Ogata, H. et al. (Eds.) (2015). Proceedings of the 23rd International Conference on Computers in Education. China: Asia-Pacific Society for Computers in Education

# Error Log Analysis in C Programming Language Courses

**Xinyu FU[a*], Chengjiu YIN[b], Atsushi SHIMADA[b], Hiroaki OGATA[b]**
[a]*Department of Faculty of Information Science and Electrical Engineering, Kyushu University, Japan*
[b]*Faculty of Arts and Science, Kyushu University, Japan*
fu.xinyu.277@s.kyushu-u.ac.jp

**Abstract:** Many universities choose the C programming language (C) as the first programming language to teach to students. As novice programmers, students frequently make simple mistakes such as syntax and typographical errors. Students often find it difficult to locate these errors, as students are not yet thoroughly familiar with C's syntax. This situation often causes students to consider programming very dull. It is therefore critical to provide clearer explanation in class, to prevent students losing interest in programming. This study aims to facilitate teaching and learning of C. We propose a system that undergraduate novice programmers may use to locate syntax errors in C. We analyze error types and resolutions using data collected during a programming course, and discuss key findings and their implications for programming education.

**Keywords:** C programming, programming education, learning analysis

## 1. Introduction

The C programming language is critical, as it is the most commonly learned first programming language. A sound understanding of C is very helpful to learning other programming languages. C is the first programming language taught to students at our university.

Novice programmers typically do not understand C's syntax very well. Students learning C frequently make simple errors, such as typographical errors or careless use of syntax. Though these errors are simple, novices typically find their resolution difficult. Novices may struggle to locate the cause of errors, and may find the nature of the error obscure. Students in this position readily come to consider programming especially dull and difficult. We have observed that, after attending a C course's first class, some students give up and do not attend the course again.

Robins and Rountree (2003) showed that students considered programming courses difficult; further, their study found that these courses often have among the highest dropout rates of courses offered. It is therefore necessary to facilitate the teaching and learning of C.

Perkins et al.(1989) classified students of programming into three types: movers, stoppers and tinkerers. Stoppers who encounter problems will stop attempting to find resolutions, and abandon hope of solving the problem on their own. In a discussion of visualizing solutions in programming education, Ala-Mutka (2004) suggested that "technical tools and visualizations are simply learning aids and materials. Teachers must thoroughly design their instructional approach to the issues in the course, and how the aiding materials are incorporated into education." In this context, we propose a system for locating syntax errors in C, and discuss our findings' implications for improving teaching materials.

Students learning C in our courses may use terminal software such as TERA-TERM to access a common server. We are able to gather C programming logs from the server through SFTP. Novice programmers' logs are a very useful resource. We aim to make these logs more useful to teaching and learning C.

At least three types of difficulty must be addressed in order to make teaching programming more effective.

- Novices are not aware that they are able to locate solutions through Internet searches.
- Classes have one or two teaching assistants (TAs), but typically have up to 50 students; this limits the amount of attention each student receives.
- Teachers may neglect to discuss simple errors while explaining course content.

In order to facilitate novice programmers' acquisition of programming expertise and understanding of error messages, and to promote teaching effectiveness particularly concerning easily neglected material, we propose a new system for analysis of error messages in teaching C.

Section 2 provides a review of relevant programming and computing education literature. Section 3 provides a description of our study. Section 4 presents the results of our analysis. Finally, we discuss our findings' implications, and indicate scope for further research.

## 2. Related Research

Robins and Rountree (2003) reviewed and discussed teaching and learning of programming. They identified "general trends comparing novice and expert programmers, programming knowledge and strategies, program generation and comprehension, and object-oriented versus procedural programming." Their study examined novices, and discussed education materials; however, most recent research examines improving learning of programming styles. Extant research examining effective education of novices is limited.

Hwang et al. (2012) discussed cooperative learning of ASP.NET using the WPASC (Web-based programming assisted system for cooperation). The research they made found that cooperative programming style is useful for many students. We consider that if the WPASC system does not effectively manage cooperative programming, less able students may not do their best to resolve errors in their programs.

Nagao and Ishii (2003) proposed an agent support system for C. This system is also a type of cooperative programming: students may share knowledge and error resolutions through agent software. Park et al. (2015) analyzed HTML and CSS syntax errors in a web-development course. They examined the JavaScript programming language, and used the openHTML editor system, to analyze difficulties that novices experienced in learning HTML and CSS' syntax.

Thus, extant research predominantly examines cooperative programming and self-education systems. Research examining initial education in the C language is limited. In our study, we propose a system that may facilitate students' understanding of why errors may be made, as well as how to resolve them. Further, we aim to essentially improve teaching of programming languages.

