Automatic On-line Generation of Student's Exercises in Teaching Programming

Danijel Radošević, Tihomir Orehovački, Zlatko Stapić
Faculty of Organization and Informatics
University of Zagreb
Pavlinska 2, 42000 Varaždin, Croatia
{danijel.radosevic, tihomir.orehovacki, zlatko.stapic}@foi.hr

Abstract. *Teaching programming faces some general teaching problems, but also confronts some specific problems such as understanding of programming concepts as well as algorithms for solving programming tasks. Our teaching experience with students at university beginner's level has shown that students often try to avoid understanding programming concepts by some "shortcuts", like learning program code by rote, copying programs from colleagues etc. In this paper we introduce automatic on-line generation of programming exercises with code examples for students. This enables high level personalization of student's programming tasks and makes avoiding of understanding concepts more difficult. Along with introduction of the on-line solution, some open questions about correctness of exercises, controlling the solutions and the whole teaching process are also discussed.*

Keywords. programming, teaching approaches, on-line generator

1 Introduction

Computer science students are faced to a task that is often very challenging – a task of learning programming. The main reasons for this are usually referred to be their lack of mathematical and informatics knowledge, undeveloped skills of abstract thinking and logical reasoning, lack of motivation and fear of programming [5][9][20]. In addition, a large number of students practice to learn programming through reading books or listening lectures passively. This results in learning to program by heart, without any understanding, which is opposite to a generally known theory that programming is a skill that can only be learned with a lot of active work.

In recent years intensive work has been done on developing specialized educational software to help students to understand the basic programming concepts and to develop problem solving skills. Mentioned software is usually installed in the classroom laboratory and used as a supplement to teaching process, but a time spent in active learning of a programming during the lectures or laboratory classes is not enough for the students to adopt all necessary knowledge and skills they need in order to solve more complex tasks. Therefore, the practice of introducing homework tasks in the form of small programming assignments that students should address and solve at home is becoming more usual. In that way, teachers get feedback and can more precisely aim their remaining lectures. In addition, for each exercise, teachers usually send feedback so each student could get a picture of a progress and learn from own mistakes.

The mentioned process of teaching programming demands much more effort from both sides and eventually becomes a challenging task for teachers too. At our faculty, courses related to teaching programming enroll more than 200 students a year. Since it is such a large number of students assigned to a relatively small number of teachers, they are faced to problem that they do not have enough time available to dedicate to each student, especially not in a described individualized approach. Finally, many other problems arise, from which the

most common are cheating and plagiarism, while teachers cannot be sure that the submitted solutions, are not copied and are genuine work by each student. These problems are even more expressed if all students are given unified programming problems.

Although there are specialized tools that deal with issues of plagiarism [14] and the automatic assessment systems (AA) [6], we tried to move forward and this paper presents an idea of generative approach in solving these problems. We introduce the system of personalized automatic generation of programming tasks that are created according to specific template but unique for each student. This approach, applied during the teaching process, dramatically reduces the likelihood of solutions being copied or downloaded from the Internet, and subsequently results in more active involvement from each student in order to create unique and own solution. The final result is more applicable knowledge in programming.

2 Related work

Today there are many tools used in the process of teaching programming, ranging from simple tools used only at universities and faculties at which they were developed to the commercial projects that are used in a number of institutions around the world. The above mentioned tools can be divided into two main categories: *automatic assessment systems* and *online compilers*. Best known representatives of each mentioned group will be presented in this chapter and their purpose and general functionalities will be briefly explained.

2.1 Automatic assessment systems

The field of automatic evaluation is huge and there are several different categorizations of existing systems. Carter et al. [6] divided the exercises into five basic categories: *multiple choice questions, programming assignments, visual answers, text answers* and *peer assessment*. However, it should be noted that the text answers are considered to be part of wider area of automatic evaluation of natural language, while the peer assessment is a part of computer-aided, not automated, assessment. Therefore, both categories are out of scope of this paper and will not be discussed in more detail.

