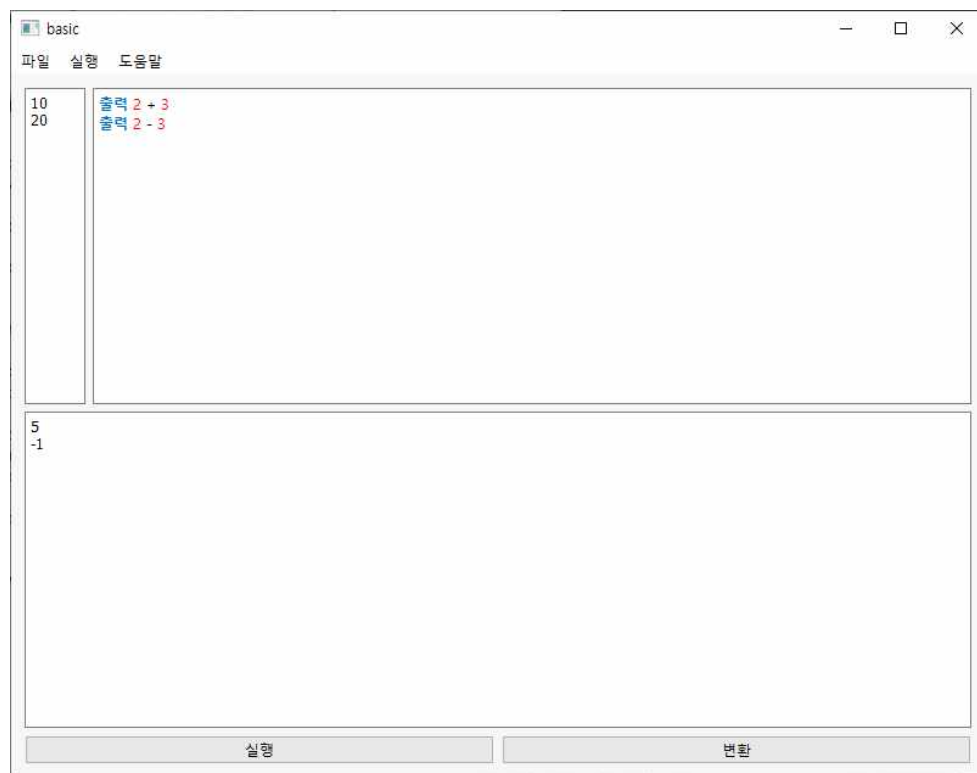


그림 23. 지연 실행 명령어 KoBASIC 예시

4.1.2 Number Format

KoBASIC은 현재 부동소수점 실수 연산을 지원하지 않는 상태이지만, 향후 개발을 통해 이를 추가할 예정이다. 이는 현재 KoBASIC에서의 수학 연산이 정수 연산으로 제한된다는 것을 의미한다. 그러나, 정수 연산은 모든 기능이 정상적으로 지원되며, 정수 기반의 계산이나 로직 처리에는 문제가 없다.

부동소수점 실수 연산은 다양한 프로그래밍 환경에서 중요한 역할을 하지만, KoBASIC의 초기 목표는 프로그래밍 초보자들이 쉽게 접근할 수 있는 언어를 제공하는 것이므로, 먼저 정수 연산을 안정적으로 구현한 상태이다. 향후 업데이트를 통해 부동소수점 실수 연산 기능을 추가함으로써, KoBASIC의 연산 범위를 확장하고, 복잡한 수학적 계산이 필요한 분야에서도 활용가능하도록 발전시킬 계획이다.

4.1.3 Print Format

PRINT 명령어에서 콤마를 포함시키면, 출력된 값 다음에 탭 필드로 이동하여 그 다음 값을 출력한다. 즉, 각 값 사이에 일정한 간격이 생긴다. 아래는 KoBASIC에서의 예시이다. 각 숫자 사이에 탭 간격이 적용되어 출력된 것을 알 수 있다.

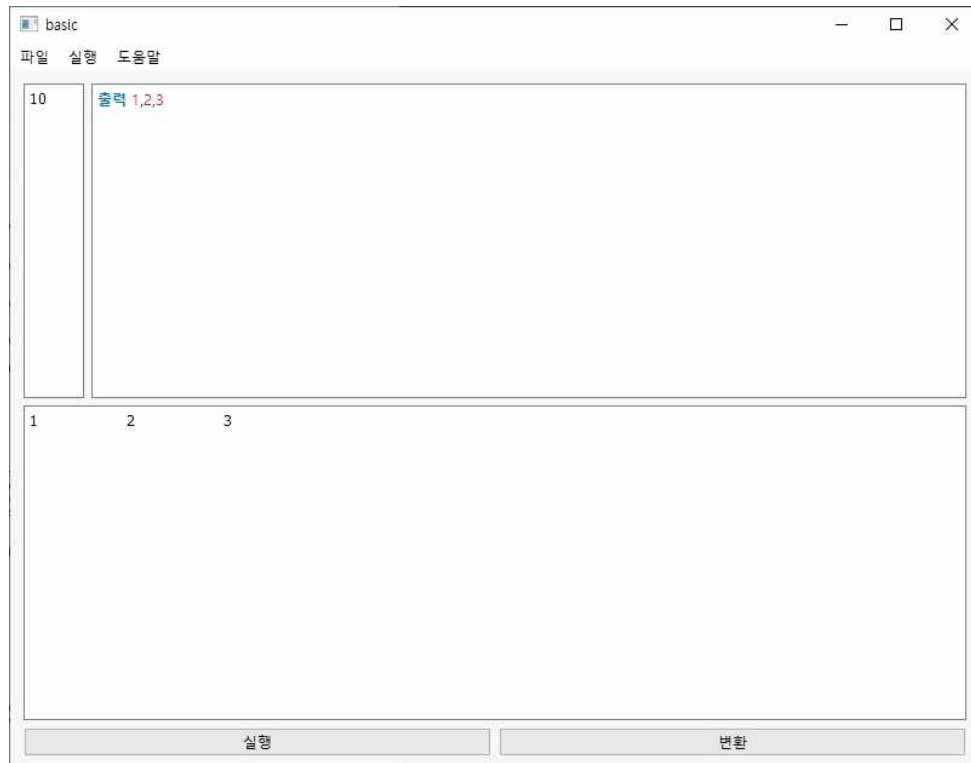


그림 24. Print 명령어 콤마 포함 KoBASIC 예시

세미콜론을 사용하면, 다음 값이 이전 값 바로 뒤에 출력된다. 즉, 세미콜론을 사용한 경우에는 출력된 값 사이에 간격이 없다.