## 3. System Design and Implementation

Figure 1 illustrates our system's architecture. Students may connect to the server through terminal software TERA-TERM using his or her student ID and password. Students are not required to install any programming software; the compile software GCC (GNU Compiler Collection) has already installed in the server. Students can get themselves' workspace by using themselves account. All reports of GCC compiling will be stored in directory "./log". Students' compiling logs will be stored in the directory which named by student ID. We are able to gather logs form the server through SFTP. We collect and analyze students' logs, and provide feedback to teachers and students.

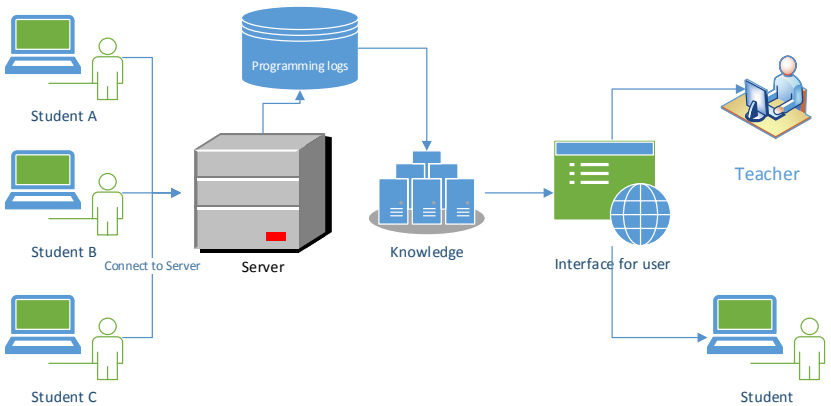


Figure1. The architecture of our system

### *3.1 Course*

We analyzed error messages with respect to the course schedule to better survey novices' syntax errors. Table 1 provides the course schedule, with topics and assessments organized by week. Since each week's topic is different, we anticipated that the collected syntax errors would be different also. Examination of errors thus illuminates which parts of course content are difficult for students to learn.

To easily distinguish topics in the analysis, we first established a method of naming exercise problems. For example, the log name "2_a-1.c" indicates that the log belongs to topic 2, and the difficulty level is "a". For every topic we have two exercises with same difficulty, "1" indicates that the log belongs to exercise one, ".c" means it is a C source file. Difficulty ranges from "a" to "c"; "c" is easier than "a." No problems are especially difficult; "a" is marginally more complex.

### *3.2 Participants, Data Source and Development Environment*

Participants: Log analysis included activity from 909 students attending a C programming course; 164 from autumn 2014 (Oct. ~ Feb. 2014), and 745 from spring 2015 (Apr. ~ Aug. 2015). All students were novices from first-year undergraduate students.

Data Collection: We collected student logs from assignments in all course components. Logs were collected via SFTP from remote servers. Presently, we have collected 53,505 errors.

Development Environment: In our college we teach C using TERA-TERM, and compilation with GCC. This programming environment is less difficult for novices, as they are not required to create the environment themselves. Error messages are collected from GCC reports.

### *3.3 Analysis Methods*

Analysis methodology was critical during the early stages of the system's preparation. Since we wished to understand the syntax errors that novices commonly make, and wish to effectively improve outcomes in C programming education, it was necessary to establish the types of errors novices typically made. Our first task was to analyze 11,581 error messages collected from 164 students in the autumn 2014 course, to identify common error types.

It would have been impractical to provide explanations and proposed resolutions for each of the 11,581 error messages. Further, to assign resolutions to individual error messages is uninformative. We found that many errors are of common types, for example "'n' undeclared (first use in this function)" and "'m' undeclared (first use in this function)." These are two different error messages, but they indicate the same type of error: a variable was not declared prior to its use. In this type of situation, it is illuminating to estimate the proportion of error messages "undeclared (first use in this function)," and determine this error type's frequency. Each error type's frequency was determined in this manner. Analysis of error types' frequency thus identifies each topic's most common error types.

Further, we found that, among the 11,581 error messages, 873 were reduplication errors. We considered that these were likely the result of students attempting to remove bugs in their code. We therefore examined theses logs to determine if students had attempted to resolve the errors they contained. If the logs were identical, we considered that the student had not attempted to rectify the code, and had simply compiled it again without alteration. By counting students' reduplication error messages in error types, and comparing their number with the total number of collected errors in that type, we established the number of instances of genuine code modification, which indicates the degree of difficulty students faced in resolving that error type. Analysis of reduplicate errors thus permits examination of the degree of difficulty involved in identifying errors.

