A Program Transformation Tool for Visualizing Control Structure in Graphical Representations

Chih-Yueh CHOU[ac*], **Pin-Cheng, LIAO**[a], **K. Robert LAI**[ac], **& Zhi-Hong CHEN**[bc]
[a]*Department of Computer Engineering and Science, Yuan Ze University, Taiwan*
[b]*Department of Information Communication, Yuan Ze University, Taiwan*
[c]*Innovation Center for Big Data and Digital Convergence, Yuan Ze University, Taiwan*
*cychou@saturn.yzu.edu.tw

Abstract: In this paper, we describe a work-in-progress research to develop a program control structure visualization tool for transforming student programs into graphical representations of control structure, including sequences, selections, and iterations. The tool can be used for assisting novice students in debugging their programs by checking whether the transformed control structure of their programs is consistent with their program plans. In addition, the graphical representation of student programs can be compared to that of a model program to detect errors in program plans.

Keywords: Computer assisted programming learning, Program transformation, Program visualization, Control structure, Debugging.

1. Introduction

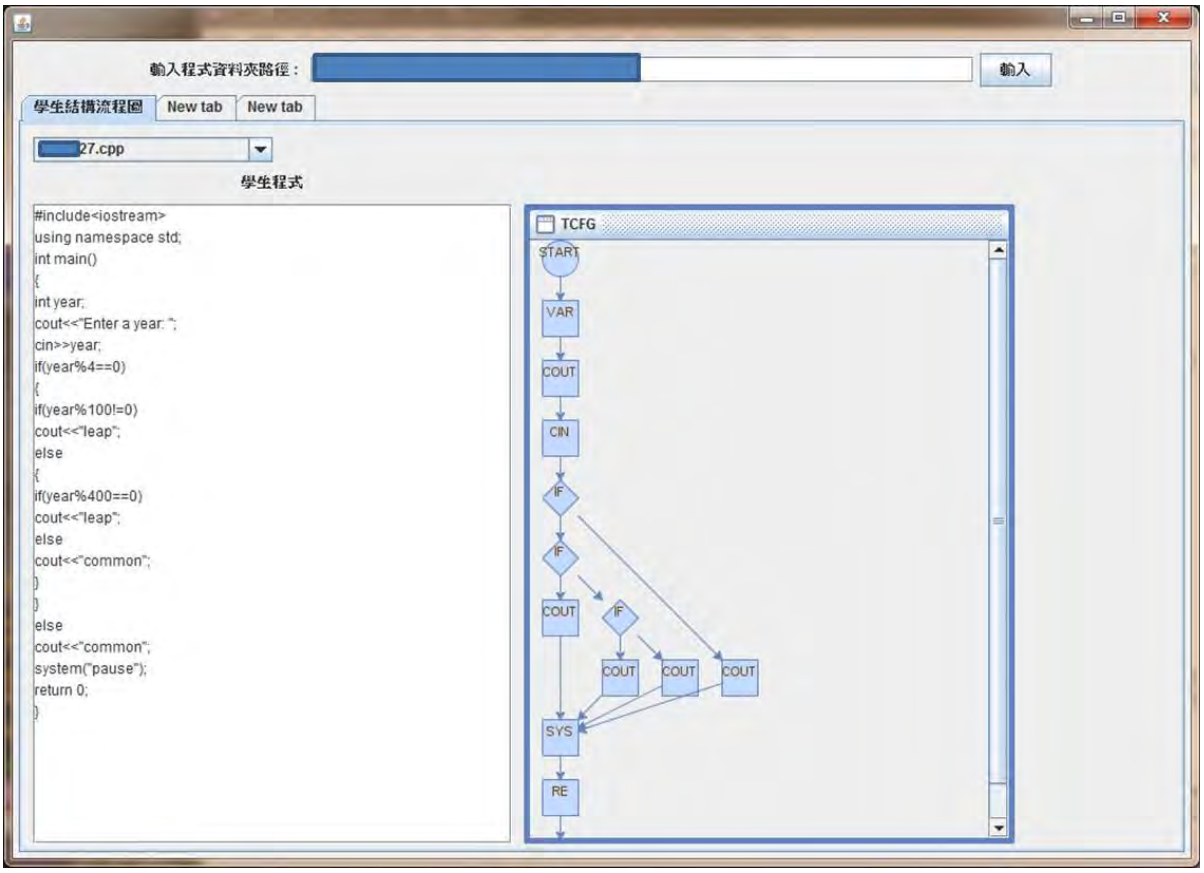
Programming involves many complex processes and knowledge, such as designing a program plan, generating program code to realize the program plan, and debugging and fixing the program if errors exist, thus students, particularly novice programmers, may encounter great difficulties (Robin, Rountree, and Rountree, 2003). Teachers and teaching assistants are helpful to diagnose the errors in student programs and to help students fix errors, but it causes heavy workload for teachers and teaching assistants. Some computer automated assessment systems have been developed to assist in assessing student programs (Ala-Mutka, 2005). These systems provide helpful feedbacks by employing two assessment approaches: dynamic assessment and static assessment. Dynamic assessment compiles and executes programs with several test input data to assess functional correctness and efficiency. Static assessment analyzes program code to assess coding style, design, and errors. Program code analysis is complex and difficult because program codes might have many semantic-preserving variations, such as different variable names, different function names, and different statement orders. To improve program code analysis, researchers proposed transforming program codes into semantic graphical representations based on control dependence and data dependence (Li, Pan, Zhang, and Chen, 2010; Wang, Su, Wang, and Ma, 2007; Xu and Chee, 2003). However, these studies hide semantic graphical representations inside the systems and do not visualize graphical representations to assist students in debugging their programs.