Multiple choice questions are the simplest form of AA in which the assessment procedure is frequently embedded into the questions themselves. The most common form of a multiple choice question has four or more alternatives, of which at least one is correct. However, the number of correct, semi-correct and incorrect choices can vary, depending on the teachers' choice. Typically a student is rewarded with points for the correct and semi-correct choice, while an incorrect answer gives either zero or a negative number of points. The assessment procedure of multiple choice questions is very easy. It compares the student's answer to the correct one and according to the grading formula gives points. The simplicity of multiple choice questions has made them a very popular feature in learning management systems, such as Moodle and WebCT. However, in teaching programming, multiple choice questions can be useful only for the adoption of basic theoretical, but not for gaining practical skills in solving programming assignments.

The automatic assessment of *programming assignments* is the most usual example of AA in the field of computer science. This category includes all systems that automatically assess some or all aspects of computer programs. The earliest assessment systems, often referred as grading programs [8][17], were based on very simple output matching method: the output created by a teacher model program was compared to the output of the student program. Today, assessment systems such as ASSYST [12] have ability to evaluate student submissions in five different areas: complexity, correctness, efficiency, style, and test data adequacy. A more sophisticated method for program assessment allows analysis of the internal structure of the student's submissions. The early work in this area was focused on estimating the execution time of each program block [21]. More recent examples include the use of abstract syntax trees in order to determine whether a submission contains the required programming constructs [22]. In addition, there are systems such as Ceilidh [3] (later CourseMaster [10]) that allow several different types of assessment: complexity analysis, dynamic correctness, dynamic efficiency, structural weakness and typographical analysis. The last aspect of the computer program that can be analyzed refers to the style of programming which students use while solving

a given problem. Programming style assessment is not concerned with the functionality of the program solution, but measures whether the student is capable to follow widely accepted coding conventions (e.g. use comments, code indentation, etc.) and write understandable program code. Today, there are several automatic systems that include style assessment feature in software development process [4][15]. We should, by all means, mention Style++ [1], which allows measurement of 64 different styles during the development of C++ programs.

The third category consists of AA supporting *visual answers* in which a student manipulates visualization in order to develop a solution of a given programming task. In addition, visualization can be used for learning basic programming concepts, particularly data structures and algorithms. The main representative of this group of AA is TRAKLA [11] system that, by using various heuristics, compares the model answer to the student's submission and can thus detect some simple errors in the final stage of the data structure. On the other hand, its successor, TRAKLA2 [16] is based on generalized assessment procedure which compares submitted solution of whole simulated algorithm to teacher model solution and tries to find identical states. Among the other systems which can be placed in this group, we should mention Stratum [13] which can help students to understand logic, regular expressions etc., much easier, and Exorciser [24] in which student can solve his exercise or learn basics of theoretical computer science through graphical manipulation of the required entities (e.g. strings).

2.2 Online compilers

The online compilers are usually defined as tools that enable online development of the software products. There are several major advantages of this approach. For example, a student does not need to have a compiler installed on his personal computer and may work on the development from any other Internet and browser enabled device. The first and obvious precondition is of course that the online compiler must support a programming language a student wishes to use in the development or programming process. In addition, another advantage is that this development environment allows students to test their solutions independently to the platform used during the original or offline development.

There are several existing solutions, and among the better-known online compilers we would like to mention JXXX [23] that compiles java source files including applets, DJGPP [7] which supports C programming language, web 2.0 technology based solution called OLC [2] that supports development of the software products in four different programming languages. Of course, these and other tools also have known drawbacks. For example, JXXX compiler could only be used to test already written code, or DJGPP provides simple text editor for writing code that does not support basic coding conventions such as keyword highlighting, text indenting, et cetera, and almost every online compiler does not put emphasis on protection from plagiarism and does not provide the mechanisms by which the teacher could be sure that the student actually wrote submitted solution. Therefore, these solutions are not suitable for teaching programming and are not applicable to be used as a supplement to existing tools in learning process. Mentioned problems motivated us to think of a generative system that will include the advantages of other online compilers, be free of their drawbacks and also provide the possibility of full personalization of programming tasks. We already implemented some of its functionality in form of on-line generator which is described in sections that follow.

