Journal Homepage: -www.journalijar.com

INTERNATIONAL JOURNAL OF ADVANCED RESEARCH (IJAR)

Article DOI:10.21474/IJAR01/12823
DOI URL: http://dx.doi.org/10.21474/IJAR01/12823

ISSN NO. 2320-5407

RESEARCH ARTICLE

SEMANTIC AND CONTEXTUAL KNOWLEDGE REPRESENTATION USING INTER - LINGUA FOR AUTOMATED ASSESSMENT OF STRUCTURED C++ PROGRAM

Prof. Maxwell Christian and Prof. Bhushan Trivedi, PhD
GLS University.

Manuscript Info

Manuscript History
Received: 05 March 2021
Final Accepted: 09 April 2021
Published: May 2021

Key words:
Automatic Evaluation, Automatic Grading, Structural Evaluation, Inter - Lingual Representation, Component-Based Evaluation, Contextual Binding

Abstract

The process of automatic evaluation of a structured C++ program has the first and foremost requirement of specifying the developed structural C++ program in a standard, generalized and unified format. This generalized and standard representation of the program needs to incorporate the specification in terms of semantics as per the language features used and also the contextual binding in regards of the program definition as perceived and desired by the subject expert. Hence an inter - lingua which can represent the semantic and contextual knowledge of the developed structured C++ program will be the first and foremost requirement towards the process of automated assessment and grading of a structured C++ program.*The work presented here is published for patent at Patent Office Branch, Mumbai, India with the reference number E-12/215/2021/MUM and application number 202121000796.*

Introduction:-

Evaluating a C++ program automatically is complex due to affecting parameters of approach towards the solution, rich language features for use, implementation order of components of the given program and contextual relevance. Hence a generalized intermediate specification of the source code of the program is highly required which can represent the program irrespective of the mentioned affecting parameters.

Motivation

Each C++ program can be implemented in various ways as per the requirement of the components in that program. The order in which the required components exist in the program can differ in the possible versions of the program. Also, the use of the applicable language feature and the choice of the perfectly applicable language feature from the possible options also affects the evaluation and achievable grades. Further the component should be identified in terms of the context of use. Assessing all the components of a C++ program on all such parameters is easily done by expert evaluators due to their experience. Such rich knowledge of experts, regarding program understanding in terms of evaluation, if represented in a generalized way can work as a stepping stone towards automated evaluation.

Challenges In Inter – Lingua

Formulating a generalized inter - lingua for a structured C++ program is hurdled by quite a number of parameters as mentioned in the below list:

Corresponding Author:- Maxwell Christian
Address:- GLS University.

1. Total number of components in each program can vary as per different implementations from different student i.e., a student can accomplish working with name of an entity using single variable while other student can achieve the same using different variables for parts of the name like first name, middle name and last name
2. The order of implementation of the components can vary i.e., some students may code member variables followed by methods using these member variables. Other students may have different sectioning of variables and then coded methods.
3. Each implemented component has a possible of variation affected by the usage of language specific feature. For example, some students may achieve fetching count using normal member functions while other students can approach the same using inline/static functions
4. The coding standard in the structured component affects the parsing process and hence the process of generalization of specification of inter - lingua. i.e., some student may like declaring all variables of same data type on a single line while other students may declare each variable on a different line
5. Identifying the portion of the program required to skip from the inter - lingua. e.g., comments from the program
6. The contextual use of a component in the program is the most crucial challenge to formulate the inter - lingua as it depends on the way the developer perceives the program definition. i.e., a student may name the function returning count as count itself while other students may label the same function with the name of total, still both represent the same context
7. Number of components in a particular C++ program can vary and hence scalability is desired i.e., a student may achieve initialization of member variables using a single constructor while another student may have achieved the same using different versions of constructors

Developed Inter – lingua

The inter - lingua formulated and presented here represents the structural components of a student's version of C++ program as per the teacher specification. The inter - lingua successfully incorporates the specification of components mentioned in the below list:

