J. Appl. Environ. Biol. Sci., 4(7S)282-286, 2014

ISSN: 2090-4274
Journal of Applied Environmental and Biological Sciences
www.textroad.com

Pedagogically Effective Subset of C++

Muhammad Shoaib Farooq[1,2], Sher Afzal Khan[2], Farooq Ahmed[3], Saeed Islam[2], *Adnan Abid[1]

[1]Department of Computer Science, University of Management and Technology, Lahore, Pakistan.
[2]Abdul Wali Khan University, Mardan, Pakistan.
[3]Faculty of Information Technology, University of Central Punjab Lahore, Pakistan.

Received: September 1, 2014
Accepted: November 13, 2014

ABSTRACT

Programming languages normally grow in size due to feature multiplicity and backward compatibility. This is the main reason behind the usual practice of teaching a subset of an easy and useful language to the students in an introductory course in computer programming. In this research, we propose a subset of C++ which is based on a conceptual framework to evaluate a First Programming Language (FPL), proposed in our earlier work. We believe that the proposed subset results into a pedagogically more effective C++, and can help improving the teaching and learning experience for a first course in computer programming.
KEYWORDS: First Programming Language, Language Subsetting, Computer Programming.

1 INTRODUCTION

The Programming languages continuously evolve and the size of a language increases by adding new features, and by ensuring the backward compatibility of constructs. This results into feature multiplicity problem[2], and due to this problem the languages offer a longer learning curve to the students, while at the same time, the instructor is unable to teach whole language. Consequently, the instructor teaches the subset of language in tight schedule of one semester course. These subsets may be used to code every type of problems, but in order to understand the program written by others, a student should learn whole language [1][4][6]. There exist no special guidelines which help the course instructors in creating such proper subsets of a programming language. Alternatively, there is another approach usually termed as pseudo language approach defined by educators in CS Community. A pseudo language is typically a subsets of an existing mainstream programming language with some extra features, in order to teach the basic programming concepts[8][7][9]. The idea of a pseudo language is to create code with as simple syntax as possible. So a student can pay more attention to learning programming concepts, and may focus more on problem solving skills instead of learning typical syntax. This approach has not been so popular, mainly for the reason that it involves some extra features which do not belong to core language. Furthermore, due to these new features it requires a new compiler implementation.

2 RELATED WORK

Defining an effective subset of programming languages for pedagogical and safety purpose has been a common practice for decades. Common Business Oriented Language (COBOL) is a first language that has been subsetted due to large number of redundant constructs and its complex syntax for novices[11][12]. Ada is derived from pascal by applying subsetting rules and then SmallAda has be derived for novices from whole set of constructs from Ada programming language[14][15]. Mini Java a subset of java was defined by Eric Roberts from Stanford University for purpose of effectively teaching and learning[16]. It's a common practice by choosing best available constructs and discarding redundant, unsafe, and semantically ambiguous constructs from a given language to make its pedagogically effective[11][13]0. Another methods is to choose subset of language by reflection and overloading[10].In the same way, recently some research

*** Corresponding Author:** Adnan Abid, Department of Computer Science, University of Management and Technology, Lahore, Pakistan.

0[2] has been conducted on the evaluation and specification of introductory programming languages which can be very useful in defining these subsets.

3 Usage of framework to make a language a better FPL

Farooq at el. [1] proposed a comprehensive framework for the assessment of first programing language. This framework consist of technical and environmental features, using these features we can evaluate conformance of programming language toward a healthy first programming language. In order to consolidate an existing language, the proposed framework approach ensures that a language should not be modified so as to add new features in it, which are demanded by the framework, but are not a part of the language. Therefore, no new features should be added to a language, so as to increase its conformance to the proposed framework. As an example, if a language does not support generic programming, then it should not be modified in such a way that new constructs are added to it, so that it starts supporting generic programming. Certainly, such a change in the language is a considered a major change in existing language and, in general, these types of changes are introduced to the languages in newer versions. So, such improvements in a language are the responsibility of the language designers.

Secondly, improve the language in the following two possible ways: (i) if possible, apply constraints on the usage of existing constructs so as to improve their compliance to the requirements of the proposed framework; (ii) eliminate the constructs which are problematic, and with their elimination the capabilities of a language are not affected. Generally, such constructs are redundant, and cause the problems like safety, readability, reliability, maintainability etc.

