프로그래밍 학습을 위한 뮤턴트 기반의 실습 문항 생성기의 구조 설계

곽용섭○, 이성희, 이우진*
경북대학교 컴퓨터학부, *소프트웨어기술연구소
e-mail: iddqt@naver.com○, lee3229910@gmail.com, woojin@knu.ac.kr*

Design of Mutant-based Practical Test Problem Generator for Programming Education

Yong-Sub Kwak○, Sunghee Lee, Woo Jin Lee*
School of CSE & *SWRC, Kyungpook National University

요 약

프로그래밍 교육에서 실습교육은 소스 코드를 직접 작성해보는 과정을 통해 이론적인 지식을 보완할 수 있는 매우 중요한 과정이다. 따라서 대부분의 프로그래밍 교과과정은 실습교육을 포함하고 있다. 그러나 실습교육을 통해 학습 성취도를 평가하는 일은 시간과 비용이 많이 소모되는 작업이다. 그래서 많은 교육기관에서는 평가를 효율적으로 하기 위해 자동 평가 시스템을 운용하고 있다. 자동 평가 시스템은 학생들의 실습 결과를 정확하고 신속하게 평가하는데 효과적이다. 그러나 실습교육에 필요한 실습문항은 대부분의 경우 교사가 수작업으로 생성하며 이 과정에서 많은 인적·시간적 비용이 발생하게 된다. 이러한 문제를 해결하기 위해서 문항 생성을 자동화하려는 연구가 진행되고 있으나 아직까지 초기 단계이며 새로운 문항을 생성하지 못하는 등의 제약 사항이 많아 적용하기에 무리가 있다. 따라서 본 논문에서는 하나의 문항으로부터 다양한 문항들을 변형하여 생성할 수 있는 방법을 제안하고 이를 지원하는 프로그래밍 실습용 문항 생성기의 구조를 설계한다.

1. 서론

효과적인 학습을 위해 대부분의 프로그래밍 교육은 이론 교육과 함께 실습교육을 병행하며 진행한다. 실습교육은 학생들이 이론 교육만으로는 습득하기 힘든 실증적인 프로그래밍 지식을 깨우칠 수 있기 때문에 매우 중요한 과정이다. 실습교육에서 학생들은 일반적으로 교사가 제시한 요구사항에 부합하는 프로그램의 소스 코드를 작성하여 제출한다.

실습교육을 진행한 후 교사는 학생들이 제출한 프로그램의 소스 코드를 바탕으로 학생들의 학습 성취도를 평가한다. 평가 방식은 대부분 블랙박스 테스팅의 형식을 취하며 교사는 미리 준비한 테스트케이스를 이용해 평가한다. 그러나 교사가 수작업으로 평가할 경우 매우 많은 시간이 소모되며 업무 부담이 과중되고 실수로 인한 오채점 등으로 인해 채점 자체의 신뢰도가 흔들릴 가능성이 있다.

이러한 이유로 많은 프로그래밍 교육과정에서는 자동 평가 시스템을 운용하고 있다. 자동 평가 시스템은 학생들의 제출물을 채점하는데 필요한 시간을 효과적으로 단축하며, 채점 결과의 정확성과 객관성을 보장해준다. 반면 자동 평가 시스템은 평가에 필요한 시간을 단축시키고 평가의 신뢰도는 높일 수 있을지 모르나 실습교육에 사용되는 실습문항을 만드는데 필요한 작업을 단축시켜 주지는 않는다. 대부분의 경우 실습문항은 교사에 의해 수작업으로 만들어지며, 이때 평가과정과 마찬가지로 많은 시간과 비용이 필요하다.

따라서 실습문항을 만드는데 필요한 비용을 줄이기 위해 실습문항을 자동으로 생성할 수 있는 방법이 필요하다. 실습문항은 학생들에게 작성해야 하는 답안의 요구사항을 효과적으로 설명해야 하며 학생들이 제출한 프로그램을 채점할 수 있는 테스트케이스를 포함해야 한다.