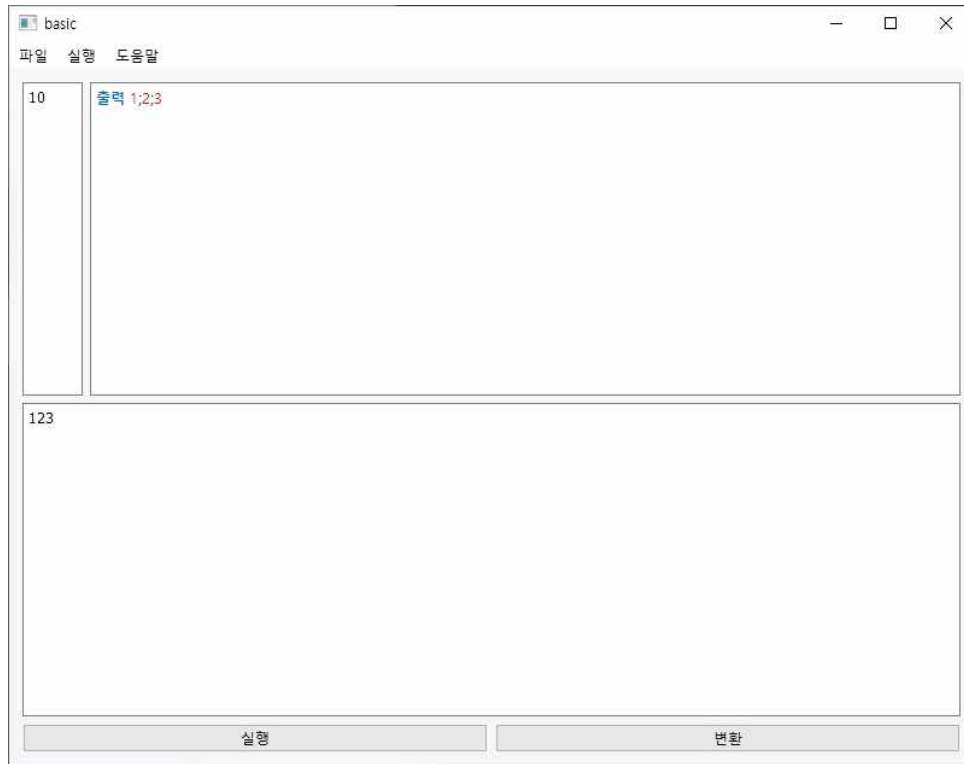


그림 25. Print 명령어 세미콜론 포함 KoBASIC 예시

음수와 양수를 포함한 값을 출력할 때도 마찬가지이다. 세미콜론을 사용하면 음수와 양수가 붙어 출력된다.

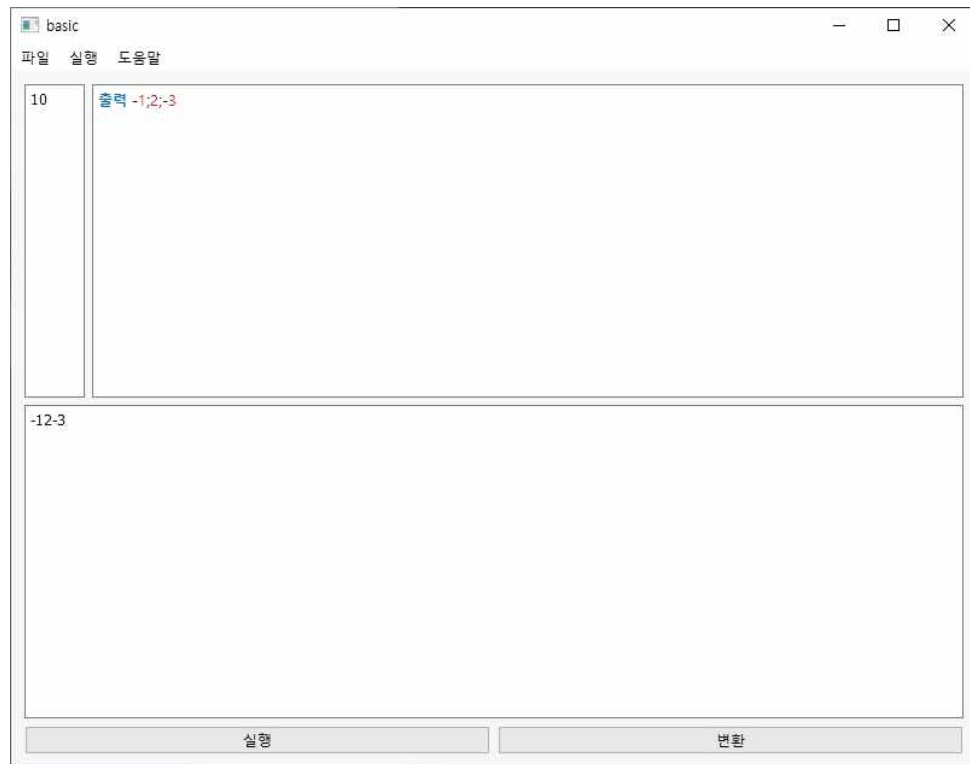


그림 26. Print 명령어 세미콜론에서 양수 음수에 대한 KoBASIC 예시

이를 바탕으로 AppleSoft BASIC의 매뉴얼의 명령어가 KoBASIC에서도 동일하게 기능이 작동함을 확인할 수 있었다.

4.1.4 Variable Names

AppleSoft BASIC에서 변수는 특정 메모리 위치를 가리키는 식별자이다. 변수 이름을 정의할 때 다음과 같은 규칙을 준수해야 한다. 변수 이름은 알파벳 문자로 시작해야 한다. 첫 번째 문자를 제외한 나머지 자리에는 알파벳 문자(A-Z) 또는 숫자(0-9)를 사용할 수 있다. 또 변수 이름은 최대 238자까지 지정할 수 있지만, AppleSoft BASIC에서는 변수 이름의 첫 두 글자만을 사용해 서로 다른 변수를 구별한다.

예를 들어, "GOODNIGHT"와 "GOLDRUSH"는 첫 두 글자(GO)가 동일하므로 같은 변수로 간주된다. 이러한 특성으로 인해 변수 이름을 지을 때 첫 두 글자가 중복되지 않도록 주의해야 한다.

AppleSoft BASIC에서는 변수 이름의 첫 두 글자 이후의 문자는 일반적으로 무시되지만, 특정한 예약어가 포함된 경우 예외가 발생할 수 있다. 변수 이름을 정할 때는 예약어와의 충돌을 피해야 하며, 이를 통해 코드의 가독성과 실행의 정확성을 확보할 수 있다.

KoBASIC에서의 변수 이름은 다음과 같은 패턴을 따른다.

```
QString identifierPattern
= "(?![ㄱ-ㅎㅏ-ㅣ가-힣a-zA-Z_0-9])[ㄱ-ㅎㅏ-ㅣ가-힣a-zA-Z_][ㄱ-ㅎㅏ-ㅣ가-힣a-zA-Z_0-9]*(?![ㄱ-ㅎㅏ-ㅣ가-힣a-zA-Z_0-9])";
```

그림 27. KoBASIC의 변수 이름 패턴

이 패턴을 이용하여 KoBASIC에서 변수 이름을 정의할 때 한글, 영문자 및 밑줄(_)을 변수의 시작 문자로 사용할 수 있다. 또한, 그 뒤에는 숫자를 포함한 한글, 영문자, 밑줄, 숫자(0-9)가 이어질 수 있다. 변수 이름은 반드시 한글, 영문 대소문자, 혹은 밑줄로 시작해야 한다. 첫 번째 문자 이후에는 한글, 영문자, 밑줄, 숫자가 포함될 수 있으며, 숫자는 두 번째 문자부터 사용할 수 있다.

