AUTOMATIC CHARACTERIZATION OF COMPUTER PROGRAMMING ASSIGNMENTS FOR STYLE AND DOCUMENTATION

Dulal C. Kar*

INTRODUCTION

Program readability is important for future maintenance of a program – a common practice in industry. Readability and understandability of a program can be enhanced by following some consistent program developing style and including enough and appropriate documentary comments at various places in the program (Dale, Weems & Headington, 2000; Coplien, 1991). In class programming assignments, students are expected to follow some guidelines consistently for style and documentation. Grading programming assignments for style and documentation for large classes is a very time consuming process as the process often involves meticulous checking as well as writing some feedback with suggestive comments for future improvement. Due to resource constraints, sometimes only one or two assignments from a set of assignments are randomly chosen for manual checking of style and documentation. Consequently, students hardly get any chance for improvement in style and documentation in future assignments.

This work presents an automatic characterization system that can check a student's program for style and documentation based on some guidelines and provide immediate feedback to the student if certain elements or components are missing or lacking in the program. An instructor can use such system to find out the students who are not following consistently the guidelines for style and documentation. Such system can also be used to categorically grade students' works on style and documentation.

CHECKING DOCUMENTATION

Checking program documentation requires understanding of the program as a whole and its individual components. Consequently, automatic checking whether some documentary comment is relevant and meaningful is not possible, at least with current technology and knowledge. However, it is relatively easy to check the presence, absence, or amount of documentary comments in a program.

In a C++ program, documentary comments can occur in three forms: *block comments*, *single-line comments* and *trailing comments*. Block comments containing multiple lines of comments are used at the beginning of a program, a module, or a function.

---

* Department of Computing and Mathematical Sciences
Texas A&M University – Corpus Christi, Corpus Christi, TX 78412

Single-line comments or even block comments are used at the beginning of a logical block of code that usually start with *for*, *while*, *do-while*, *switch*, or *if* statement. Single-line or block comments are also used at the beginning to describe a data structure or a class abstraction. Trailing comments are used to describe variable, constant, or function parameter declarations. Trailing and line comments are also used for program statements. We assume that in an introductory programming class, programs written by students are small and simple enough that students put documentary comments only at designated places as set by the course instructor. Accordingly, the automatic checking process can be carried out as follows:

1. Find the beginning block comment of the program, count lines and words in the comment block.
2. Locate each function or method definition, class declaration, structure declaration, and so on in the program, check for existence of any documentary comment block as heading, and count lines and words in the heading accordingly.
3. Find the location of each logical block of code beginning with *for*, *while*, *if*, *do-while* or *switch* statement, check for the beginning block or line comments for the code segment, and count lines and words in it.
4. Find the first occurrence of a variable name, a function parameter, or a constant, check for any trailing comment on it, and count words in the trailing comment.
5. Check trailing comments for statements in a function, module, or program and count words in each comment.
6. For each of the above steps, report any missing documentation with appropriate position information such as line number, function or module name, etc.
7. Calculate statistics on various counts and report them.

CHECKING PROGRAMMING STYLE

Program readability is essential to understanding of a program. Adopting a good programming style for the development of an entire program is important for its readability. The following style rules can be considered for characterizing and checking a student's program written in C++ (Henricson & Nyquist, 1992):

1. Use meaningful names for all identifiers. Checking of such rule is not possible by an automated system. However, the identifier naming convention, structure, and size can be checked easily.
2. Do not exceed 80 characters for a statement on a line.
3. Limit all functions, methods, classes, structures, etc. within 40 lines.
4. Write only one statement per line.
5. Indent all statements in a logical structure such as inside a *for* loop, a *while* loop, an *if* block, a *switch* block, a class or structure declaration, a function or method definition, and so on.
6. Use whitespace before and after a binary operator.

7. Leave at least one blank line before the beginning of a logical structure such a function, a class or structure declaration, a *for* loop, and so on.
8. Limit number of parameters in a function to 7.
9. Indent continuation of a statement.
10. End a logical block with a closing brace on a separate line with the same spacing as the first line of the block.

Similar to checking program documentation, parsing and counting techniques can be used to check a program for compliance of all the above rules for programming style.

IMPLEMENTATION

A prototype system has been developed to automate the process of characterizing students' programs written in C++. A student can submit a program to the system from any host on Internet through a client program. After checking the authenticity and enrollment of a student in a class, it performs the analysis of the program for style and documentation. It produces a report, similar to an error report generated by a compiler, with line numbers of all lines in the program that do not follow the rules for style and documentation. It also produce a report on statistics of comment words, comment lines, comment blocks, programming statements, programming blocks, etc. It then returns the complete report on the characterization of the program to the student via e-mail.

CONCLUSION

An automatic characterization system can provide immediate feedback on some student's program for style and documentation. Students can use such feedback for improvement or an instructor can use such feedback grading or warning a student. A prototype system for the purpose has been developed and found effective in characterizing students' programs.

BIBLIOGRAPHY

Dale, N., Weems, C., & Headington, M. (2000). Programming and Problem Solving with C++, Boston: Jones and Bartlett Publishers.

Coplien, J. O. (1991). Advanced C++ Programming Styles and Idioms, Reading, Mass.: Addison-Wesley Publishing Company.

Henricson, M. & Nyquist, E. (1992). URL: http://www.cs.umd.edu/users/cml/cstyle/Ellemtel-rules.html