본 논문에서는 소스 코드를 기반으로 뮤턴트를 생성해 프로그래밍 실습수업에서 사용할 수 있는 실습문항을 자동으로 변형하여 생성하는 방법을 제안한다. 소스 코드로부터 뮤턴트들을 생성한 후 뮤턴트 소스 코드와 테스트케이스를 다시 뮤테이션하여 다양한 실습문항들을 생성한다. 이러한 방법을 이용하면 하나의 소스 코드와 테스트케이스로부터 다수의 실습문항을 자동으로 생성할 수 있다.

2. 관련 연구

Richard A. DeMilli의 연구[2]는 뮤테이션 기반의 테스트 데이터 생성 방법을 제안하고 있다. 대수적 제약 조건을 기반으로 뮤턴트의 상대적 타당성을 분석한 뒤 테스트 데이터를 생성하는 방법을 제안하였다. 이 방법을 이용하면 다양한 학습 관점에 적합한 테스트 데이터를 자동으로 생성할 수 있는 장점이 있다. 그러나 테스트 데이터를 생성하기 위한 제약 조건을 정의해야하는 작업이 필요하고 새

로운 실습 문항을 자동으로 생성하지는 못한다.

Ihantola의 연구[3]는 심볼릭 실행 기법을 이용한 테스트 데이터 생성 방법으로, 학생들이 제출한 프로그램으로부터 테스트 데이터를 생성하는 방법을 제안하였다. 이 방법은 학생들이 제출한 프로그램마다 프로그램을 테스트할 수 있는 최적의 테스트 데이터를 자동으로 생성함으로써 교사가 미흡한 테스트 데이터를 준비하더라도 이를 보완해 줄 수 있다는 장점이 있으나 모든 학생들이 동일한 테스트 데이터를 사용하지 않는다는 점에서 채점의 객관성을 보장하기 힘들고 역시 새로운 실습문항을 생성하지는 못한다는 한계가 있다.

3. 실습문항 생성기의 구조 설계

본 논문에서는 실습문항 생성기를 C 언어 프로그래밍 실습교육을 가정하여 설계한다. 그리고 큰 수정 없이 Java 등 다른 프로그래밍 언어 실습교육에도 적용할 수 있도록 확장성을 고려하여 설계한다.

그림 1은 5개의 모듈로 구성된 실습문항 생성기의 구조를 나타낸다. 뮤테이션 모듈은 소스 코드와 테스트케이스로부터 실습 목적에 맞는 뮤테이션 연산자를 적용하여 뮤턴트 소스 코드와 테스트케이스를 생성하는 역할을 수행한다. 문항 생성 모듈은 소스 코드, 테스트케이스로부터 교사가 원하는 유형의 문항 생성 연산자를 적용해 실습문항을 생성한다. 뮤테이션 연산자 관리 모듈과 문항 생성 연산자 관리 모듈은 각 연산자를 실습문항 생성기에 추가하고 관리하는 역할을 한다. 문항 추출 모듈은 생성된 실습문항 정보를 외부 시스템에서 요구하는 형태로 변형하여 제공하기 위한 모듈이다.