이 규칙을 통해 KoBASIC에서는 가독성을 유지하면서도 유연한 변수 이름 사용이 가능하도록 설계되었다.

4.1.5 IF...THEN

IF...THEN 문은 특정 조건에 따라 프로그램의 흐름을 변경하는 데 사용된다. 조건을 평가하여 참(True)일 경우 특정 명령을 실행하며, 거짓(False)일 경우 다른 경로를 따른다.

IF와 THEN 사이에는 조건식이 있다. 조건식의 결과가 참일 경우 THEN 다

음에 오는 명령이 실행된다. 예를 들어, IF X = 0 THEN GOTO 50에서 X가 0이면 50번 줄로 이동한다. 조건식은 두 표현식 사이에 비교 연산자를 사용해 작성하며, 지원되는 연산자는 다음과 같다.

=
<>
<
>
<=
>=

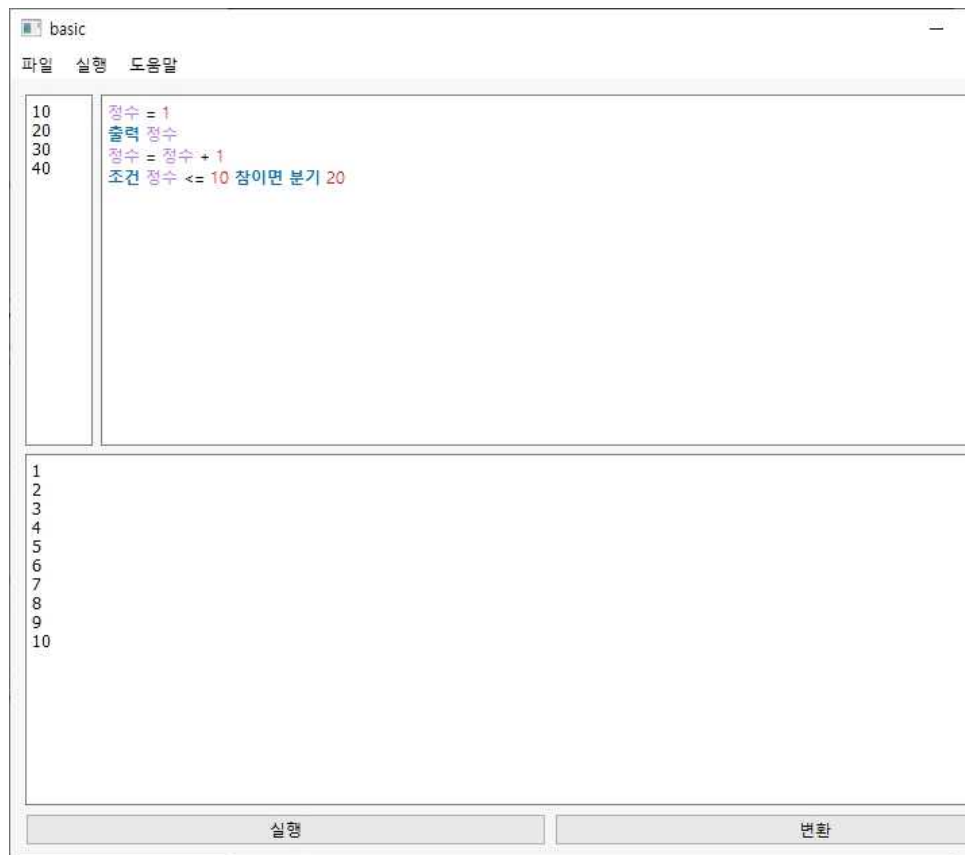


그림 28. KoBASIC의 IF...THEN 예시

위 매뉴얼을 참고해서 KoBASIC에서 해당 예제 프로그램을 실행시켰다. 예제에 설명된 대로 프로그램이 정상적으로 작동함을 확인할 수 있었다. 사용자 입력에 따라 올바른 메시지를 출력하고, 조건에 따라 프로그램의 흐름이 정확

하게 제어되었다. 이를 통해 KoBASIC이 AppleSoft BASIC의 IF...THEN 구문을 충실히 구현하고 있음을 확인할 수 있었다.

4.1.6 FOR...NEXT

FOR...NEXT 문은 반복적인 작업을 수행할 때 사용되는 루프 구조이다. 이를 통해 프로그램은 일정한 범위 내에서 명령을 반복적으로 실행할 수 있다. 초기에 작성된 프로그램은 각 숫자에 대해 수작업으로 PRINT 명령을 사용해 비효율적이었다. 그러나 FOR...NEXT를 사용하면서 훨씬 더 간결하고 효율적인 코드로 변경할 수 있었다. 예를 들어, 1부터 10까지의 정수에 대한 제곱근 표를 계산하는 프로그램을 작성할 수 있다. AppleSoft BASIC에서 제곱근을 계산하는 함수는 SQR(X)이며, X는 제곱근을 구할 숫자를 나타낸다. KoBASIC에서는 제곱근을 계산하는 함수를 만들어서 해당 예시를 실행하였다.

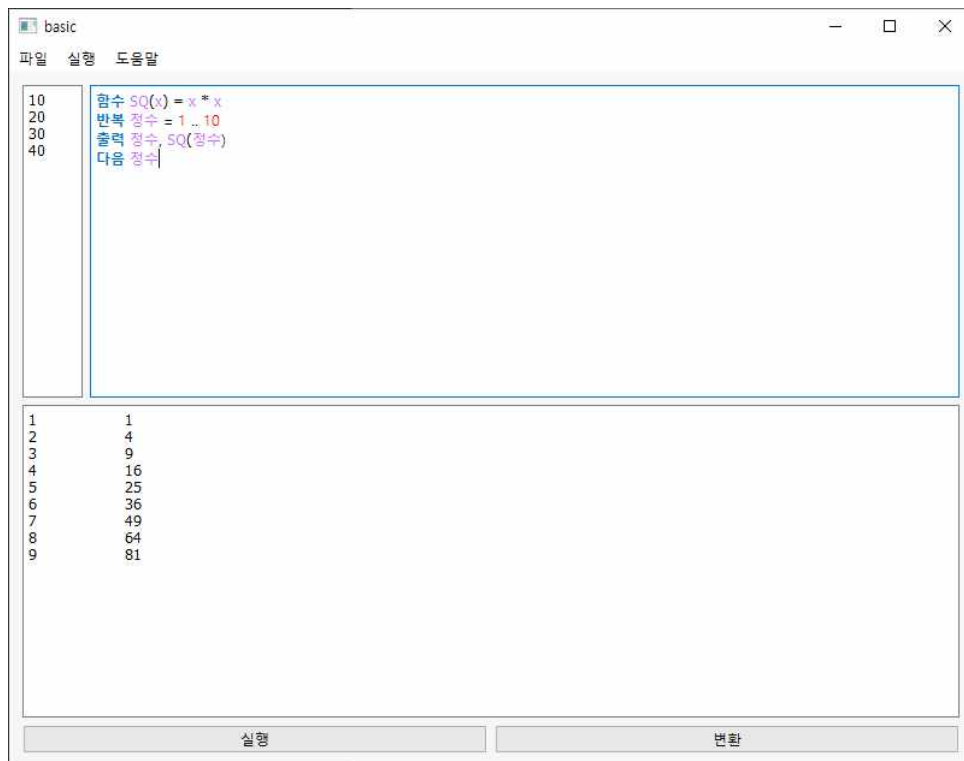


그림 29. KoBASIC의 제곱근 계산 예시

STEP 키워드는 AppleSoft BASIC의 FOR...NEXT 반복문에서 반복할 때마다 변수의 값을 얼마씩 증가(또는 감소)시킬지를 지정하는 데 사용된다. 기본적으로

