

2024년 전기 졸업과제 중간보고서

Python 변환을 지원하는, 한글
프로그래밍 언어 KoBASIC의 구현
금정골사과

분과명 B

팀명 금정골사과

지도교수 우균 교수님

정보컴퓨터공학부 201624543 이석원

정보컴퓨터공학부 202155574 유수민

경영학과 202043154 김정한

CONTENTS

1. 과제 선정 배경 및 목표	2
2. 요구조건 및 제약 사항 수정사항	2
3. 설계 상세화 및 변경 내역	3
4. 갱신된 과제 추진 계획	4
5. 진척도	4
6. 과제 수행 내용 및 중간 결과	5
7. 참고 문헌	12

1. 과제 선정 배경 및 목표

현대 사회에서 컴퓨터 프로그램은 필수적인 요소로 자리 잡았다. 프로그램을 구현하는 도구인 프로그래밍 언어는 여러 가지 종류가 있지만, 대부분이 영어로 되어 있어 영어에 익숙하지 않은 사람들이 프로그래밍을 배우는 데 어려움이 있다. 이 문제는 특히 프로그래밍 교육 분야에서 두드러진다. 비영어 기반 프로그래밍 언어는 영어를 모국어로 사용하지 않는 사람들에게 프로그래밍을 쉽게 접근할 수 있도록 돕는다[1].