3 Architecture of on-line generator

The on-line generator of student's exercises is based on a Generator Scripting Model (GSM), introduced in 2005 by Radošević [18], and further described by Radošević et al. in 2009 [19]. The implemented generation system is shown in Figure 1.

As shown in Figure 1, students enter their registry data (ID number, surname and name) via web interface. The generating system uses the ID number in generation of program specification.

Program specification is in a form of Python list and consists of attribute-value pairs, describing properties of program example (together with student's exercise) to be generated. Our approach was to use entry Python lists with possible properties to be chosen, the process of choosing options based on

student's ID number, and building of specification list. The entry Python lists could look as follows (Figure 2).

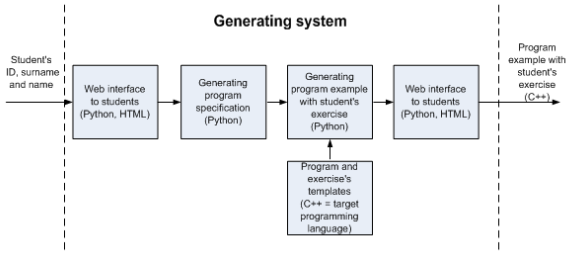

Figure 1: Generation system of student's exercises

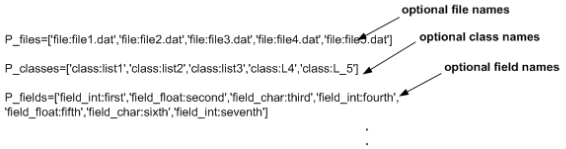

Figure 2: Entry Python lists

Options are chosen using student's ID (by usage of modulo operation), which results in Python specification list (Figure 3).

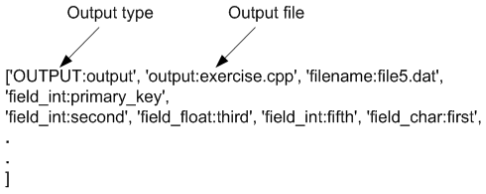

Figure 3: Python specification list

The structure of program specification could be shown by using of Specification diagram (from Generator scripting model, as described by Radošević et al. [19]), Figure 4.

Specification diagram contains all possible attributes to be used in particular student's exercise specification, while specification list defines attribute values. Square brackets define container for lower-level attributes and groups end by '_' sign

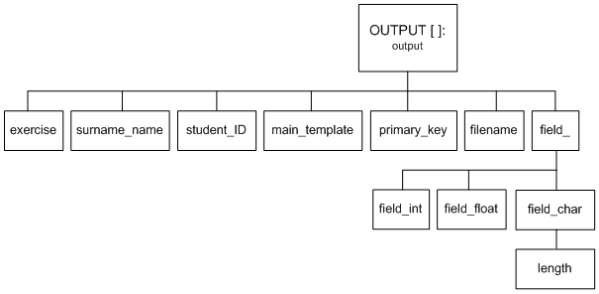

Figure 4: Specification diagram of student's exercise generator

(e.g. field_ means group of attributes that contain field_ in name), as described by Radošević et al. [19]. The attribute OUTPUT is predefined and is used for defining output types (e.g. type of output files to be generated).

4 Example of generation

Except the program specification, the implemented generating system uses sets of program code templates in target programming language (usually C++) which include question for students, in a form of remarks. There are a main program code template (Figure 5.) and lower-level templates.

The code template contains replacing marks in '#' signs (e.g. #field_declarations#) that are replaced by specification values (and/or lower-level templates) during the process of code generation. Student's exercises are incorporated in the same file with the program code template.

The process of generation is defined by generator configuration. The configuration defines what to do with the replacing marks in program code template (Figure 6).