로 FOR 문에서 변수는 1씩 증가하지만, STEP을 사용하면 원하는 값을 지정할 수 있다. FOR N = 1 TO 10처럼 STEP을 생략하면 기본적으로 N은 1씩 증가한다. FOR N = 1 TO 10 STEP 2와 같이 사용하면 N의 값이 1에서 시작하여 2씩 증가한다. 즉, N의 값은 1, 3, 5, 7, 9와 같이 된다. 아래는 STEP을 사용한 KoBASIC의 예시이다. KoBASIC에서 STEP은 폭이라는 키워드로 사용되었다.

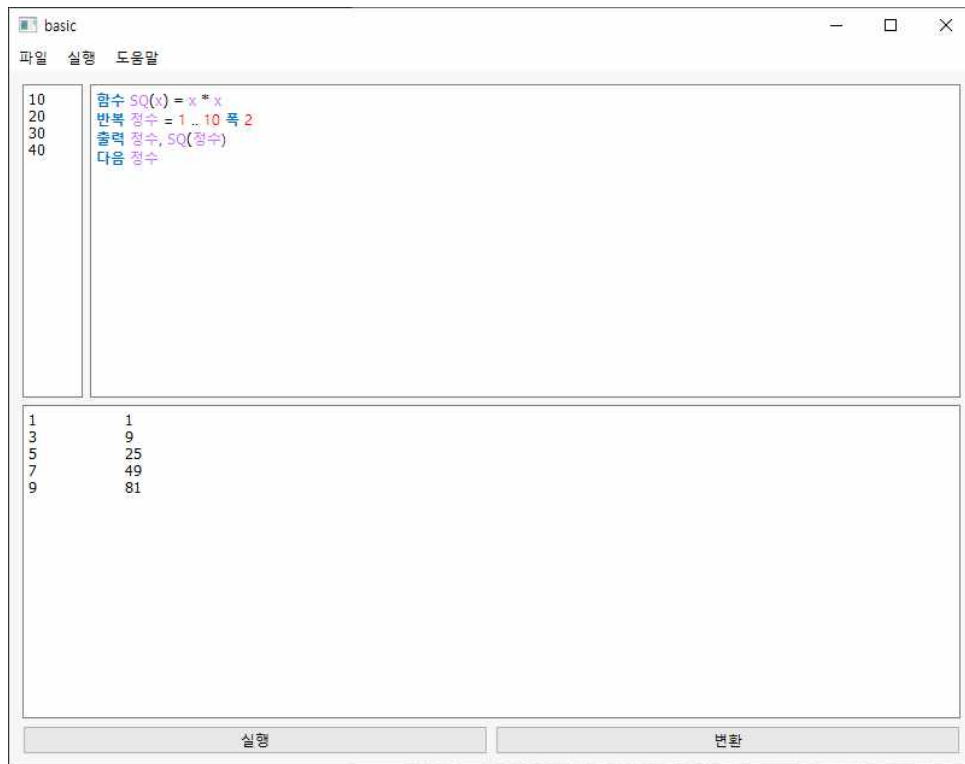


그림 30. KoBASIC의 STEP 예시

또 FOR...NEXT 루프를 중첩하여 사용할 수 있다. 즉, 하나의 FOR 루프 안에 또 다른 FOR 루프를 넣어 반복 작업을 수행할 수 있다. 이를 통해 2차원 배열이나 테이블 형식의 데이터를 처리하거나 특정 범위 내에서 반복 작업을 효율적으로 수행할 수 있다. 아래는 KoBASIC에서의 이중 FOR문 예시이다.

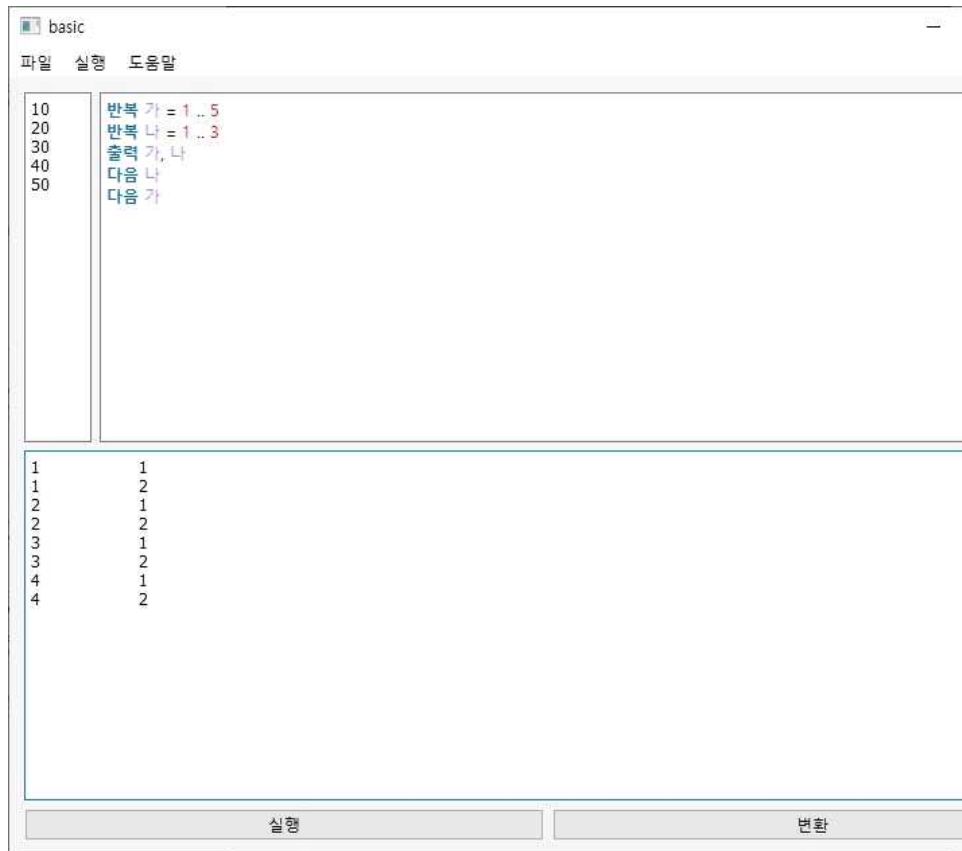


그림 31. KoBASIC의 이중 FOR문 예시

위의 예제들은 KoBASIC에서 실행되었으며, FOR 루프가 올바르게 작동함을 확인할 수 있었다. 특히, 올바르게 작성된 루프는 기대한 대로 각 반복을 수행하며 정확한 결과를 출력했습니다. 이를 통해 KoBASIC이 AppleSoft BASIC의 FOR...NEXT 구문을 충실히 구현하고 있으며, 중첩된 반복문에서도 정확하게 동작함을 확인할 수 있었다.

