IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

AN ADAPTIVE SYSTEM FOR AUTOMATED EVALUATION OF STRUCTURED C++ PROGRAM USING INTER - LINGUA AND SPECIFICATION DICTIONARY

[1]Prof. Maxwell Christian, [2]Prof. Bhushan Trivedi, PhD.

[1]Asst. Professor, Faculty of Computer Technology, GLS University, Ahmedabad, India

[2]Dean, Faculty of Computer Technology, GLS University, Ahmedabad, India

Abstract: Automatic assessment of the performed practical program of a student, independent of the approach, has always been a challenging job. Hence continuous progress for improvisations in evaluation techniques has been observed over the time. A lack of automatic evaluation of structural components of the program is also observed. This paper presents the system developed to automate the process of evaluation and grading of a structured C++ program in terms of structural and contextual requirement of required components from the program. The system represents the automated process to evaluate, grade and also suggests grade enhancement measures from the program provided for evaluation.

Index Terms - Automatic evaluation, automatic grading, structural evaluation, self – learning models, grade enhancement, suggestive measures, inter – lingual representation, component – based evaluation, contextual binding.

INTRODUCTION

Automatic evaluation of source code of a C++ program is experienced to be complex because of the variety of implementation approaches towards the solution, a wide choice of the language specific features, the order in which the language feature is used in the program targeted to automatic evaluation and the contextual relevance of components of the program provided for automatic evaluation. So there needs to be a mechanism which can represent an entire C++ program in terms of the context of the component, the alternate and possible language features that can be used for the component and also the applicable grades for each possible and permissible variation of the component in the program to be evaluated.

MOTIVATION

The much time – consuming process of program evaluation, when automated, will enable the senior faculties [and also the students] in receiving the first phase of result in less time. It will also aid junior faculties towards well – structured approach of evaluation. Using an automated evaluation system will further empower examination authorities to have a view of consistent and unambiguous evaluation process and transparent results as compared with current ad – hoc and non - transparent system [1].

CHALLENGES IN AUTOMATED EVALUATION

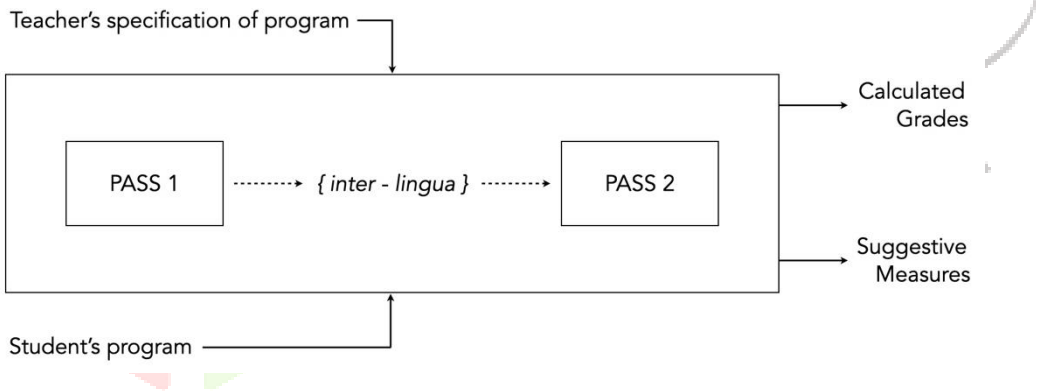
Automated evaluation of a structured C++ program is hurdled by many challenges because of the wide number of language features and the enormous number of possible implementation approach incorporated by students. Below listed are some obvious challenges that occurs in the process of evaluation and are most important to be met:

- Number of components may differ in different programs
- Each program may not consist of the same components in same order
- The contextual use of a component in the program
- A possible variation may be implemented for any component
- The coding style also effects the parsing approach. e.g., Opening and closing braces usage in class/struct/enum declarations
- Special cases like the comment may also contain the keywords of the programming language and needs to be skipped from the parsing process

DEVELOPED SYSTEM

The system presented here provides automated evaluation of a structured C++ program by generating an inter - lingua which represents each component in terms of the context, the language specific feature extracted and the scope in which the language feature has been used. For achieving automatic evaluation, system incorporates a TWO pass mechanism as depicted in figure 1.

