C언어 스타일의 Python 클래스 선언 정렬 방법에 대한 연구

박관익, 노범석, 백승호
LIG넥스원
gwanik.park@lignex1.com, beomseok.noh@lignex1.com, seungho.baek.@lignex1.com

A Study on Python Class Declaration Sorting in C Language Style

Park Gwan Ik, Noh Beom Seok, Baek Seung Ho
LIGNex1

요 약

최근 Python 언어의 활용이 급증함에 따라, 국방 소프트웨어 분야에서 기존에 C언어로 작성된 코드를 Python으로 변환하려는 수요가 증가하고 있다. 특히 ctypes 모듈을 활용하여 C언어 스타일의 구조체를 Python에서 재현할 경우, 클래스 간 의존성으로 인해 선언 순서가 코드의 정확한 실행에 큰 영향을 미치게 된다. 본 논문에서는 클래스 간 의존 관계를 자동으로 분석하고, 이를 기반으로 클래스 선언을 올바른 순서로 정렬하는 알고리즘을 제안한다. 제안된 기법은 의존성 그래프를 구성하고, 큐 기반으로 정렬 순서를 도출함으로써, 수작업 정렬의 오류를 줄이고 코드 변환의 효율성이 향상된다. 향후에는 GUI를 포함한 자동화 시스템으로 확장하여 사용자 편의성을 더욱 높일 수 있을 것으로 기대된다.

Ⅰ. 서 론

최근 몇 년간 Python의 간결성과 생산성 향상, 그리고 다양한 라이브러리 지원 덕분에 개발 속도를 높이고 유지보수성을 개선하려는 목적으로 C언어에서 Python으로의 코드 변환이 활발히 진행되고 있다.[1] 국방 분야에서는 높은 신뢰성과 보안성을 유지하면서도 빠른 개발과 업데이트할 수 있어야 하므로, C언어에서 Python으로의 전환이 중요한 연구 주제로 부상하고 있다.[2]

하지만 기존의 C언어에서 Python으로의 전환은 단순 치환이 아닌 구조적 변환을 요구하므로, 상호 운용성을 확보하기 위한 과정은 단순하지 않다. 특히 C언어 스타일의 ctypes 모듈을 사용할 때, 클래스 간 상호 참조가 있는 경우 클래스 선언 순서에 따라 메모리 매핑 결과가 달라지기 때문에 선언 순서의 일관성이 매우 중요한 문제로 부각된다.[3] 이는 대규모 코드를 수작업으로 이식하는 과정에서 상당한 시간과 노력을 요구하며, 구조체가 많아질수록 오류 가능성이 큰 문제가 있다.

본 논문에서는 이러한 문제점을 해결하기 위해, 클래스 간 상호 참조 관계에 따른 의존성을 분석해 자동으로 선언 순서를 정렬하는 알고리즘을 제안한다. 이를 통해 코드의 안정성을 높이고, 대규모 코드 변환 시 정합성을 높여 개발자의 작업 부담을 획기적으로 줄일 수 있음을 보여준다.

Ⅱ. 본론

본 논문에서는 크게 5가지 단계의 Python 클래스 정렬 방법을 제안한다. 먼저 입력할 Python 파일을 텍스트 파일로 변환한 후, 각 클래스 선언을 추출한다. 다음으로 클래스 간의 의존성을 분석하여 의존성 정보를 딕셔너리 형태로 저장한다. 이후, 큐(queue) 기반 알고리즘을 활용하여 클래스 간 의존성에 따라 선언 순서를 정렬하며, 정렬된 순서대로 클래스를 재정렬하며 새로운 Python 파일을 생성한다. 그림 1은 본 논문에서 제안하는 클래스 정렬 기법의 전체 흐름도를 나타낸다. 본 절에서는 그림 2(a)의 예시 클래스 파일을 바탕으로 제안된 각 단계를 설명한다.