4.1.7 Arrays

AppleSoft BASIC에서는 배열을 사용하여 숫자들의 집합을 관리할 수 있다. 배열은 동일한 이름 아래에 여러 개의 데이터를 저장할 수 있는 자료 구조로, 각 데이터를 쉽게 참조할 수 있다. 배열의 각 원소는 인덱스 또는 첨자(subscript)를 사용해 접근할 수 있다.

배열 이름은 일반 변수 이름과 동일한 방식으로 정의할 수 있으며, 예를 들어 A라는 배열을 사용할 수 있다. 배열 이름 A는 단순한 변수 A와는 다르며, 같

은 프로그램에서 둘 다 사용할 수 있다.

배열의 특정 원소를 선택하려면 첨자를 사용한다. 예를 들어 A(I)는 배열 A의 I번째 원소를 의미한다. 배열을 사용하기 전에 크기를 지정해야 한다. 이는 DIM 명령어를 사용해 이루어진다. 배열을 사용하면 프로그램에서 다량의 데이터를 효율적으로 관리할 수 있어 계산 작업이나 데이터 테이블 관리에 유용하다.

```
10 DIM A(8) : DIMENSION ARRAY WITH MAX. 9 ELEMENTS
20 REM ASK FOR 8 NUMBERS
30 FOR I = 1 TO 8
40 PRINT "TYPE A NUMBER: ";
50 INPUT A(I)
60 NEXT I
70 REM PASS THROUGH 8 NUMBERS, TESTING BY PAIRS
80 F = 0 : REM RESET THE ORDER INDICATOR
90 FOR I = 1 TO 7
100 IF A(I) <= A(I+1) THEN GOTO 140
110 REM INTERCHANGE A(I) AND A(I+1)
120 T = A(I)
130 A(I) = A(I+1)
140 A(I+1) = T
150 F = 1 : REM ORDER WAS NOT PERFECT
160 NEXT I
170 REM F = 0 MEANS ORDER IS PERFECT
180 IF F = 1 THEN GOTO 80 : REM TRY AGAIN
190 PRINT : REM SKIP A LINE
200 REM PRINT ORDERED NUMBERS
210 FOR I = 1 TO 8
220 PRINT A(I)
230 NEXT I
```