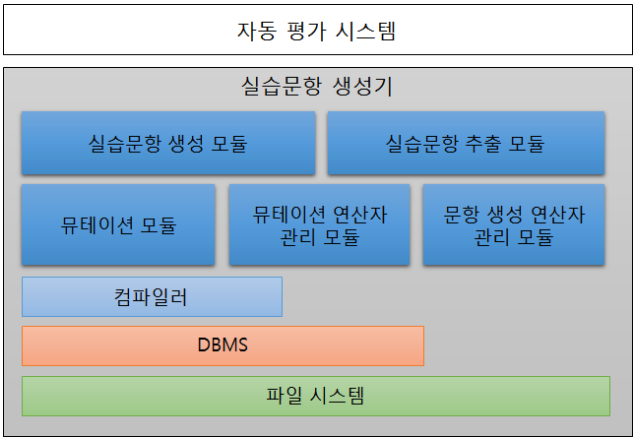

그림 1. 실습문항 생성기의 구조

그림 2는 본 논문에서 제시하는 문항 생성 절차를 나타내고 있다. 첫 번째 절차는 교사가 뮤턴트 생성의 재료가 되는 소스 코드와 테스트케이스를 실습문항 생성기에 입력하는 과정이다. 이 과정은 전체 생성 절차에서 교사가 소스 코드와 테스트케이스를 직접 준비해야 하는 유일한 과정이다. 일반적으로 실습교육에서는 학습 주제마다 핵심이 되는 소스 코드가 존재하므로 이러한 소스 코드와 테스트케이스를 준비하여 입력할 수 있다. 두 번째 단계는 첫 단계에서 입력한 소스 코드와 테스트케이스에 뮤테이션 연산자를 적용하여 뮤턴트를 생성하는 작업이다. 이때 학습 주제에 적합한 뮤턴트 소스 코드를 생성하도록 교사는 적절한 뮤테이션 연산자를 선정할 수 있다. 뮤테이션 모듈은 선별된 뮤테이션 연산자를 이용하여 다수의 뮤턴트 소스 코드를 생성하고, 빌드 및 실행 과정을 거쳐 각 소스 코드의 테스트케이스를 만든다. 이 때 부모 소스 코드의 테스트케이스를 사용하여 뮤턴트의 테스트케이스를 생성하기 때문에 교사의 추가적인 작업이 필요하지 않다. 문항 생성 절차의 마지막 단계는 생성된 뮤턴트 소스 코드와 테스트케이스로부터 문항 생성 연산자를 적용하여 문항을 생성하는 단계이다. 이 과정에서 생성되는 실습문항의 유형과 내용은 전적으로 적용된 문항 생성 연산자의 구현에 따라 결정된다. 따라서 실습 목적에 따라 뮤테이션 과정과 동일하게 교사가 적용할 문항 생성 연산자를 선별하여 사용한다.

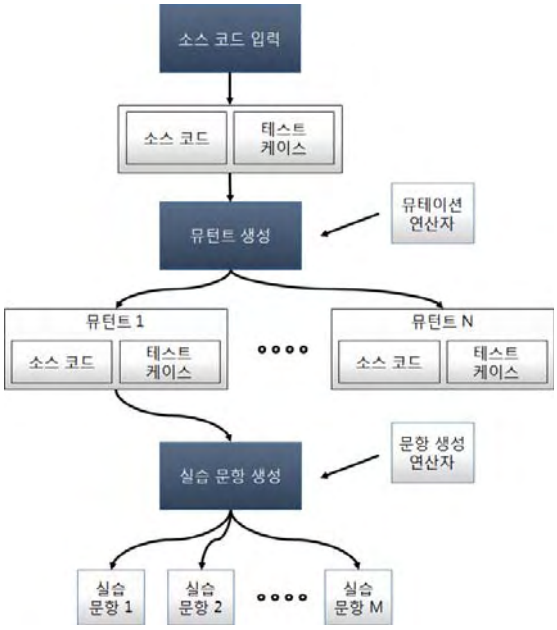

그림 2. 문항 생성 절차

그림 3은 실습문항 생성기에서 사용하는 데이터베이스의 테이블 구조를 나타낸다. 실습문항 생성기는 하나의 소스 코드, 테스트케이스 쌍으로부터 다수의 뮤턴트를 생성한다. 이 때 생성된 소스 코드와 테스트케이스는 파일 형태로 저장되며 소스 코드의 ID가 포함된 경로에 저장된다. 뮤턴트의 소스 코드들은 부모의 소스 코드에 대한 참조를 가지고 있으며 이를 통해 뮤테이션 과정을 역추적할 수 있도록 설계한다. 실습문항이 생성되면 문항 정보들은 파일의 형태로 문항 ID가 포함된 경로에 저장된다. 생성된 문항들은 학생들이 제출한 답안을 채점할 수 있도록 소스 코드와 테스트케이스에 대한 참조를 가진다. 마지막으로 뮤테이션 연산자 및 문항 생성 연산자들은 그룹화하여 필요한 경우 재활용하기 쉽도록 설계한다.