Figure 1. System overview



A provided C++ program can have numerous structural components and use of various language features to be evaluated. The provided mechanism successfully achieves parsing and evaluation and grading of the following listed C++ language features:

- Pre – processor directives [symbolic constants and macros]
- Variable declarations
- struct declarations
- struct member variable declarations
- struct member function declarations
- class declarations
- class member variable declarations
- class member function declarations
- enum declarations
- enum member declarations
- global function declarations

– global function parameter declarations

To achieve automatic evaluation, grading and suggestive measure for grade enhancement, the input C++ program is converted to a dictionary format for further processing. The process of conversion of C++ program to dictionary is achieved using following mentioned steps:

1. Identification: Each member of the component is identified and categorised as per the grammar of C++ i.e., identified as symbolic constants or macros in case of pre - processor directives
2. Tokenisation: Each identified member of the component is tokenised to group the parts of the member into appropriate categories i.e., in case of function declaration, in type of the function, return type, contextual name of the functions, the argument list and the trailer of the function (if exists as in case of const functions)
3. Labelling: Each parsed token is labelled [in terms of context] for inclusion in directory in form of key for the directory, for example, each variable declared in the program is coupled with an applicable context as per teacher specification

The developed system achieves the automated assessment of a C++ program through the process depicted in figure 1. As seen in figure 1, the input program for evaluation and the teacher's specification is provided which is used by PASS 1 of the system. The output of PASS 1 is an inter - lingua which is an intermediary representation generated using the specification provided by the instructor/evaluator. The format of the inter - lingua is as mentioned in figure 2.

Figure 2. Inter - lingua format

<scope> <language feature type> <contextual name> [<language specific feature used 1> <language specific feature used 1> ..... <language specific feature used N>]

The inter - lingua is generated for each executable statement of the input program and a new program comprising the formulated inter - lingua as a comment for the targeted statement of the input program is stored for further usage by PASS 2 from figure 1.

The PASS 2 of the developed system comprises of following mentioned steps:

1. **Segregation**: Each component of the targeted program for evaluation is segregated on the basis of the language feature i.e symbolic constant, macro, class, structure, variable declaration, function declaration, etc.
2. **Storage**: Each segregated component is stored into the scalable data structure of dictionary
3. **Comparison**: Each component from the program targeted for evaluation is compared with the matching component from the teacher specification
4. **Grading**: Each matched component from the program to be evaluated and the teacher specification, is then graded as per the applicable grade as specified in the teacher specifications.
5. **Suggestions**: All such components where the maximum grade specified by the instructor is not achieved, the possible measure to enhance the grading [using the component – based specification] is suggested
6. **Adaptation**: A final step where in case the instructor/evaluator modifies/improvises the grades [for any component] calculated by the automated process [finding that the version implemented in the targeted program is quite novel/unique], the implemented version of the component from the targeted program will be added to the teacher specification for future usage.

ADVANTAGES

The system developed for automated evaluation has clear benefits of unambiguous, consistent and faster evaluation over the manual process for several variations of same program as well as for different programs [1]. Also, the adaptive feature of the developed system makes it growing in terms of its knowledge base of evaluation parameters and grading scheme. The developed system also demonstrates the helpful feature of grade enhancement suggestions thus providing valuable feedback to the students for improvising his version of the program as well as understanding the differences between his implemented version as well as possible variants of the same program. Also, the system as tokenises the entire program, these parsed tokens of an entire C++ program provide the clear idea about the components used in the program as well as the way they are used. Also, the alternate applicable values for the components can be identified, for example, data type options in case of variable declaration. Parsing and generating a dictionary provides the benefit of scalability in terms of the number of components of the C++ program. The dictionary also has the benefit to incorporate the independent order of the components present from the program targeted for automatic evaluation. Incorporating dictionary also adds scalability, extendibility, serialisability and inter-operability features to the automated evaluation system.

CONCLUSION

The outcome is a self – learning system which is capable of evaluating a structured C++ program(s) performed by students, against the specification set provided by evaluator, providing grades for the same program and also suggest grade enhancement measures to all applicable structural components of the program. The developed system also adapts to unseen implementation approaches and incorporates the same in the instructor specification set for future usage. The generated inter - lingua along with the scalable and portable dictionary provides extensibility features for the same system to be incorporated for other programming languages also. The presented system addresses the much – desired issues of automated evaluation like contextual understanding by the automated system, independence in terms of order of the components in the program and a common representation of the program in terms of the definition of the program.