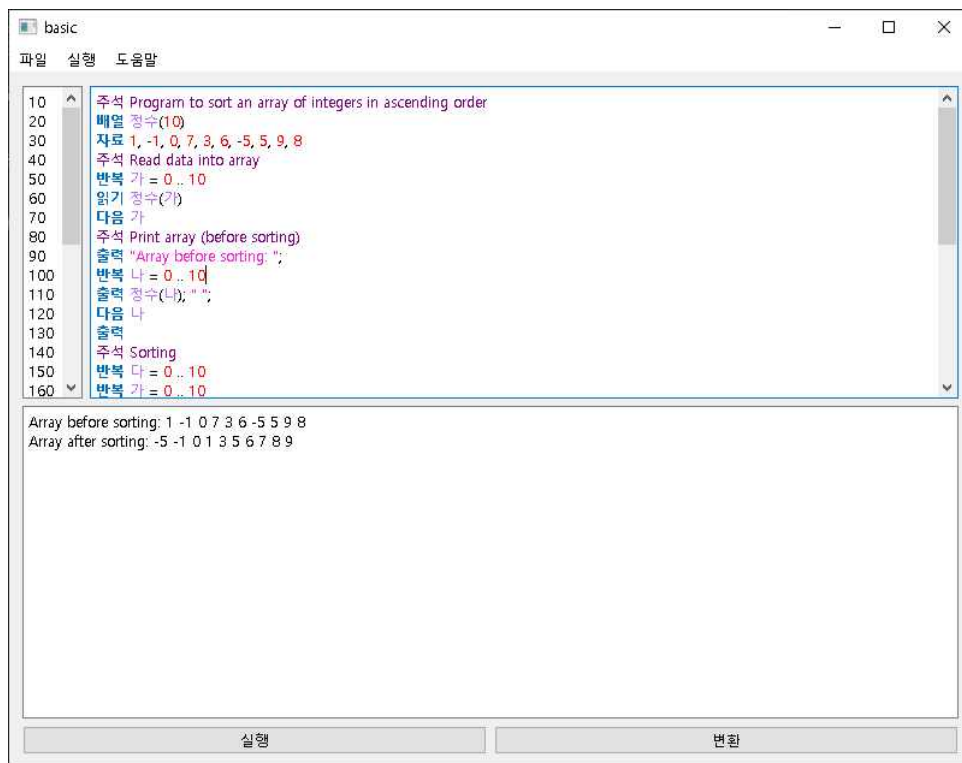


그림 32. KoBASIC을 이용한 정렬 예제

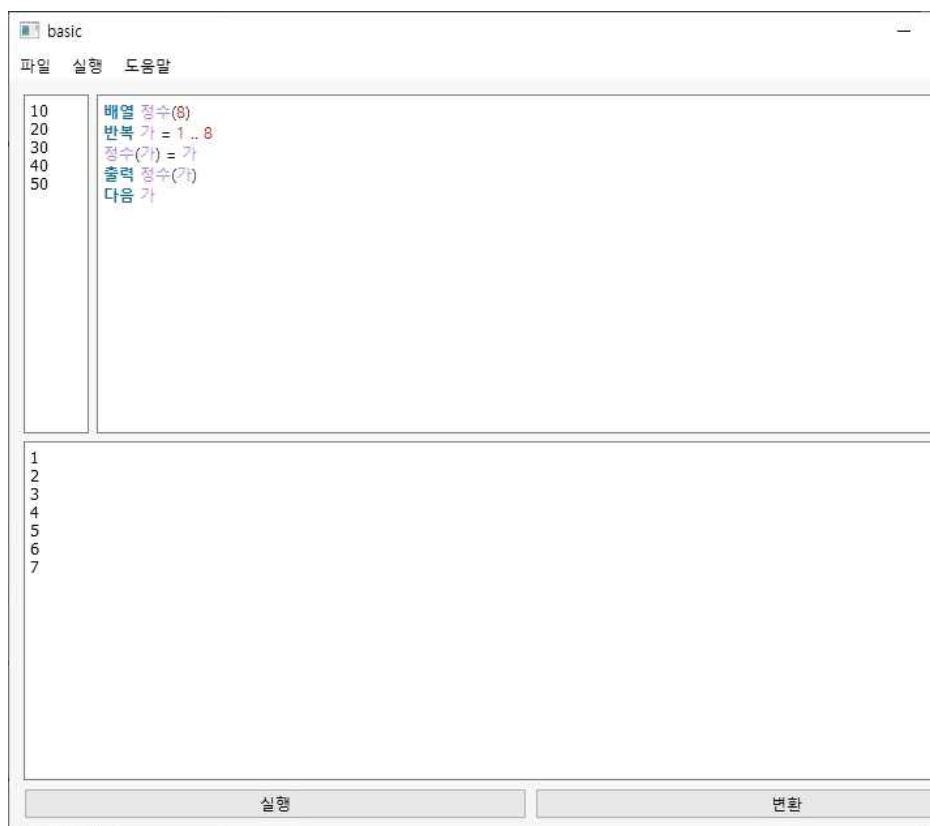


그림 33. KoBASIC을 이용한 정렬 예제

위의 배열 예제들은 KoBASIC에서 실행되었으며, 예상대로 작동함을 확인할 수 있었다. 배열의 크기를 설정하고, 각 원소에 접근하며 데이터를 저장 및 출력하는 과정이 정상적으로 이루어졌다. 이를 통해 KoBASIC이 AppleSoft BASIC의 배열 관련 기능을 충실히 구현하고 있음을 확인할 수 있었다. 이를 통해 KoBASIC 사용자는 배열을 이용해 다양한 데이터를 효과적으로 관리하고 처리할 수 있다.

4.1.8 GOSUB...RETURN

GOSUB와 RETURN은 AppleSoft BASIC에서 자주 사용되는 명령어로, 프로그램 내에서 같은 작업을 여러 곳에서 수행해야 할 때 유용하게 사용할 수 있다. 이 명령어들은 특정 코드 블록을 서브루틴으로 작성하고, 필요한 곳에서 호출하여 실행한 후, 원래 위치로 되돌아갈 수 있게 해준다.

프로그램이 GOSUB 명령어를 만나면, 지정된 줄 번호로 이동하여 서브루틴을 실행한다. 이때 GOSUB를 만나기 전의 프로그램 위치를 기억해 둔다.

RETURN 명령어를 만나면, 프로그램은 GOSUB 명령어 직후의 위치로 돌아가서 계속 실행을 이어간다. 이렇게 하면 서브루틴에서 실행된 작업을 마치고 원래의 흐름으로 돌아갈 수 있다.

GOSUB와 RETURN을 사용하면 프로그램의 동일한 코드를 여러 번 작성하지 않고도 여러 곳에서 사용할 수 있다. 이는 프로그램을 간결하고 유지보수하기 쉽게 만들어 준다. 예를 들어, 같은 계산이나 출력 작업이 필요할 때마다 GOSUB로 해당 서브루틴을 호출하여 사용할 수 있다.

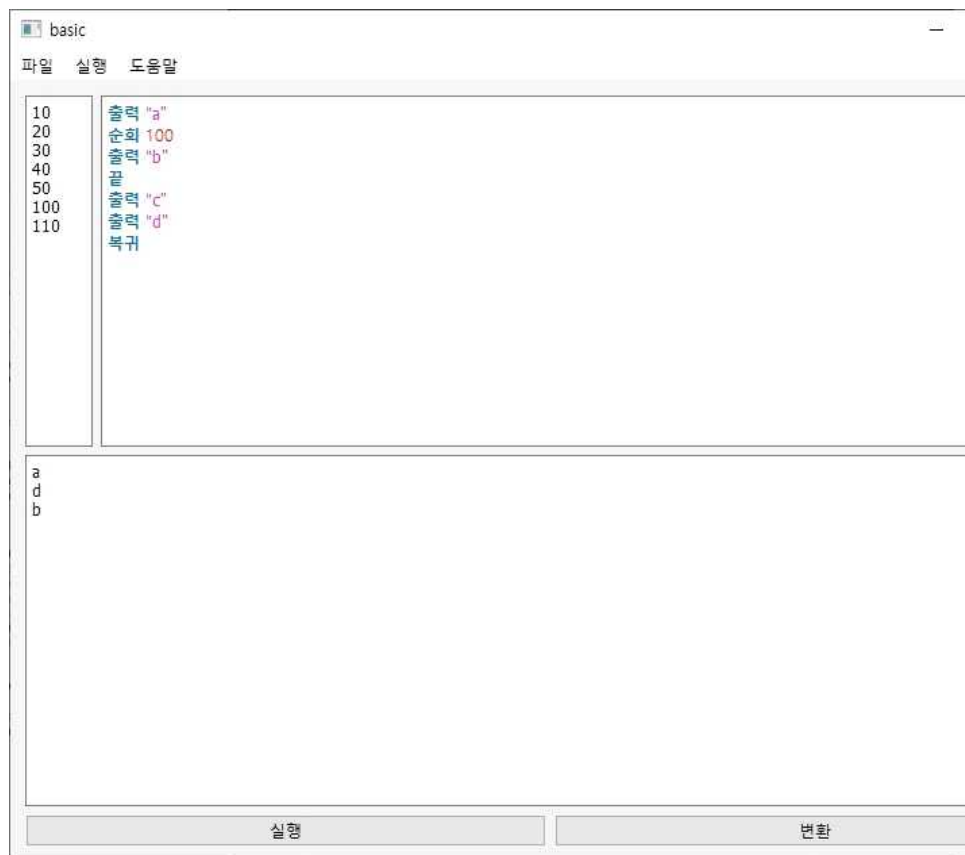


그림 34. KoBASIC을 이용한 GOSUB RETURN 예제

위의 프로그램은 KoBASIC에서 실행되었으며, GOSUB와 RETURN 명령어가 AppleSoft BASIC과 동일하게 작동하여 서브루틴을 호출하고 종료할 수 있었다. 이를 통해 KoBASIC이 서브루틴 관련 명령어를 충실히 구현하고 있음을 확인할 수 있었다.

4.1.9 READ...DATA...RESTORE

READ, DATA, 그리고 RESTORE 명령어는 프로그램에서 변경되지 않는 데이터를 관리하거나, 여러 번 사용할 수 있는 데이터 목록을 정의할 때 유용하다. 이를 통해 입력값 없이도 프로그램에서 데이터를 순차적으로 불러와 사용할 수 있다.

DATA는 프로그램 내에서 일정한 데이터 목록을 미리 정의한다. 예를 들어, DATA 키워드를 사용해 숫자나 문자열 리스트를 저장할 수 있다. DATA 구문

은 프로그램 어디에든 위치할 수 있지만, 데이터가 순차적으로 읽혀지므로, 위치에 따라 순서가 달라질 수 있다. READ 명령어는 DATA 구문으로부터 데이터를 하나씩 읽어와 변수에 저장한다. RESTORE 명령어는 DATA 목록의 읽기 위치를 처음으로 되돌린다. 이를 통해 READ 구문이 다시 처음부터 데이터를 읽을 수 있게 한다.