따라서 본 연구는 한글을 사용하는 초보자들이 쉽게 접근할 수 있는 프로그래밍 언어를 제공하고자 한다. 이를 위해 교육용 프로그래밍 언어인 BASIC을 기반으로 한 한글 프로그래밍 언어인 KoBASIC을 제안한다. KoBASIC은 명령어와 키워드를 한글로 표현하여서 영어에 대한 부담을 줄인다. 또 초보자들이 쉽게 이해하고 사용할 수 있도록 간단한 문법과 구조를 제공한다. 이러한 특징을 통해 KoBASIC은 한글 사용자들이 프로그래밍을 쉽게 배우고 이해할 수 있도록 도와줄 수 있을 것이다. 이는 프로그래밍 교육의 접근성을 높이고, 더 많은 사람이 프로그래밍의 기초를 배울 수 있는 기회를 제공할 것이다.

```

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. 요구조건 및 제약 수정사항

이번 졸업과제의 목표 및 요구조건은 한국어로 프로그래밍할 수 있는 '코베이직(KoBASIC)' 언어를 개발하는 것이다. 이 언어는 BASIC 문법을 기반으로 하여 초보자도 쉽게 접근할 수 있도록 설계되었다. 특히, KoBASIC 코드를 Python 코드로 변환하는 기능을 포함하여 프로그래밍 언어의 활용도를 높이려 한다. Python은 그 교육적 가치로 인해 널리 사용되며, KoBASIC과 같은 초보자 친화적 언어의 교육적 도입을 지지하는 사례를 제공한다[2]. KoBASIC의 주된 기능으로는 변수 선언, 산술 연산, 조건문, 반복문, 흐름 제어, 함수 및 주석 등이 있다.

기술적인 제약 사항으로는 한글 변수명과 함수명을 처리하기 위한 Unicode 문자열 처리와 키보드 입력 및 출력을 위한 문자 인코딩 문제 해결이 필요했다. 또한, GUI 인터페이스 구현을 위해 C++ 와 Qt를 사용하였으며, 토큰 분석, 구문 분석, 의미분석 등 컴파일러 기능을 구현해야 했다.

추가로 요구조건과 제약 사항에 따른 수정사항이 생겼다. 우선 BASIC에는 있고 Python에는 없는 Goto, Gosub와 같은 기능을 Python에서 동일하게 구현하기 위해 수정이 필요했다. 이를 위해 Python에서는 일반적으로 사용되지 않는 제어 흐름을 함수를 활용하여 구현하였다. 또, GUI 인터페이스에 대한 요구조건도 수정되었다. 초기에는 기본적인 코드 편집기 기능만 요구되었으나, 개발 과정에서 라인 번호 표시, 코드 강조 등 추가적인 편의 기능을 구현하기로 하였다. 이는 사용자 경험을 개선하는 데 중요한 역할을 한다. 마지막으로, Python 코드 변환 기능을 웹에서도 사용할 수 있도록 추가적으로 구현하였다. 이를 통해 사용자는 웹 브라우저를 통해서도 KoBASIC 코드를 Python 코드로 변환할 수 있게 된다.

3. 설계 상세화 및 변경 내역