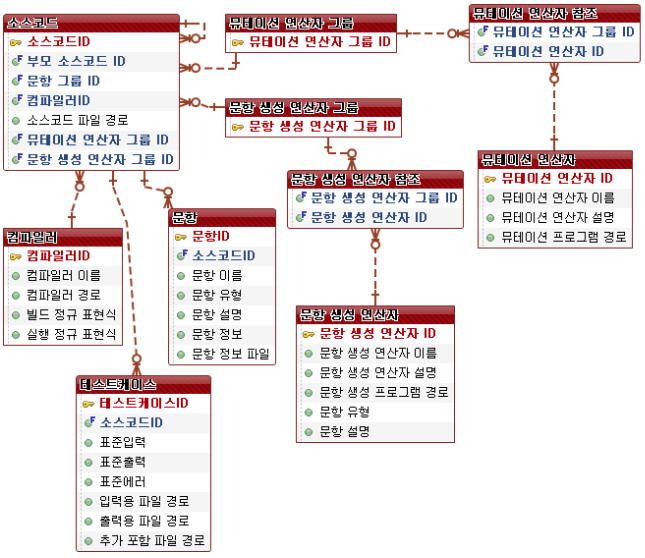

그림 3. 실습문항 생성기의 DB 테이블 구조

3.1 뮤테이션 연산자와 문항 생성 연산자

뮤테이션 연산자는 소스 코드와 테스트케이스를 입력으로 받아 뮤턴트를 생성하는 별도의 프로그램으로 정의할 수 있다. 마찬가지로 문항 생성 연산자는 소스 코드와 테스트케이스를 입력으로 받아 문항 정보를 출력하는 프로그램으로 정의할 수 있다.

뮤테이션 연산자와 문항 생성 연산자는 실습문항 생성기에 의해 실행되며 필요한 정보를 명령어 인수로 전달받아 작업하게 된다. 이때 필요한 정보는 소스 코드와 테스트케이스 등 크기가 큰 정보이기 때문에 명령어 인수로 곧바로 전달하지는 않고, 파일 형태로 파일 시스템에 저장하여 파일의 경로를 전달하는 방식을 취한다. 문항 생성 연산자는 하나의 문항유형에 대한 문항 생성을 담당한다. 문항 생성 연산자는 생성한 문항을 파일 형태로 저장하고 파일의 경로를 출력한다. 표 1은 실습문항 생성기에서 적용할 수 있는 뮤테이션 연산자의 예를 보여준다.

표 1. 뮤테이션 연산자

뮤테이션 연산자 유형	뮤테이션 연산자
조건문 수정	if 문 주석 처리
	조건식의 산술 연산자를 교체 (+,-,*,/)
	조건식의 관계 연산자를 교체 (<,>,<=,>=,==,!=)
	조건식의 논리 연산자 교체 (||,&&)
	조건식의 결과 반전 (!)
반복문 수정	초기화구문의 리터럴을 임의값으로 수정
	반복 조건의 관계 연산자 교체 (<,>,<=,>=,==,!=)
	반복 조건의 논리 연산자 교체 (||,&&)
	증감연산문의 연산자 교체 (++,--)
리터럴 값 수정	정수 리터럴을 임의의 값으로 수정
	실수 리터럴을 임의의 값으로 수정

3.2 뮤턴트 기반 실습 문항 생성 적용 사례

표 2는 교사가 실습문항 생성기에 입력하는 소스 코드와 테스트케이스의 예시를 나타낸 것이다. 표 3은 표 2의 소스 코드와 테스트케이스에 조건식의 관계 연산자를 교체하는 뮤테이션 연산자를 적용하여 생성된 뮤턴트 중 하나를 나타낸 것이다. 조건문의 관계연산자를 >에서 <=로 교체되었다. 뮤턴트 코드가 생성되면 부모 테스트케이스의 입력값을 이용해 뮤턴트의 테스트케이스를 구할 수 있다. 표 4는 표 2의 소스 코드에 if문을 주석 처리하는 뮤테이션 연산자가 적용된 또다른 뮤턴트 예시를 보여준다.