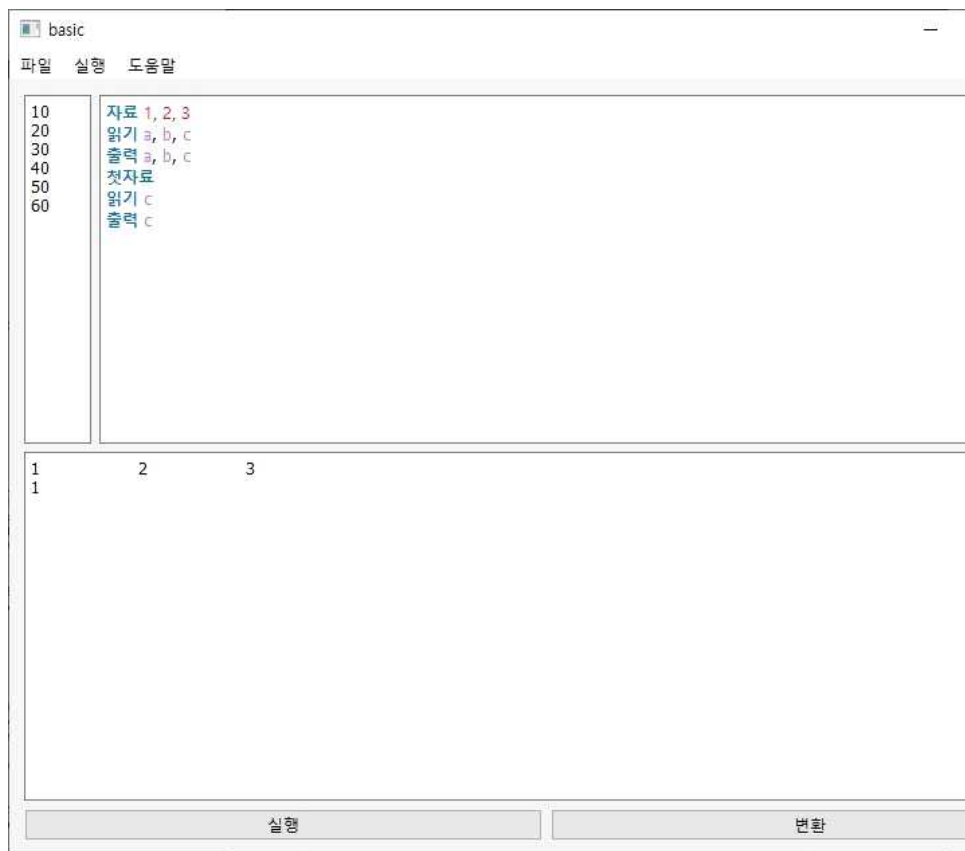


그림 35. KoBASIC을 이용한 READ DATA RESTORE 예제

이 프로그램은 KoBASIC에서 테스트되었으며, AppleSoft BASIC과 동일하게 READ, DATA, RESTORE 명령어가 정상적으로 작동함을 확인할 수 있었다. 이를 통해 KoBASIC이 AppleSoft BASIC의 데이터 관리 명령어를 정확히 구현하고 있음을 확인했다.

4.1.10 Real, Integer and String Variables

AppleSoft BASIC에서는 세 가지 유형의 변수를 사용할 수 있다.

실수 변수는 기본적으로 AppleSoft BASIC에서 사용하는 변수 유형이다. 소수점 이하 최대 9자리까지의 정확도를 가지며, 값의 범위는 10^{-38} 에서 10^{38} 사이이다. AppleSoft BASIC은 내부적으로 실수 값을 이진수로 변환하여 저장하고, 출력할 때 다시 10진수로 변환한다. 이 과정에서 반올림 오류가 발생할 수 있다.

정수 변수는 % 기호를 사용하여 정의되며, 변수 이름 뒤에 %를 붙인다. 정수는 -32767부터 32767까지의 값을 저장할 수 있고, 정수 변수는 FOR 문이나 DEF 구문에서는 사용할 수 없다.

문자열 변수는 \$ 기호를 사용하여 정의되며, 변수 이름 뒤에 \$를 붙인다. 문자열 변수는 0부터 255개의 문자까지 저장할 수 있다. 문자열 변수는 텍스트 데이터를 다루거나 문자 기반의 입력과 출력을 처리할 때 유용하다.

아래는 AppleSoft BASIC의 변수를 표로 정리하였다.

유형	변수에 추가하는 기호	예시	설명
문자열	\$	A\$, NAME\$	최대 255개의 문자 저장 가능
정수	%	A%, B%	-32767에서 32767 사이의 값 저장 가능
실수	없음	X, Y	소수점 이하 9자리, -10^{-38} 에서 10^{38} 까지

표 4. AppleSoft BASIC의 변수 요약표

KoBASIC에서는 실수, 정수, 문자열 변수를 선언할 때 변수 이름 뒤에 별도의 기호를 붙이지 않아도 된다. 이는 Apple BASIC에서 문자열 변수를 선언할 때 변수명 뒤에 \$ 기호를 붙여야 하는 것과 차별화된 KoBASIC의 특징이다.

KoBASIC에서는 데이터 타입에 따른 변수명 구분이 없으며, 변수명 뒤에 특

정 기호를 붙일 필요가 없다. 모든 변수는 하나의 통일된 방식으로 선언되며, 데이터 타입에 대한 구분은 코드 실행 시점에 KoBASIC의 내부 처리에 의해 자동으로 이루어진다.

4.1.11 Strings

AppleSoft BASIC에서 문자열은 따옴표로 묶인 문자들의 집합을 의미한다. 문자열은 텍스트 데이터를 다루는 데 사용되며, 숫자 변수와는 다르게 \$ 기호를 사용해 정의된다. 문자열은 "BILL", "APPLE"과 같이 따옴표로 묶어서 표현한다. 문자열 변수는 \$ 기호를 사용해 정의되며, 예를 들어 A\$ = "Hello"처럼 문자열 값을 변수에 할당할 수 있다.

KoBASIC에서는 AppleSoft BASIC과 비교하여 문자열 처리가 더 간편하게 이루어진다. AppleSoft BASIC에서는 문자열 변수에 \$ 기호를 붙여 사용해야 하지만, KoBASIC에서는 이와 달리 문자열을 그냥 변수로 설정하고 출력할 수 있다.



그림 36. KoBASIC을 이용한 문자열 예제

이 차이점 덕분에 KoBASIC에서는 문자열 변수를 다루는 방식이 더 직관적이고 간편하다. 이러한 방식은 기존 AppleSoft BASIC의 규칙에서 벗어나지만, 사용자의 편의성을 높이는 방향으로 개선되었다.

4.1.12 Evaluating Expressions

AppleSoft BASIC에서 수식을 평가할 때, 연산자 우선순위에 따라 연산이 수행된다. 연산자는 특정 우선순위를 가지며, 우선순위가 높은 연산자가 먼저 실행된다. 우선순위가 같은 연산자들끼리는 왼쪽에서 오른쪽 순서대로 우선순위가 정해진다.

KoBASIC에서 비교 연산은 Apple BASIC과 동일한 방식으로 동작한다. 즉, 변수나 값 간의 크기 비교나 동등성 비교를 위한 연산이 기본적으로 지원된다. 이와 더불어, 비트 연산도 지원하여 논리적 연산을 처리할 수 있다.