Now, the elimination of constructs means that such constructs are no more available to the programmer, and this elimination can be enforced with the help of a stricter pre-processor. Similarly, a smarter pre-processors along with more sophisticated IDEs should be used to apply the constraints on the usage of the language constructs so as to align their usage according to the considerations put forth by the framework [1]. The framework is composed of two main feature sets, namely, technical and environmental features.

Above all, the proposed method for the improvement of an existing language does not add any new features to it, and restricts the usage of conflicting, unsafe, and redundant constructs so as to increase the suitability score of a language based on the proposed framework. It is pertinent to mention here, that the improved version of the language would produce valid code, which can be run by using any compiler of the language. The reason is fairly simple, i.e. the improved language will have fewer constructs than the existing language by applying subsetting and thus, removing the unwanted constructs. Secondly, by applying restrictions on the usage of constructs again allows subsets of the ways a construct can be used in the coding of a language. Therefore, all programs written in the improved language must be valid programs according to the original language as well.

3 Constructs Selection/Rejection of C++ based on Conceptual Framework.

This section presents the discussion on the improvement of C++, which is a widely used FPL, so as to increase its conformance to the defined framework. Firstly, this section considers subsetting of C++, where the problematic constructs are eliminated from the targeted newer C++. To this end, we focus on the C++ constructs which are used in an introductory course in computer programming, which include, data type, modifiers, life time, operators, conditional structures, loops, arrays, and functions. As a result in Table 1, all constructs related to the afore mentioned topics of the language have been presented under specific construct types, and each for each rejected construct the conflicting feature/sub-features have been mentioned. Resultantly, this section provides us with a cleaner subset of C++ which eliminates some language constructs, and hence, increases its conformance to the proposed framework.

Table 1. Subset of C++

C++ Language Construct			
Type	Subtype	Selected	Comment
Data Type	int	Yes	
	Long	No	Learning Overhead for long literal e.g. 24*l* Feature multiplicity
	Float	No	Learning Overhead for float literal e.g. 2.25f Feature multiplicity
	double	Yes	
	Char	Yes	
	Bool	Yes	
	Void	Yes	
	string	Yes	
	wchar_t	No	Feature multiplicity
Modifiers	signed	No	Feature multiplicity
	unsigned	No	Feature multiplicity
	short	No	Feature multiplicity
	Long	No	Feature multiplicity/Learning Overhead required
	register	No	Feature multiplicity/Learning Overhead required
	const	Yes	
Life Time	auto (Stack Dynamic)	No	Orthogonality/Consistent Rule
	static local variable	No	Orthogonality/Consistent Rule
Operators	Arithmetic Binary Operators	Yes	
	Arithmetic unary Operators (-,+)	No	Enforceability of Good Habits
	Arithmetic pre and post increment operators (++,--)	No	Enforceability of Good Habits/No Side Effects
	Relational Operators	Yes	
	Logical Operators	Yes	
	Assignment Operator (=)	Yes	
	Compound Assignment Operators (+=,-=,*=,/=,%=)	No	Enforceability of Good Habits/No Side Effects
	Bitwise Operators (<<,>>,~,^,|,&)	No	Learning Overhead
	Comma Operator (,)	No	Quality Coding Standard/MISRA[5]
	sizeof	No	Enforceability of Good Habits/Side Effects
Language Constructs (condition)	If-else	Yes	
	switch	No	Feature Uniformity/Feature Multiplicity
	Ternary Operators (: ?)	No	Feature Uniformity/Feature Multiplicity Enforceability of Good Habits
Language Constructs (Loops)	For loop	No	Feature Uniformity/Feature Multiplicity
	While loop	Yes	
	Do-while loop	No	Feature Uniformity/Feature Multiplicity
Language Constructs (Control jump)	break	No	Feature Uniformity/Feature Exclusiveness Quality Coding Standard
	continue	No	Feature Uniformity/Feature Exclusiveness

			Quality Coding Standard
	goto	No	Feature Uniformity/Feature Exclusiveness Quality Coding Standard
	return	Yes	
Arrays	C-Style Array	No	Security/Control over Array Index out of Bounds
	Vector	Yes	
	String	Yes	
Type Conversion	C Style Explicit Type Casting	No	Security Strongly Typed
	Static_cast	Yes	
	cont_cast	Yes	
	dynamic_cast	Yes	
	reinterpt_cast	No	Strongly Typed
Functions	Parameter passing by reference using pointer	No	Security
	Parameter passing by reference using reference variables	Yes	
	Default value of function arguments	Yes	
	Command Line Arguments	Yes	
	C Style unspecified number of arguments	No	Readable Syntax
	Function Overloading	Yes	
	Passing Array to functions using Pointers	No	Security/Array index out of bounds
Comments	Mega Comment #if …….#endif	No	Quality Coding Standard/Comments
	Block Comment /*……..*/	No	Quality Coding Standard/Comments
	End of Line Comment //	Yes	