1. Scope: representing the scope of each variable and function coded in the program. i.e., global or member scope with possibility of nested scope
2. Specific language feature: use of C++ language specific feature like static functions, inline functions, const functions, declaration of member variables outside classes, friend functions, etc. There also exists the possibility of variation depending on the version of language used
3. Context binding: this refers to the use of a program component as per the context of the definition. i.e., a student may refer the total number of students using the variable count while another student may refer the same using a variable name total.
4. Features specific to main type used: this refers to detailed specification of the language feature used for proper applicability. For example, a student may code age of an entity using int data type while another student may code the same using unsigned int

The format of the inter - lingua developed and formulated in a generalized manner is depicted in figure 1.

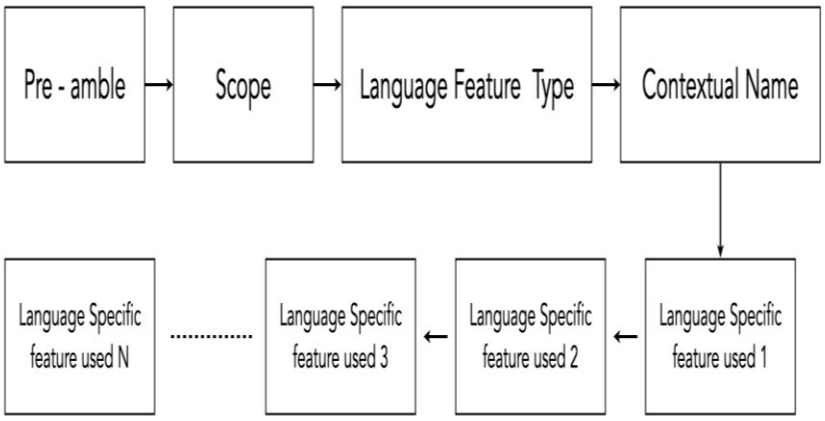

Figure 1:- Inter - lingua components.

The inter - lingua specified in figure 1 has been successfully tested for various features of C++ like symbolic constants and macros, global variables, scope-based variables, variables as struct and class members, variables as enumeration members, global function signatures and scope-based function signatures like functions as members of classes. Specifically in case of functions, the inter - lingua will also be incorporating the specification of function type i.e., static or friend, return type, each parameter specification and the special function type if applicable e.g., constant function as it occurs at the end of a function signature. The inter - lingua is generated and stored as comment alongside each executable statement of the component to be evaluated from the source code of the desired C++ program. Hence the pre - amble is used to differentiate the inter - lingua from other source code comments.

Advantages

The developed and formulated inter - lingua works as a core mechanism for the automated process of assessment of structured C++ programs. Only due to the presence of inter - lingua, any implemented version of the structured C++ program can be specified in a generalized and standard way as per the specification provided by the subject expert or the teacher. The inter - lingua actually mimics the way the subject expert uses his knowledge to understand a structured program. Thus, the inter - lingua is the knowledge representation in terms of the program definition, the structural requirement and the applicable language features i.e., C++ language features. This generalized Inter - lingua further is also a standard and common representation of knowledge irrespective of the vast possible implementation variations of the same structured C++ program definition. And storing inter - lingua is stored as comments alongside the source code of the program provides clear indication to the automated assessment process in selection the statements of the program that are required to be evaluated.

Conclusion:-

The developed and formalized inter - lingua successfully achieves the solution helpful for addressing the challenges faced in automated assessment of structured C++ program. The formalized inter - lingua is capable of capturing the scope, the specific component implemented, the contextual binding of the implemented component and also the language specific feature implemented for all the developed components of the desired program definition. Also, the inter - lingua is scalable as it is generated for all structural components from the developed version of the program as per the specification from the subject expert or teacher. Also, the generalized format of the inter - lingua can be extended for program languages other than C++ also.

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, PehrSoderman, 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 RikuSaikkonen. 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 WalfredoCirne. 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.