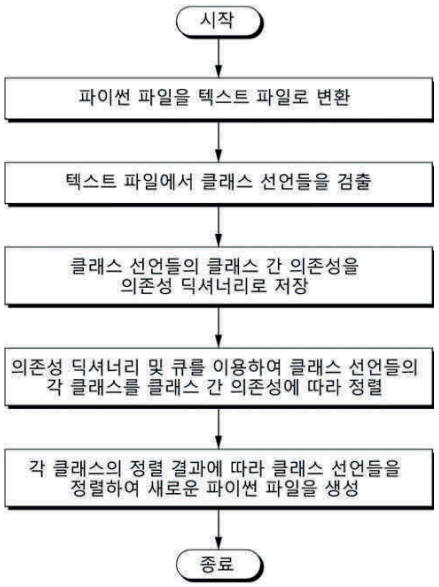

그림 1 제안 방법 흐름도

```
import ctypes

1 usage
class ClassA(ctypes.Union):
    _fields_ = [("member1", ctypes.c_int),
                ('member2', ClassC)]

class ClassC(ctypes.Structure):
    _fields_ = [("member", ctypes.c_int)]

class ClassD(ctypes.Structure):
    _fields_ = [('member', ClassB)]

class ClassB(ctypes.Structure):
    _fields_ = [("member", ClassA)]
```

(a)

```
import ctypes

1 usage
class ClassC(ctypes.Structure):
    _fields_ = [("member", ctypes.c_int)]
1 usage
class ClassA(ctypes.Union):
    _fields_ = [("member1", ctypes.c_int),
                ('member2', ClassC)]
1 usage
class ClassB(ctypes.Structure):
    _fields_ = [("member", ClassA)]

class ClassD(ctypes.Structure):
    _fields_ = [('member', ClassB)]
```

(b)

그림 2 본 논문의 예시 Python 클래스 파일
(a) 정렬 전 (b) 정렬 후

Ⅱ-1. 클래스 간 의존성 검출

클래스들의 상호 참조 관계를 분석하기 위해, Python 파일에서 추출된 각 클래스의 멤버 필드를 조사한다. 이때, 특정 클래스의 멤버 변수에 다른 클래스가 포함되어 있다면, 해당 관계를 의존성으로 간주한다.[4] 예를 들어, classA가 classB를 상속하거나 멤버 변수로 포함하고 있다면, classA는 classB에 의존한다고 정의한다.

이러한 의존성 관계는 Python의 딕셔너리 자료구조로 표현하며, Key는 참조되는 클래스명, Value는 해당 클래스를 참조하는 클래스로 저장된다. 표 1은 예시 클래스 파일에서 추출된 의존성 딕셔너리를 보여준다.

Key	Value
ClassC	ClassA
ClassB	ClassD
ClassA	ClassB

표 1 예시 클래스 파일의 의존성 딕셔너리 결과

Ⅱ-2. 의존성 기반 클래스 정렬

그림 3은 의존성 정보를 바탕으로 클래스의 선언 순서를 정렬하는 과정을 나타낸 흐름도이다. 먼저, 의존성 딕셔너리를 순회하면서 각 클래스의 진입차수(indegree)를 계산한다. 클래스의 진입차수는 다른 클래스들로부터 참조되는 횟수를 의미하며, 별도의 딕셔너리에 저장된다. 표 2는 예시 클래스들의 진입차수 계산 결과를 보여준다.

정렬 과정은 큐를 기반으로 동작한다. 진입차수가 0인 클래스를 큐에 먼저 삽입하고, 큐가 비어 있지 않는 동안 반복문을 수행한다. 큐의 맨 앞에 있는 클래스를 꺼내 결과 리스트에 저장한 뒤, 해당 클래스가 의존하고 있는 클래스들의 진입차수를 1씩 감소시킨다. 만약 특정 클래스의 진입차수가 0이 되면, 이는 해당 클래스가 더 이상 다른 클래스에 의해 참조되지 않음을 의미하며, 이 클래스를 큐에 추가한다.

이 과정을 모든 클래스가 정렬될 때까지 반복함으로써, 의존성 관계를 만족하는 정렬 순서를 도출할 수 있다. 그림 4와 그림 2(b)는 각각 예시 클래스 파일이 제안된 방법으로 정렬되는 과정 및 결과를 보여준다.