Applesoft BASIC Programming Reference Manual에 수록된 다양한 예제를 KoBASIC으로 실행한 결과, 기본적인 프로그래밍 기능을 비롯하여 조건문, 반복문, 배열, 문자열 처리 등 교육적으로 중요한 개념들이 KoBASIC에서 충분히 구현 가능한 것으로 나타났다. 이 과정에서 그래픽과 부동소수점 연산과 관련된 예제는 KoBASIC에서 구현하지 않아 테스트하지 못했지만, 그 외의 기능들은 정상적으로 작동하는 것을 확인할 수 있었다. 이러한 테스트는 KoBASIC이 프로그래밍 학습을 위한 실용적인 도구로서의 잠재력을 가지고 있음을 뒷받침하며, 한국어 사용자를 위한 프로그래밍 언어로서 학습용으로 적합하다는 결론을 도출할 수 있었다.

5. 멘토 의견서 반영

5.1 멘토 의견서 반영

멘토 의견서에 작성된 내용을 토대로 긍정적으로 검토하신 부분을 더 보강하고, 제시하신 의견을 반영하고자 하였다.

의견1 : 프로그래밍 결과가 콘솔로만 나오는 게 아니라 GUI 구현을 가능하게 한다면 학생들이 추상적인 것보다 구체적으로 결과물을 볼 수 있어 수업 콘텐츠가 훨씬 다양해질 수 있다. Scratch 나 엔트리도 GUI를 제어해서 여러 스토리라인이나 게임을 구현할 수 있으므로 이랑 비슷하게 하거나 makecode처럼 마이크로 비트와 같은 임베디드 장치와 연결할 수 있어야 한다.

- 학생들이 프로그래밍 결과를 시각적으로 확인할 수 있도록, Qt 프레임워크를 사용하여 GUI를 구현하였다. 또한, 파이썬 변환 기능은 웹 환경에서도 지원되어, 학생들이 별도의 설치 과정 없이도 학습을 이어갈 수 있도록 하였다. 다만, Scratch나 엔트리처럼 스토리라인을 구현하거나, makecode처럼 마이크로 비트와 같은 임베디드 장치와의 연결 기능은 구현되지 않았다. 이는 해당 기능이 추가적인 하드웨어와의 연동 및 통합 테스트를 해야 하여, 현재 프로젝트의 주요 개발 범위를 넘어서는 부분이었다. 이러한 기능들은 향후 연구에서 추가로 고려해 학습의 확장성을 높일 계획이다.

의견2 : 학생들이 사용하기 쉽도록 UI / UX가 학생 친화적이면 좋을 것 같다. 단순한 프로그래밍이 아니라 캐릭터를 움직인다는 등의 가시적인 것이 있으면 좋을 것 같다.

- 사용자가 코드를 작성할 때 편의성을 높이기 위해, 라인 넘버링과 문법 강조 기능을 추가하여 보다 직관적인 코딩 환경을 제공하였다. 이러한 기능들은 학생들이 코드 작성 중에 발생할 수 있는 오류를 쉽게 식별하고 수정할 수 있도록 돕는다. 캐릭터를 움직이는 시각적인 피드백 기능은 구현되지 않았지만, 이는 그래픽 및 애니메이션 요소가 추가로 요구되어, 현재 프로젝트의 중점이었던 프로그래밍 환경 개선 및 학습 지원과는 다소 다른 기술적 접근이 필요했다. 향후에는 이러한 시각적 요소도 추가하여, 학생들이 더 흥미롭게 학습할 수 있는 환경을 조성할 수 있도록 노력할 계획이다.

의견3 : Python 변환을 할 때 클래스와 함수 개념보다는 기초적인 프로그래밍 문법에 집중해서 개발하는 것이 좋아 보인다.

- 기초적인 프로그래밍 문법에 집중하여 설계를 진행하고자 했으나, BASIC과 Python 간의 문법적 차이로 인해 언어 변환 과정이 예상보다 복잡해졌다.

두 언어는 제어 구조, 함수 호출 방식 등에서 큰 차이가 있어, 단순한 변환보다는 이러한 차이를 조정하는 과정이 필요했다. 그 결과, 언어 간 변환이 다소 복잡하게 이루어졌으나, 이는 사용자들이 더 자연스럽게 Python 환경에 적응할 수 있도록 하기 위한 필수적인 과정이었다. 향후에는 이러한 차이를 더 쉽게 조정할 방안을 모색하여, 변환 과정을 개선할 계획이다.

6. 결론 및 향후 연구 방향

이번 연구에서는 BASIC 인터프리터를 한글로 구현하여 프로그래밍 초심자들이 더욱 친숙하게 코딩 개념을 접할 수 있는 환경을 제공하고자 하였다. 한글 명령어의 도입을 통해 사용자들은 모국어를 통해 보다 직관적으로 프로그래밍 언어의 구조와 동작 방식을 이해할 수 있을 것으로 보이며, 이는 학습 효율을 높이는 데 긍정적인 영향을 미칠 것으로 전망된다. 특히, 한글화된 코드를 통해 프로그래밍 교육에서 언어적 장벽을 낮추자는 목표에 집중하였다.

향후 연구로는 한글화된 표준 라이브러리를 설계하고 주요 함수와 키워드를 한국어로 제공하여, 프로그래머들이 더 쉽게 코드 작성과 디버깅을 수행할 수 있도록 할 계획이다. 또한 한글화된 BASIC 언어를 교육 현장에서 사용하여 학습자의 이해도와 학습 효과를 측정하고, 이를 바탕으로 교육적 유용성을 검증할 예정이다.

이번 연구와 향후 계획을 통해 한글 사용자가 더 쉽게 프로그래밍 언어를 접하고 배울 수 있는 환경을 제공함으로써, 더 많은 이들이 프로그래밍에 도전할 수 있도록 돕고자 한다.

7. 참고 문헌

- [1]Wikipedia contributors. "Non-English-based Programming Languages." Wikipedia, The Free Encyclopedia, August 2024, https://en.wikipedia.org/wiki/Non-English-based_programming_languages.
- [2]Anonymous. "Python in Education." Python.org, August 2024, <https://www.python.org/community/sigs/edu-sig/>.
- [3]Chong-sun Hwang, Yoo-hun Won, & Ho-young Kwak. "Design and Implementation of HANGUL BASIC Language." Journal of the Korea Information Science Society, 12(1), 52-59, 1985.
- [4]Ga-Young Kim, Seung-Wan Jeong, & Kwang-Hoon Choi. "A Study on the Design and Implementation of a Small Basic Program Interpreter." Proceedings of the Korean Institute of Information Scientists and Engineers Conference, pp. 1827-1829, 2016.
- [5]Jun-Seok Cheon, Do-Hoon Kang, Geon-Woo Kim, & Gyun Woo. "A Concise Korean Programming Language ‘Sprout’." Journal of KIISE, Vol. 42, No. 4, pp. 496-503, April 2015.
- [6]Helmets, C. "An Apple to Byte." BYTE, Vol. 3, No. 3, pp. 18-46, 1978.
- [7]Severance, Charles. "Guido van Rossum: The Early Years of Python." Computer, Vol. 48, No. 2, pp. 7-9, 2015.