Generator configuration defines file containing main code template (here 'main.template') and the process to be done with the replacing marks:

- direct replacements of replacement mark with the specification attribute value (2 member groups, e.g. #length#, length) or
- replacement with usage of lower-level template (3 member groups, e.g. #forming_record_int#, field_int, field_record_int.template).

```
#include <iostream>
#include <fstream>
#include "library.cc"
struct w#duzina#{
#field_declarations#
};
tindex index;
void F_#length#_1(#formal_arguments#){
 fstream dat,ind;
 w#length# z#length#;
#formal_into_record#
   dat.open ("#file#.dat",ios::out|ios::in|ios::binary);
   ind.open ("#file#.ind",ios::out|ios::in|ios::binary);
   dat.seekp (0, ios::end); ind.seekp (0, ios::end);
   dat.write ((char *)&z#length#,sizeof(w#length#));
   index.primary_key=z#length#.primary_key;
   index.adress=(int)dat.tellp()-sizeof(w#length#);
   ind.write ((char *)&index,sizeof(tindex));
   dat.close();dat.clear();ind.close();ind.clear();
};
.
.
.
int main(){
 fstream dat,ind;
 dat.open ("#file#.dat",ios::out|ios::binary);
 ind.open ("#file#.ind",ios::out|ios::binary);
 dat.close();dat.clear();ind.close();ind.clear();
#field_declarations#
 char forward;
 do{
#field_entry#
   F_#length#_1(#real_arguments#);
   cout << "Forward (y/n)? "; cin >> forward;
 } while (forward=='y');
 F_#length#_2();
 cout << "#surname_name# #student_ID# \nEnd of program." << endl;
}
```

program code template

```
//Exercises:
//Compile and run the example. Enter test data (in file 'Testdata.txt'; 1 point).
//Answer the questins (in Word document):
//1. Which is the size of file "#file#".dat (in bytes, when contains test data).
//   Which is the size of index structure (in bytes)?
//2. Which is the record type of main file, and which of index file?
//3. Write the value of function F_#length#_2 for entered test data?
//Save Word document into file 'exercise_#student_ID#.doc'
//Make the required program modification (1 point):
//Write a function for searcing file according to primary key.
//Function should write the values of all attributes from the record.
//Put the updated program into file 'update_#student_ID#.cpp'
```

exercise template

Figure 5: Example of main program code template

```
#1#,,main.template  <-------  main code template
#exercise#,exercise
#surname_name#,surname_name
#student_ID#,student_ID
#main_template#,main_*,main_*.template
#field_declarations#,field_*,field_*.template
#field#,field_*
#length#,length
#formal_arguments#,formal_arguments(field_*)
#formal_into_record#,field_*,formal_into_record_*.template
#fields_struct#,field_*,field_struct_*.template
#forming_record_int#,field_int,field_record_int.template
#forming_record_float#,field_float,field_record_float.template
#field#,field
#write_record_fields#,field_*,write_record_fields.template
#fields_entry#,field_*,fields_entry.template
#real_arguments#,list(field_*)
#test_data#,field_*,test_data_*.template
#test_data_header#,field_*,test_data_header.template
#field_int#,field_int
#field_float#,field_float
#field_char#,field_char
```

Figure 6: Generator configuration

There are also some special possibilities like specifying atribute groups by asterisk (e.g. #write_record_fields#, field_*, write_record_fields.template) and template names according to attribute names (e.g. #test_data#, field_*, test_data_*.template).

4.1 Generated program with student's example

Generated program consist from several parts:

- identification part (student's ID, surname and name; in a form of comments),
- program code in C++ that includes function which generates file with test data to be entered and
- student exercises (in form of comments).

The whole student session with generation system is shown by Use-case diagram in Figure 7.