Students may have errors in planning and coding, but students might lack of awareness of errors in their programs. Syntax errors in coding can be detected by a program compiler, but it is difficult for students to detect errors in planning and semantic errors in coding. Studies of visualizing control structure of programs revealed that the program visualization, which maps programs to graphical representations, improved students' program comprehensibility and learning (Ben-Ari et al. 2011; Hendrix, Cross, and Maghsoodloo, 2002). In addition, Brusilovsky (1993) suggested that program visualization could be used as a debugging tool for novices. This study presents a work-in-progress prototype system to visualize the control structure of student programs in graphical representations for assisting novice students in debugging their programs.

2. The Prototype System and Possible Applications

A prototype system, named ProgramVisualAid, was developed to transform student programs into graphical representations of control structure. A structured program consists of three control structures: sequence, selection, and iteration. Figure 1 displays the graphical representations of a student program, which computes whether a year is a leap year or not. Each node denotes a code and each link indicates a control flow. The representations reveal that the program has three selection structures to check whether the year can be divisible by 4, 100, and 400. The third selection structure is inside the second selection structure and the second selection structure is inside the first selection structure.

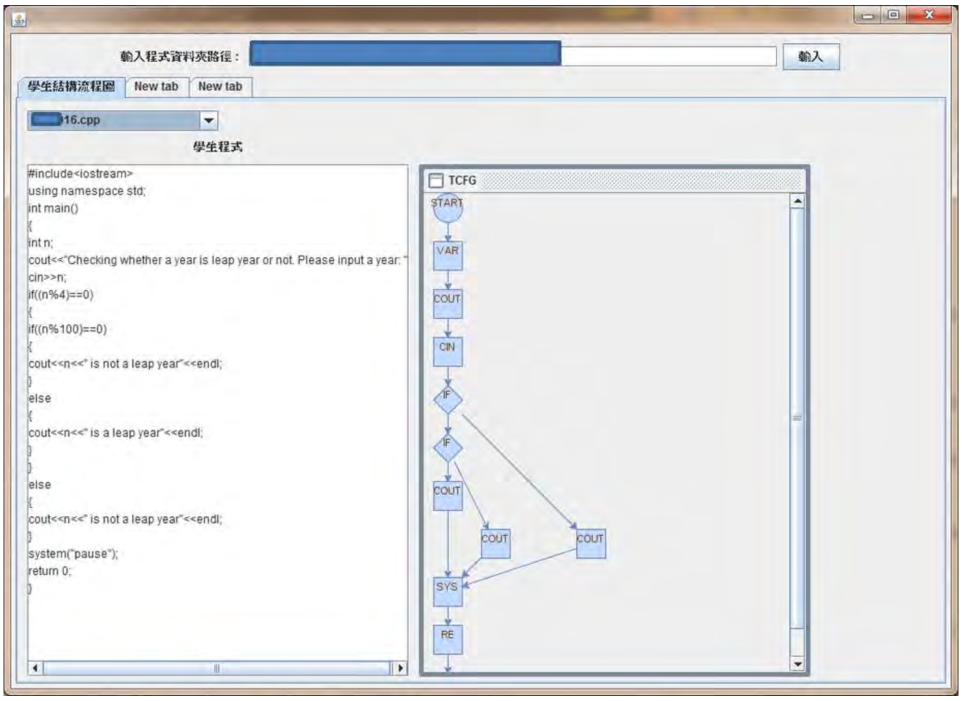
The visualization tool can be used as a debugging tool for students. First, students can check whether the transformed control structures of their programs match their program plans or not. If student programs have semantic errors in coding program plans, the transformed control structure will be inconsistent with the plan. For instance, the third selection structure in Figure 1 has two control flow links to separately link two "cout" codes. If the "else" code in the 16th line is missed, the graphical representation of the third selection structure will have a control flow link to sequentially link two "cout" codes. Students' awareness of the inconsistence leads students to fix the error.



```
#include<iostream>
using namespace std;
int main()
{
int year;
cout<<"Enter a year: ";
cin>>year;
if(year%4==0)
{
if(year%100!=0)
cout<<"leap";
else
{
if(year%400==0)
cout<<"leap";
else
cout<<"common";
}
}
else
cout<<"common";
system("pause");
return 0;
}
```