4 Conclusion

In this work we propose a proper subset of C++ programming language to make it pedagogically effective, easy to learn language. The language improvement process mainly involves i) preprocessing, and ii) subsetting. We propose preprocessing, where certain types of restriction can be applied through rewriting language lexical, syntax and semantic preprocessor. Whereas, we propose the usage of subsetting where we can easily eliminate redundant feature from language due to feature multiplicity. These new constraints employed in the preprocessor shall improve its ability to perform lexical, syntax, and semantic analyses. Therefore, the conformance of C++ to the proposed framework can be improved in many ways using a more sophisticated preprocessor. After that we use these guidelines on C++ for creating proper subset. Every program written using this subset is a valid program of C++.

REFERENCES

[1] Farooq, Muhammad Shoaib, Sher Afzal Khan, Farooq Ahmad, Saeed Islam, and Adnan Abid. "An Evaluation Framework and Comparative Analysis of the Widely Used First Programming Languages." PloS one 2014; 9(2) : e88941.

[2] Muhammad Shoaib Farooq, Sher Afzal Khan, Adnan Abid A Framework for the Assessment of a First Programming Language, Journal of Basic and Applied Scientific Research, 2012; 2(8): 8144-8149.

[3] Muhammad Shoaib Farooq, Adnan Abid, Sher Afzal Khan, Muhammad AzharNaeem, Amjad Farooq, Kamran Abid, A Qualitative Framework for Introducing Programming Language at High School, Journal of Quality and Technology Management, Punjab University, Pakistan. 2012; 8(2).

[4] DePasquale, P., Lee, J. A., & Pérez-Quiñones, M. A. “Evaluation of subsetting programming

language elements in a novice's programming environment". ACM SIGCSE Bulletin, 2004; 36(1), 260-264.
[5] Hatton, L. Safer language subsets: an overview and a case history, MISRA C. Information and Software Technology, 2004; 46(7), 465-472.
[6] DePasquale, P. "Subsetting language elements in novice programming environments". In Proceedings of the RESOLVE Workshop 2002; pp. 108-111.
[7] Lusth, J. C., Kraft, N. A., &Tacey, J. "Language subsetting via reflection and overloading". In Frontiers in Education Conference, 2009. FIE'09. 39th IEEE 2009, October; pp. 1-6. IEEE.
[8] Harvey, B., &Mönig, J. Bringing "No ceiling" to scratch: can one language serve kids and computer scientists. Proc. Constructionism, 2010.
[9] Federici, S."A minimal, extensible, drag-and-drop implementation of the C programming language". In Proceedings of the 2011 conference on Information technology education, 2011 october; pp. 191-196.ACM.
[10] Lusth, John C., Nicholas A. Kraft, and James Tacey. "Language subsetting via reflection and overloading." Frontiers in Education Conference, 2009. FIE'09. 39th IEEE.
[11] Ben-Ari, Mordechai, KevlinHenney. "A critique of the advanced placement C++ subset." ACM SIGCSE Bulletin,1997, 29(2): 7-10.
[12] Sammet, J. Programming Languages: History and Fundamentals, 1st ed. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1969.
[13] Sebesta, R. W. Concepts of Programming Languages, 4th ed. Addison-Wesley, 1999, ch. 1, p. 9.
[14] Webber, A. B. The Pascal Trainer. In Proceedings of the Twenty Seventh SIGCSE Technical Symposium on Computer Science Education (New York, New York, March 1996, Association for Computing Machinery, pp. 261–265.
[15] Feldman, M. B., Lopes, A. V., and P´erez-Qui˜nones, M. SmallAda: Personal Computer Courseware for Studying Concurrent Programming. In Proceedings of the Twenty-First SIGCSE Technical Symposium on Computer Science Education, 1990, ACM Press, pp. 206–211.
[16] Roberts, E. (2001) An Overview of MiniJava. In Proceedings of the Twenty Seventh SIGCSE Technical Symposium on Computer Science Education, ACM Press,2001, 1–5.