1) 전체 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. 과제를 수행하면서 블록 변환기를 구현할지는 아직 보류 중이다.

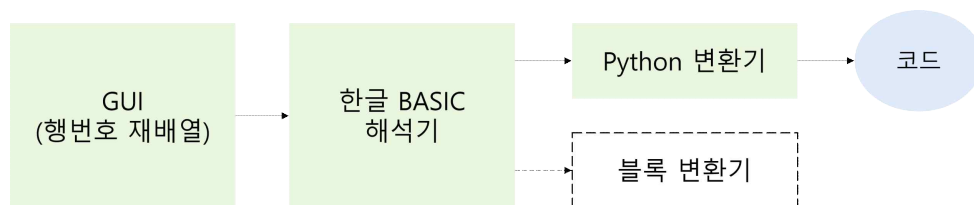


그림 7. KoBASIC 실행 구조

2) 한글 BASIC 해석기

KoBASIC은 BASIC의 문법은 유지하되, 키워드와 변수명 등을 한글로 제공하는 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. 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. 의미분석은 프로그램의 의미를 이해하고 검증하는 단계이다. 변수의 선언과 사용 등의 검사가 이루어지며, 코드의 의미적 일관성을 보장한다. 아래 그림은 KoBASIC의 실행 구조를 자세히 나타낸다.

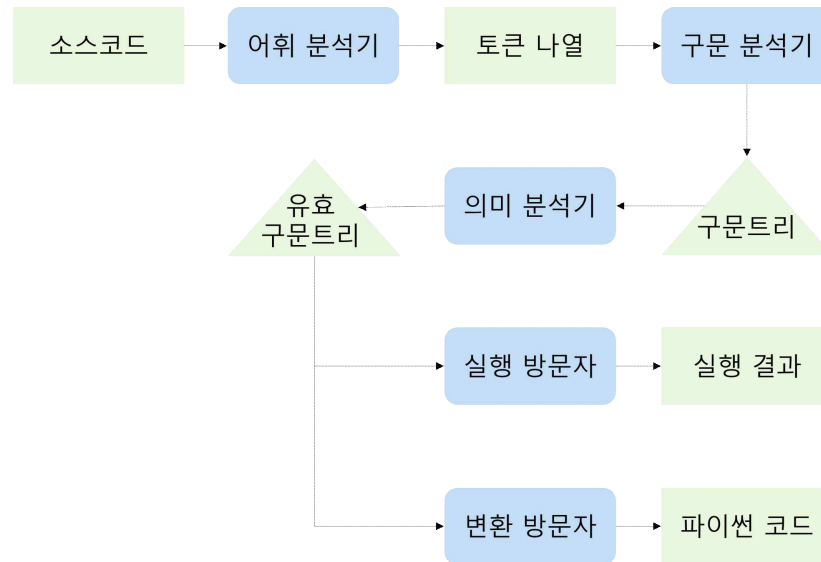


그림 9. KoBASIC 실행 구조

추가적으로, KoBASIC 코드를 Python 코드로 변환하는 모듈을 설계하였다. 이 모듈은 KoBASIC으로 작성된 코드를 입력받아 Python 코드로 변환하는 기능을 수행한다. 이를 통해 KoBASIC 사용자는 학습한 내용을 실무에서도 활용할 수 있다.

초기 설계 단계에서는 다음과 같은 사항에 중점을 두었다. 우선 BASIC 문법을 유지하면서 한국어로 프로그래밍할 수 있는 환경을 제공하는 데 중점을 두었다. 이를 위해 Qt를 사용하여 사용자 인터페이스를 구현하였고, SyntaxHighlighter 클래스를 통해 코드의 가독성을 높였다. 또한, 다양한 제어문과 연산자를 지원하도록 설계되었다.

4. 갱신된 과제 추진 계획

착수보고서 작성 당시에는 고려하지 못했던 여러 사전 처리 작업을 새롭게 정의하였으며, 과제 수행 과정 중 발생한 문제를 해결하기 위한 작업이 추가되었다. 초기 목표는 한국어로 프로그래밍할 수 있는 'KoBASIC' 언어를 개발하고, 이를 Python 코드로 변환할 수 있도록 하는 것이었다. 이 과정에서 Python 코드 변환 기능을 더욱 구체화하고, GUI 인터페이스에 대한 요구사항을 추가하는 등의 수정이 이루어졌다.

큰 관점에서 봤을 때, 처음 계획했던 추진 계획에서 크게 벗어나지 않았다. 초기 목표를 빠르게 달성하고, 이후 피드백을 받아 기능을 개선해 나가는 방향으로 계획을 수정하였다. 이후 오류 문구 추가 부분에 시간을 할애할 예정이다.

5. 진척도

오류와 난점들을 수정하면서 각 구성원의 해야 할 일들이 조금씩 빠지고 추가되었다. 현재는 주요 작업이 성공적으로 진행되고 있으며, 다음과 같은 진척도가 이루어졌다.

현재 BASIC 한글 지원 구현 작업은 성공적으로 완료되었으며, 한글 키워드 및 함수명을 추가하