Figure 1. Control Structure Visualization of a Correct Leap Year Program.

Second, graphical representations of student programs can be compared to that of a model program to detect errors in planning. For instance, if the program in Figure 1 is a model program and the program in Figure 2 is a student program, the comparison of graphical representations of these two programs reveals that the student program lacks of a selection structure to check whether the year is divisible by 400 or not. We are implementing a comparison mechanism of comparing graphical representations of two programs to find out the difference of control structures. The difference between a student program and a model program might indicate errors of the student program plan. However, a problem might be solved by many variations of programs. We are also implementing dynamic assessment mechanism, which compiles and executes programs with several test input data to assess functional correctness, to find out more possible model programs. The system will compare a student program to all model programs and adopt the comparison result with the most similar model program.



```
#include<iostream>
using namespace std;
int main()
{
int n;
cout<<"Checking whether a year is leap year or not. Please input a year: "
cin>>n;
if((n%4)==0)
{
if((n%100)==0)
{
cout<<n<<" is not a leap year"<<endl;
}
else
{
cout<<n<<" is a leap year"<<endl;
}
}
else
{
cout<<n<<" is not a leap year"<<endl;
}
system("pause");
return 0;
}
```

Figure 2. Control Structure Visualization of a Wrong Leap Year Program.

3. Summary

This paper presents a work-in-progress research to develop a program visualization tool for transforming student programs into graphical representations of control structure. The tool can be used for assisting students in debugging their programs by checking whether the transformed control structure of their programs is consistent with their program plans. The graphical representations of student programs can be compared to that of a model program to detect errors in program plans. We are developing comparison and more analysis mechanisms to improve the visualization tool.

Acknowledgements

The authors would like to thank the support of the Ministry of Science and Technology, Taiwan (NSC 100-2511-S-155 -004 -MY3 and MOST 103-2511-S-155 -003)

References

Ala-Mutka, K. M. (2005) A survey of automated assessment approaches for programming assignments, Computer Science Education, 15 (2), pp. 83-102.
Ben-Ari, M., Bednarik, R., Ben-Bassat Levy, R., Ebel, G., Moreno, A., Myller, N., & Sutinen, E. (2011). A decade of research and development on program animation: The Jeliot experience. *Journal of Visual Languages & Computing*, *22*(5), pp. 375-384.
Brusilovsky, P. (1993). Program visualization as a debugging tool for novices. In *INTERACT'93 and CHI'93 Conference Companion on Human Factors in Computing Systems*, pp. 29-30.
Hendrix, D., Cross, J. H., & Maghsoodloo, S. (2002). The effectiveness of control structure diagrams in source code comprehension activities. *Software Engineering, IEEE Transactions on*, *28*(5), 463-477.
Li, J. Pan, W., Zhang, R. & Chen, F. (2010). Design and implementation of semantic matching based automatic scoring system for C programming language, In *Proceedings of Edutainment 2010*, LNCS 6249, pp. 247 – 257.
Robin, A., Rountree, J., and Rountree, N. (2003). Learning and teaching programming: A review and discussion, *Computer Science Education*, 13, no. 2, 137-172.
Wang, T., Su, X., Wang, Y. & Ma, P. (2007). Semantic similarity-based grading of student programs, *Information and Software Technology*, 49, 99-107.
Xu, S. & Chee, Y.S. (2003) Transformation-based diagnosis of student programs for programming tutoring systems. *IEEE Transaction on Software Engineering*, pp. 360 – 384.