REFERENCES

1. Doshi, J.C.; Christian, M.; Trivedi, B.H. (2014), Effect of Conceptual Cue Based (CCB) Practical Exam Evaluation of Learning and Evaluation Approaches: A Case for Use in Process-Based Pedagogy, Technology for Education (T4E), 2014 IEEE Sixth International Conference on Technology for Education, pp 90 - 94
2. Forsythe, G. E., Wirth, N. (1965). Automatic Grading Programs. Commun ACM, vol. 8, pp. 275- 278.
3. Higgins, C. A., Gray, G., Symeonidis, P., Tsintsifas, A. (2005). Automated Assessment and Experiences of Teaching Programming. Journal on Educational Resources in Computing (JERIC), vol. 5, pp. 5.
4. Douce, C., Livingstone, D., Orwell, J. (2005). Automatic Test-based Assessment of Programming: A Review. Journal on Educational Resources in Computing (JERIC), vol. 5, pp. 4.
5. Ihantola, P., Ahoniemi, T.,Karavirta, V., Seppälä, O. (2010). Review of Recent Systems for Automatic Asessment of Programming Assignments. Proceedings of the 10th Koli Calling International Conference on Computing Education Research, pp. 86-93.
6. Romli, R., Sulaiman, S., Zamli, K. Z. (2010). Automatic Programming Assessment and Test Data Generation a Review on its Approaches. Information Technology (ITSim), 2010 International Symposium, pp. 1186-1192.
7. Caiza, J. C.; Del Alamo, J. M. Programming Assignments Automatic Grading: Review of Tools and Implementations. Inted 2013 Proceedings, 2013, 5691-5700
8. Rodríguez-del-Pino, J. C., Rubio-Royo, E., Hernández-Figueroa, Z. J. (2012). A Virtual Programming Lab for Moodle with Automatic Assessment and Anti-plagiarism Features.

9. Queirós, R. A. P., Leal, J. P. (2012). PETCHA: A Programming Exercises Teaching Assistant. Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education, pp. 192-197.
10. Spacco, J., Hovemeyer, D., Pugh, W., Emad, F., Hollingsworth, J. K., Padua-Perez, N. (2006). Experiences with Marmoset: Designing and Using an Advanced Submission and Testing System for Programming Courses. ACM SIGCSE Bulletin, vol. 38, pp. 13-17
11. Jelemenská, K. Čičák, (2012). Improved Assignments Management in MOODLE Environment. INTED2012 Proceedings, pp. 1809-1817.
12. Jackson, D., & Usher, M. (1997). Grading student programs using ASSYST. *Proceedings of the Twenty- Eighth SIGCSE Technical Symposium on Computer Science Education*, USA, 335-339.
13. Schorsch, T. (1995). CAP: An automatic self-assessment tool to check Pascal programs for syntax, logic and style errors. *Proceedings of the 26th SIGCSE technical symposium on Computer science education*, USA, 168-172.
14. Emma Enstro¨m, Gunnar Kreitz, Fredrik Niemela, Pehr Soderman, and Viggo Kann. 2011. Five Years with Kattis – Using an Automated Assess- ment System in Teaching. In Frontiers in Education Conference (FIE), 2011. Institute of Electrical and Electronics Engineers, Piscataway, NJ, T3J–1.
15. Mike Joy, Nathan Griffiths, and Russell Boyatt. 2005. The BOSS on- line submission and assessment system. ACM Journal on Educational Resources in Computing 5, 3, Article 2 (Sept. 2005), 28 pages. DOI:http://dx.doi.org/10.1145/1163405.1163407
16. Lauri Malmi, Ari Korhonen, and Riku Saikkonen. 2002. Experiences in automatic assessment on mass courses and issues for designing virtual courses. ACM SIGCSE Bulletin 34, 3 (2002), 55–59.
17. Jacques Philippe Sauve´, Oso´rio Lopes Abath Neto, and Walfredo Cirne. 2006. EasyAccept: a tool to easily create, run and drive development with automated acceptance tests. In Proceedings of the 2006 international workshop on Automation of software test (AST '06). ACM, New York, NY, USA, 111–117. DOI:http://dx.doi.org/10.1145/1138929.1138951
18. Brad Vander Zanden, David Anderson, Curtis Taylor, Will Davis, and Michael W. Berry. 2012. CodeAssessor: An Interactive, Web-Based Tool for Introductory Programming. The Journal of Computing Sciences in Colleges 28, 2 (2012), 73 – 80.
19. Yuen Tak. Yu, Chung Keung Poon, and Marian Choy. 2006. Experiences with PASS: Developing and Using a Programming Assignment Assess- ment System. In Quality Software, 2006. QSIC 2006. Sixth International Conference on. Institute of Electrical and Electronics Engineers, Piscat- away, NJ, 360–368. DOI:http://dx.doi.org/10.1109/QSIC.2006.28
20. A.T. Chamillard and J.K. Joiner, "Using lab practica to evaluate programming ability". In Proceedings of the 32nd ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2001), ACM Press, pages 159-163, 2001.
21. Andrew Sutton and Bjarne Stroustrup, "Design of Concept Libraries for C++", Texas A&M University Department of Computer Science and Engineering {asutton, bs}@cse.tamu.edu