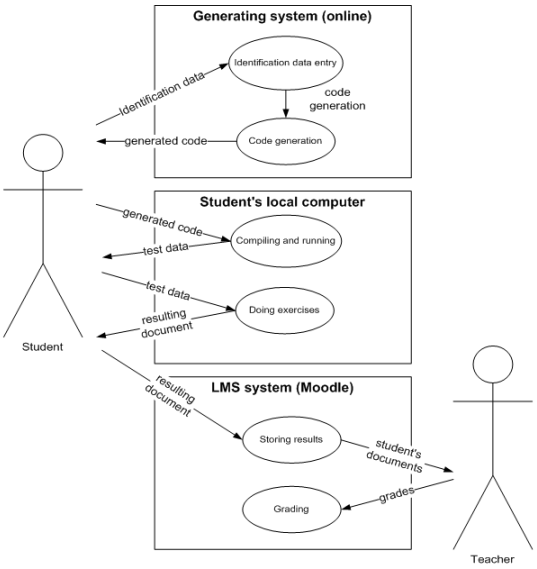

Figure 7: Use-case diagram of the generation system

Student enters his/her identification data (student's ID, surname and name) into the online generation system, which generates program code, together with exercises. Student downloads the code and compiles it on a local computer or by using online compiler. The program generates file with test data. Student does the exercises using test data and makes the resulting document which contains

required answers. In the current solution the document should be manually uploaded on the Learning management system (LMS), and graded by the teacher.

5 Discussion and future work

The offered solution in a form of generating system has the main goal to personalize the student's programming exercises, primarily their homework. That should aggravate copying solutions among students. The generation system is easy to use for the students because they just need to enter their identification data to receive their program codes with exercises. Also, they need to upload their solutions to the LMS system (usually Moodle) which is their usual way to submit their homeworks and other exercises.

Requirements for the teachers are relatively high at the moment, because each exercise requires its particular set of program code templates, which could be partly inherited from previous exercises, and requires a generator configuration which can also partly be inherited from previous exercises. So, some skills of teacher in template meta-programming would be welcome.

Another problem is grading of such personalized programming exercises, which could take some time from the teachers. It could be solved by generating the two polymorphic variants of programming code: one for student and another one for teacher, which includes the print of required solution. That could be automated by on-line compiling of program with the output in form of dynamic generated web page. So, the system which is planned for our future work should introduce that possibility for teachers in order to enable easier grading.

Also, the concept of generating student's exercises could be used in personalization of written exercises, which is also planned to be investigated in our future work.

6 Conclusion

Understanding of programming concepts and algorithms for solving programming tasks are the key points in achievement of programming skills for student. But, students often try to avoid understanding programming concepts by some "shortcuts", like learning program code by rote, copying programs from colleagues etc. Personalization of student's programming exercises should aggravate copying solutions among students who try to avoid understanding of programming concepts. Automatic on-line generating of student's programming exercises offers a solution, where exercises are tied to student's ID-s. The solution is easy to be used by students, but in current implementation phase has some additional requirements for the teachers. At first, the offered approach requires more teachers' work in preparation of exercises and some skills in template meta-programming. Grading of student's exercises could also take some additional time from the teachers, but that could be solved by some improvements of the generating system, which are planned for our future work

References

[1] Ala-Mutka, K., Uimonen, T., Järvinen, H-M.: Supporting students in C++ programming courses with automatic program style assessment, Journal of Information Technology Education, vol. 3, pp. 245-262.

[2] Artal, CG., Suarez, M.D.A., Perez, I.S., Lopez, R.Q.: OLC, On-Line Compiler to Teach Programming Languages, International Journal of Computers, Communications & Control, vol. 3, no. 1, pp. 69-79.

[3] Benford, S.D., Burke, E.K., Foxley, E.: Courseware to support the teaching of programming, Proceedings of the Conference on Developments in the Teaching of Computer Science, University of Kent, 1992, pp. 158-166.

[4] Benford, S.D., Burke, E., Foxley, E., Gutteridge, N., Zin, A.M.: Ceilidh: A course administration and marking system, Proceedings of the 1st International Conference of Computer Based Learning, Vienna, Austria, 1993.

[5] Byrne, P., Lyons, G.: The Effect of Student Attributes on Success in Programming, Proceedings of 6th Conference on Innovation and Technology in Computer Science Education, June 25-27, United Kingdom, 2001, pp. 49-52.

[6] Carter, J., English, J., Ala-Mutka, K., Dick, M., Fone, W., Fuller, U., Sheard, J.: ITICSE working group report: How shall we assess this? SIGCSE Bulletin, vol. 35, no. 4, pp. 107-123.

[7] Delorie, D.: DJGPP Public Access Cross Compiler, available at `http://www.delorie.com/djgpp/compile/`, Accessed: 11th May 2010.