여 한국어로 프로그래밍할 수 있는 환경을 조성했다. 또 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 진행되고 있다. 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. 현재는 KoBASIC과 Python 변환 기능의 품질 향상을 위해 데이터 학습을 진행하고 있으며, 사용자 피드백을 반영하여 인터페이스와 기능을 지속해서 개선하고 있다. 표 1은 수정된 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. 수정된 추진 일정표

6. 과제 수행 내용 및 중간 결과

1) 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 정렬 프로그램과 그 결과를 나타낸 것이다.

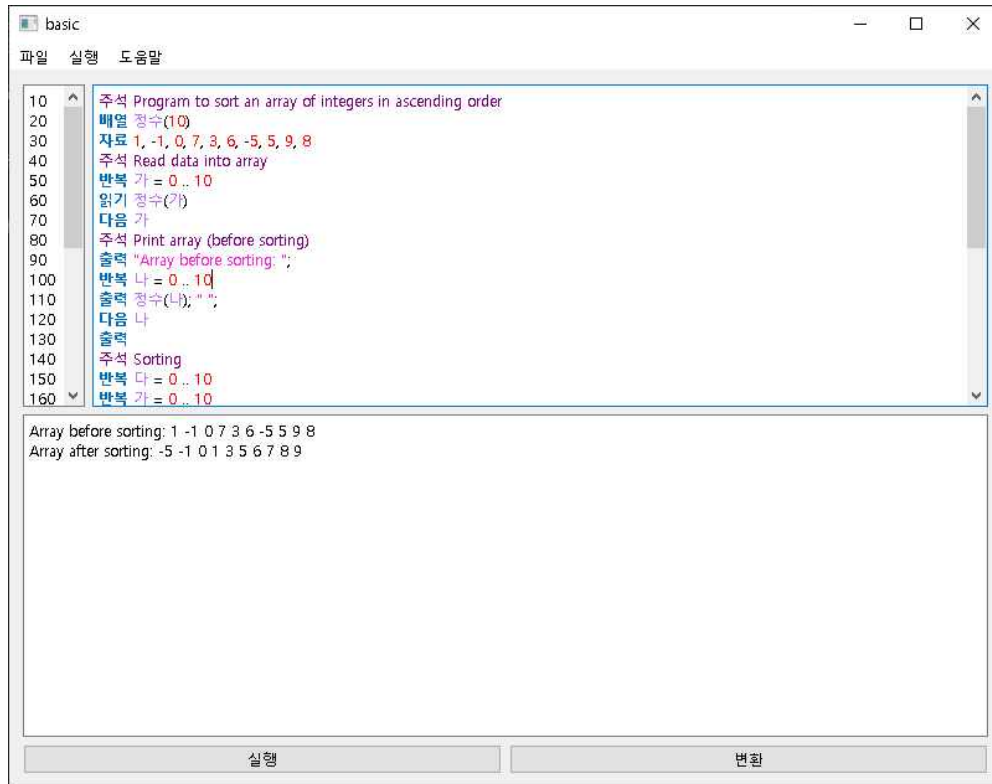


그림 8. KoBASIC을 이용한 정렬 예제

KoBASIC UI 상단에는 코드 작성 부분이 있으며, 하단에는 출력 부분이 있다. 코드 작성은 한글로 할 수 있으며, 코드의 가독성을 위해 문법 강조 기능을 적용하였다. 또 라인 번호를 따로 분리하여서 리넘버링 하고자 하면 순서대로 정렬해주는 기능을 추가로 구현하였다. 코드 작성 후 실행 버튼을 누르면 실행 결과가 아래에 출력된다.

추가적으로, KoBASIC 코드를 Python 코드로 변환하는 기능은 변환방문자 클래스를 통해 구현되었다. 이 클래스는 크게 세 가지 연산을 수행하는 메소드로 이루어지는데 한글 BASIC 해석기에서 생성한 유효 구문 트리를 순회하면서 Python 코드를 생성하고, 생성된 코드를 알맞은 구조로 변환하여 저장한 후, 최종 코드를 GUI 프로그램에 출력한다. 정리하면, KoBASIC으로 작성한 코드를 바로 실행하려는 사용자는 실행 방문자 모듈을 통해 위 그림과 같이 생성된 실행 결과를 얻을 수 있다.

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 차

단되기 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.

2) 한글 키워드, 함수명 추가

설계 과정에서 다음과 같은 주요 변경 사항이 있었다. 우선 한글 키워드 및 함수명이 추가되었다. 한국어로 프로그래밍할 수 있도록 기본적인 명령어와 함수명을 한글로 추가하였다. 예를 들어, "PRINT"를 "출력"으로, "IF"를 "조건"으로 사용할 수 있게 하였다.

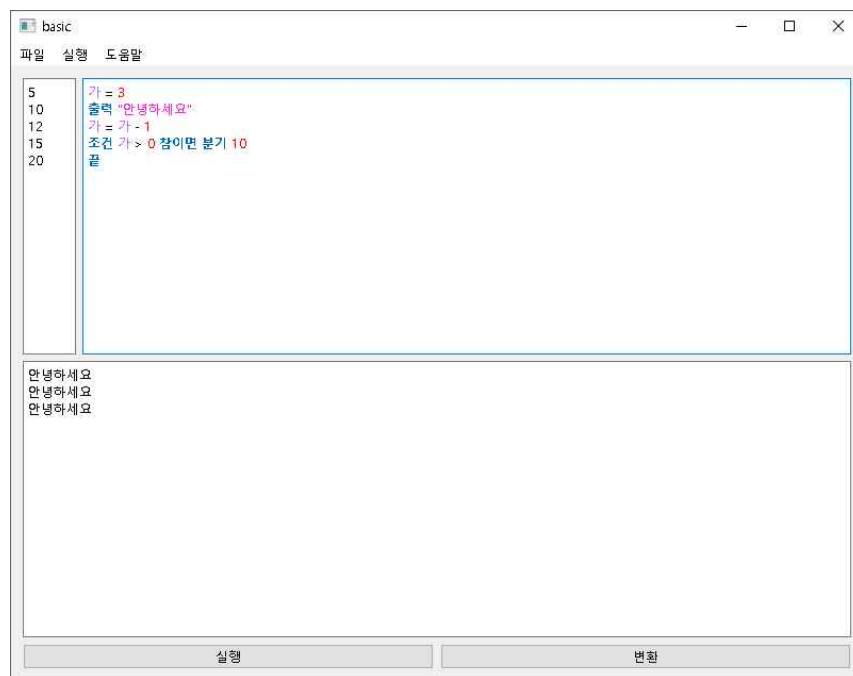


그림 3. 한글 키워드 명령어

3) 사용자 인터페이스 개선