표 2. 뮤턴트 생성 전 소스 코드 및 테스트케이스

소스 코드

```
#include <stdio.h>
int main() {
    int x, y;
    scanf("%d%d", &x, &y );
    if( x + y > 0 )
        printf("sum=%d\n", x+y );
    else
        printf("mult=%d\n", x*y );
    return 0;
}
```

테스트케이스

TC #	input	output
1	8 -2	sum=6
2	-10 5	mult=-50
3	9 2	sum=11
4	-3 -7	mult=21

표 3. 생성된 뮤턴트 소스 코드와 테스트케이스 예시 1

소스 코드

```
#include <stdio.h>
int main() {
    int x, y;
    scanf("%d%d", &x, &y );
    if( x + y <= 0 )
        printf("sum=%d\n", x+y );
    else
        printf("mult=%d\n", x*y );
    return 0;
}
```

테스트케이스

TC #	input	output
1	8 -2	mult=-16
2	-10 5	sum=-5
3	9 2	mult=18
4	-3 -7	sum=-10

표 4. 생성된 뮤턴트 소스 코드와 테스트케이스 예시 2

소스 코드

```
#include <stdio.h>
int main() {
    int x, y;
    scanf("%d%d", &x, &y );
    // if( x + y > 0 )
        printf("sum=%d\n", x+y );
    // else
        printf("mult=%d\n", x*y );
    return 0;
}
```

테스트케이스

TC #	input	output
1	8 -2	sum=6 mult=-16
2	-10 5	sum=-5 mult=-50
3	9 2	sum=11 mult=18
4	-3 -7	sum=-10 mult=21

이렇게 생성된 뮤턴트 소스 코드와 테스트케이스를 바탕으로 실습 문항을 생성하는 다양한 유형의 문항 생성 연산자를 생각해볼 수 있다. 예를 들면 표 3의 뮤턴트를 바탕으로 소스 코드를 보고 출력결과를 예상해보는 유형의 실습 문항을 생각해볼 수 있다. 학생들은 소스 코드와 출력값이 제거된 테스트케이스를 보고 입력값에 따른 출력값을 작성하는 방법으로 실습하게 된다. 또는 뮤턴트 테스트케이스와 함께 뮤턴트 소스 코드가 아닌 부모 소스 코드를 학생에게 제시하고, 주어진 테스트케이스와 같은 실행 결과가 나오도록 부모 소스 코드를 수정하는 유형의 실습 문항도 생각해볼 수 있다.

4. 결론

본 논문에서 제안하는 실습문항 생성기는 프로그래밍 실습교육에서 사용하는 자동 평가 시스템을 위한 실습문항을 만드는데 필요한 비용을 줄이고자 하는 목적으로 설계되었다. 소스 코드와 테스트케이스는 자체만으로도 프로그램의 요구사항에 대한 충분한 단서를 제시할 수 있으며 따라서 이들을 이용하여 실습교육에 사용되는 실습문항을 만들 수 있을 것으로 생각된다. 뮤테이션 기법을 이용하여 뮤턴트들을 생성하고, 이를 다시 문항 생성에 이용함으로써 교사의 수작업을 최소화하면서 충분한 수의 문항을 만들 수 있다.

※ 본 논문은 2014년도 정부(교육부)의 재원으로 한국과학재단의 지원을 받아 수행된 기초연구사업(No. NRF-2014R1A1A2058733)으로 수행되었음.

참고문헌

[1] Rohaida Romli, S. Sulaiman, K. Zamli, "Automatic Programming Assessment and Test Data Generation," 2010 International Symposium in Information Technology (ITSim), vol. 3. IEEE, pp. 1186-1192, 2010.

[2] Richard A. DeMilli, A. Jefferson Offutt, "Constraint-based automatic test data generation," IEEE Transactions on Software, 1991.

[3] P. Ihantola, Automatic Test Data Generation for Programming Exercises with Symbolic Execution and Java PathFinder, Master Thesis of Helsinki University of Technology, 2006.

[4] C. Douce, "Automatic Test-based Assessment of Programming: A Review," Journal on Educational Resources in Computing, Vol. 5, Issue 3, 2006.

[5] L. Malmi, V. Karavirta, A. Korhonen, J. Nikander, O. Seppala, and P. Silvasti, "Visual Algorithm Simulation Exercise System with Automatic Assessment: TRAKLA2," Informatics in Education, Vol. 3, Issue 2, pp. 267-288, 2004.

[6] Mike Papadakis, Nicos Malevris, "Mutation based test case generation via a path selection strategy," Information and Software Technology, 2012.