[8] Forsythe, G.E., Wirth, N.: Automatic grading programs, Communications of the ACM, vol. 8, no. 5, pp. 275-278.

[9] Gomes, A., Carmo, L., Bigotte, E., Mendes, A.J.: Mathematics and programming problem solving, Proceedings of the 3rd E-Learning Conference−Computer Science Education (CD-ROM), September 7-8, Coimbra, Portugal, 2006.

[10] Higgins, C., Symeonidis, P., Tsintsifas, A.: The marking system for CourseMaster, Proceedings of the 7th annual conference on Innovation and Technology in Computer Science Education, June 24-28, Aarhus, Denmark, 2002, pp. 46-50.

[11] Hyvönen, J., Malmi, L.: TRAKLA−a system for teaching algorithms using email and a graphical editor, Proceedings of HYPERMEDIA in Vaasa, 1993, pp. 141-147.

[12] Jackson, D., Usher, M.: Grading student programs using ASSYST, Proceedings of 28th ACM SIGCSE Technical Symposium on Computer Science Education, February 27 − March 01, San Jose, California, USA, 1997, pp 335-339.

[13] Janhunen, T., Jussia, T., Järvisalo, M., Oikarinen, E.: Teaching smullyan's analytic tableaux in a scalable learning environment, Proceedings of Kolin Kolistelut / Koli Calling − Fourth Finnish / Baltic Sea Conference on Computer Science Education, October 1-3, Helsinki University of Technology, 2004, pp. 85-94.

[14] Konecki M., Orehovački, T., Lovrenčić, A.: Detecting Computer Code Plagiarism in Higher Education, Proceedings of the 31st International Conference on Information Technology Interfaces, June 22-25, Cavtat, Croatia, 2009, pp. 409-414.

[15] Mäkelä, S., Leppänen, V.: Japroch: A tool for checking programming style, Proceedings of Kolin Kolistelut / Koli Calling − Fourth Finnish / Baltic Sea Conference on Computer Science Education, October 1-3, Helsinki University of Technology, 2004, pp. 151-155.

[16] Malmi, L., Karavirta, V., Korhonen, A., Nikander, J., Seppälä, O., Silvasti, P.: Visual algorithm simulation exercise system with automatic assessment: TRAKLA2, Informatics in Education, vol. 3, no. 2, pp. 267-288.

[17] Naur, P.: Automatic grading of students' ALGOL programming, BIT 4, pp. 177-188.

[18] Radošević, D., Kliček, B.: The Scripting Model of Application Generators, Proceedings of The 16th International DAAAM Symposium Intelligent Manufacturing & Automation: Focus on Young Researchers and Scientists, October 19-22, Opatija, Croatia, 2005.

[19] Radošević, D., Konecki, M., Orehovački, T.: Java Applications Development Based on Component and Metacomponent Approach, Journal of Information and Organizational Sciences, vol. 32, no. 2, pp. 137-147.

[20] Radošević, D., Orehovački, T., Lovrenčić, A.: Verificator: Educational Tool for Learning Programming, Informatics in Education, vol. 8, no. 2, pp. 261-280.

[21] Robinson, S.K., Torsun, I.S.: The automatic measurement of the relative merits of student programs, ACM SIGPLAN Notices, vol. 12, no. 4, pp. 80-93.

[22] Saikkonen, R., Malmi, L., Korhonen, A.: Fully automatic assessment of programming exercises, Proceedings of The 6th Annual SIGCSE / SIGCUE Conference on Innovation and Technology in Computer Science Education, Canterbury, United Kingdom, 2001, pp. 133-136.

[23] Tschalär, R.: JXXX Compiler Service, available at `http://www.innovation.ch/java/java_compile.html`, Accessed: 11th May 2010.

[24] Tscherter, V., Lamprecht, R., Nievergelt, J.: Exorciser: Automatic generation and interactive grading of exercises in the theory of computation, Fourth International Conference on New Educational Environments, May, Lugano, Switzerland, 2002, pp. 47-50.