또 사용자 인터페이스를 개선하였다. 코드 편집기에 라인 번호 표시, 구문 강조 기능을 추가하여 사용자가 편리하게 코드를 작성하고 디버깅할 수 있도록 하였다. 이때 라인 번호 리넘버링 기능을 추가하였고, 이를 통해 라인 번호를 10단위로 정렬하여 코드를 더 체계적이고 정돈된 방식으로 표시할 수 있도록 하였다.

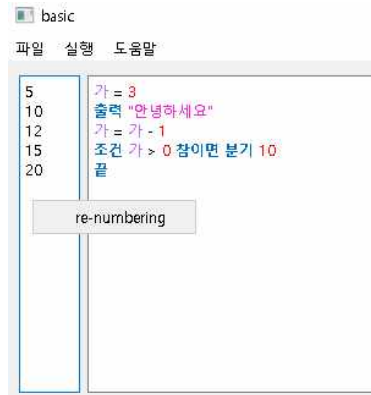


그림 4. Re-numbering 전



그림 5. Re-numbering 후

4) Python 변환기

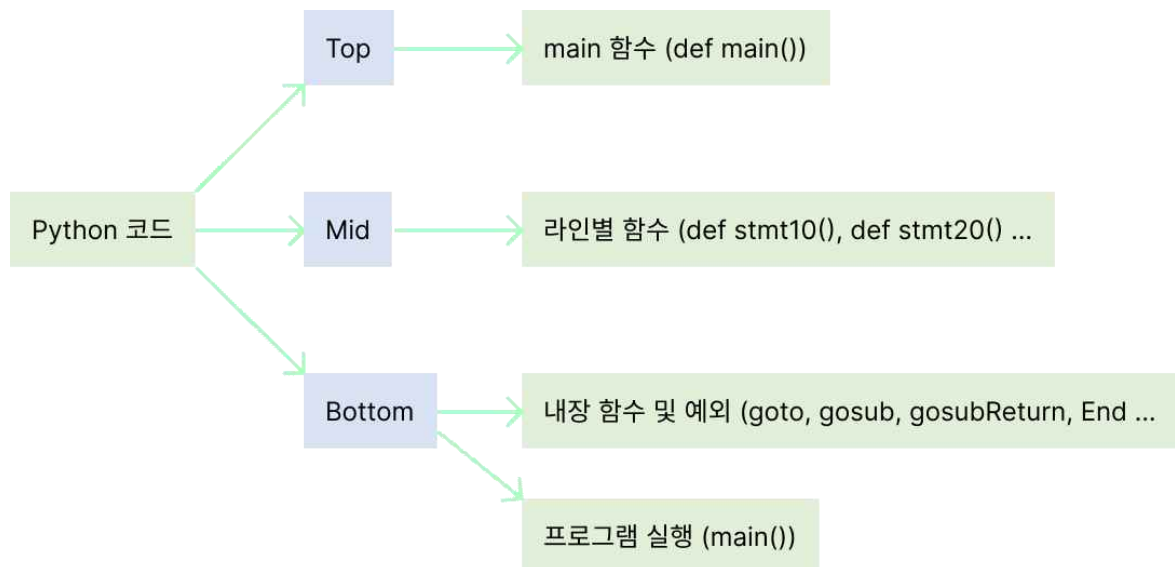
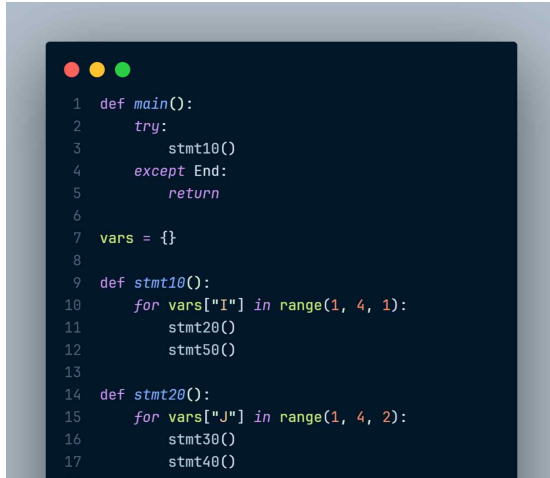


그림 10. Python 변환 구조

KoBASIC을 변환한 Python 코드는 위와 같은 구조로 작성되었다. 크게 Top, Mid, Bottom으로 나뉘는데, 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 예제를 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()

```

그림 11. 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 12. 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 13. 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 st

ack이 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 14. 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 15. 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
```

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.

5) Python 변환 웹 인터페이스 구현

KoBASIC은 현장에서 활용된다면 교육 목적으로 사용될 가능성이 크다. 따라서 사용자들이 쉽

게 사용할 수 있도록 접근성을 높이는 것이 중요한 목표 중 하나이다. 이러한 필요성에 따라 추가적으로 KoBASIC 코드를 Python 코드로 변환해주는 웹 인터페이스를 구현했다.



그림 6. 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, Input()을 처리할 수 있도록 만드는 과정이 까다롭고(Input을 만날 때마다 서버와 클라이언트의 소통이 필요하므로), 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://kimjeonghan0>

625.github.io/kobasic/에 있다.

8. 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/>.