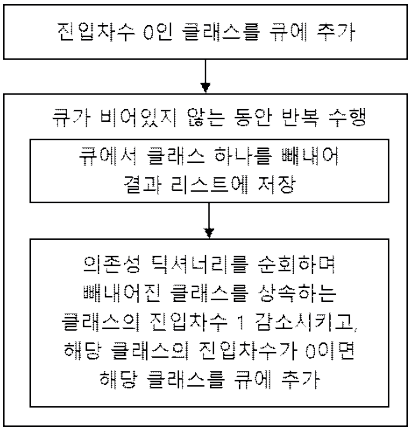

그림 3 진입차수를 활용한 의존성 기반 정렬 알고리즘 흐름도

Class	Indegree
ClassA	1
ClassB	1
ClassC	0
ClassD	1

표 2 예시 클래스 파일의 진입차수 결과

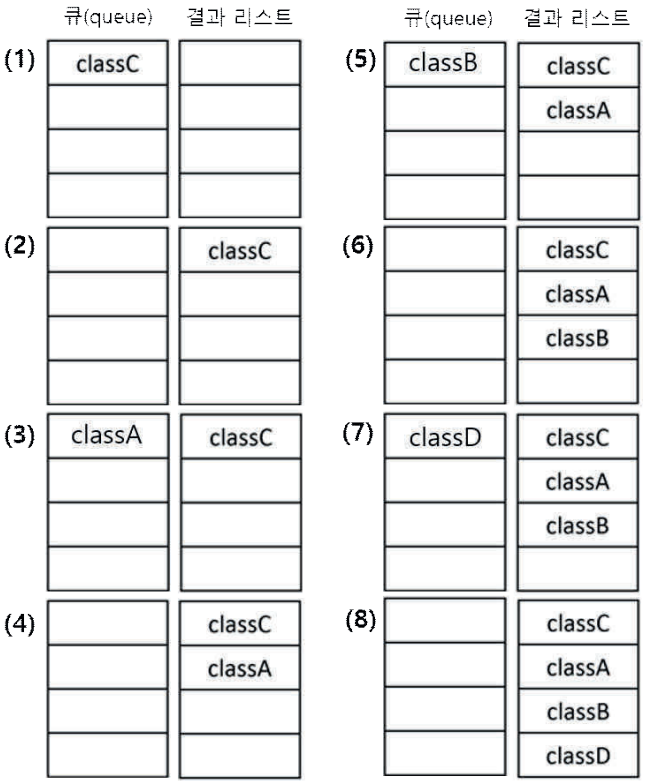

그림 4 예시 클래스 파일의 정렬 과정

Ⅲ. 결론

본 논문에서는 C언어 스타일인 ctypes 기반의 Python 코드에서, 클래스 간의 의존성에 따라 클래스 선언 순서를 자동으로 정렬하는 알고리즘을 제안하였다. 제안된 방법은 Python 파일 내 클래스 간 상호 참조 관계를 분석하고, 이를 토대로 안정적인 선언 순서를 도출함으로써, 구조체 수가 많은 대규모 코드 전환 과정에서도 효율적으로 대응할 수 있도록 한다.

이 기법은 수작업 정렬 과정에서 발생할 수 있는 오류를 예방하고, 클래스 선언의 정확도를 보장함으로써 개발자의 작업 시간을 줄이고 코드의 신뢰성을 향상하는 데 기여한다. 또한, 향후 사용자 인터페이스(GUI)를 추가하거나 자동화 수준을 더욱 높인 통합 시스템으로 확장할 경우, 비전문가도 손쉽게 사용할 수 있는 직관적인 도구로 발전시킬 수 있을 것으로 기대된다.

참 고 문 헌

[1] Van Rossum, G., & Drake, F. L. (2003). An introduction to Python (p. 115). Bristol: Network Theory Ltd..

[2] Fangohr, H. (2004). A comparison of C, MATLAB, and Python as teaching languages in engineering. In Computational Science-ICCS 2004: 4th International Conference, Kraków, Poland, June 6-9, 2004, Proceedings, Part IV 4 (pp. 1210-1217). Springer Berlin Heidelberg.

[3] Smith, K. W. (2015). Cython: A Guide for Python Programmers. " O'Reilly Media, Inc.".

[4] Orru, M., Tempero, E., Marchesi, M., & Tonelli, R. (2015, December). How do Python programs use inheritance? a replication study. In 2015 Asia-Pacific Software Engineering Conference (APSEC) (pp. 309-315). IEEE.