Finally, we used error types identified in the autumn 2014 course to analyze logs from the spring 2015 course. New error message types will be added to our error dictionary as they are established.

### *3.4 Presentation*

Subsequent to analysis of the syntax error logs, we provided the system to students and teachers via Web pages. Users may use their account to browse results from our analysis. Three types of detailed information are provided.

First, the system is real-time.

The C programming course typically runs on weekdays; we therefore set our system to update error logs every 5 minutes from 8 am to 6 pm. Students may use their account to check for available explanations of recent errors they have made. We provide examples of similar functional programs in C, and resolution methods, on students' homepages. Teachers may use the system to examine common errors in real time, and provide proper explanation of these errors to students. We suggest that these facilities may promote students' understanding of C syntax.

Second, analysis results are refreshed during a fixed period.

We update the error logs at midnight, and use each day's logs to analyze syntax errors. Results are automatically sent to teachers by email before the next course. Using these results, teachers may summarily explain simple errors made in the previous topic. Additionally, as the results are stored in the system, results may be later used to optimize C language education materials. We suggest that teachers may be able to provide more targeted explanations of C programming to novices, and improve novices' satisfaction with their learning progress, using these results.

Finally, our system provides an opportunity to review learning.

Our system provides students with similar examples, to allow students to notice and understand errors as early as possible. If the error is of a type a student has successfully addressed before, the student's own earlier program may be provided as an example. This is a type of reflection-based learning, which allows students to more securely retain the proper syntax. If the student has not encountered the error before, the system permits provision of stock examples.

## 4. Results

This section discusses syntax errors students were unable to resolve on their own. We provide an overview of all error types, and then discuss errors' prevalence and process of change as students progressed through the courses' topics.

### *4.1 Overview*

Figures 2 and 3 list error types' frequency (overall count) and incidence of reduplication in the two semesters. A list of error types is also provided in table 2 in the appendix.

In Figure 2 and Figure 3, series 1 indicates error frequency; series 2 indicates incidence of reduplication.

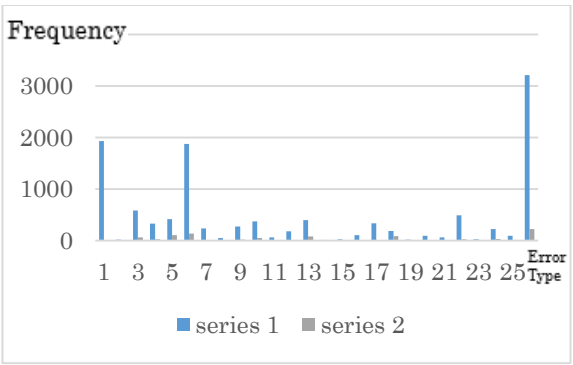


Fig 2. Errors distribution of 2014 autumn semesters

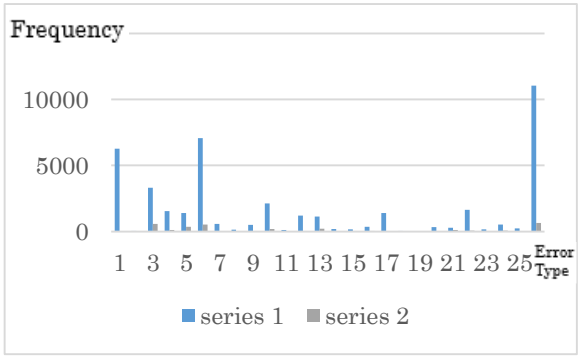


Fig 3. Errors distribution of 2015 spring semesters

Figures 2 and 3 indicate that error distribution was very similar between the two semesters. The most common error types are 1, 3, and 6. Review of these three error types' error logs indicated that these errors were typically caused by missing semicolons. Other frequently occurring error types were

4, 5, 10, 12, 13, 17, and 22. These error types were mostly caused by limitations in students' knowledge of C's syntax, though some instances were caused by mistyping. Type 26 refers to all other errors; this type contains many unclassified types of error messages. Analysis of error types is ongoing; collection of more error messages will eventually allow more accurate analysis of these errors.

Figure 4 presents incidences of reduplicative errors in the two semesters. Similarly to the error frequency distribution, reduplicative errors of types 3, 5, 6, 10 and 13 were most frequent; however, this result alone does not allow the inference that these error types were difficult to locate. Subsequent to completion of this paper, we will analyze each error type's average reduplication rate. Rates of error reduplication may be used to more effectively teach C.

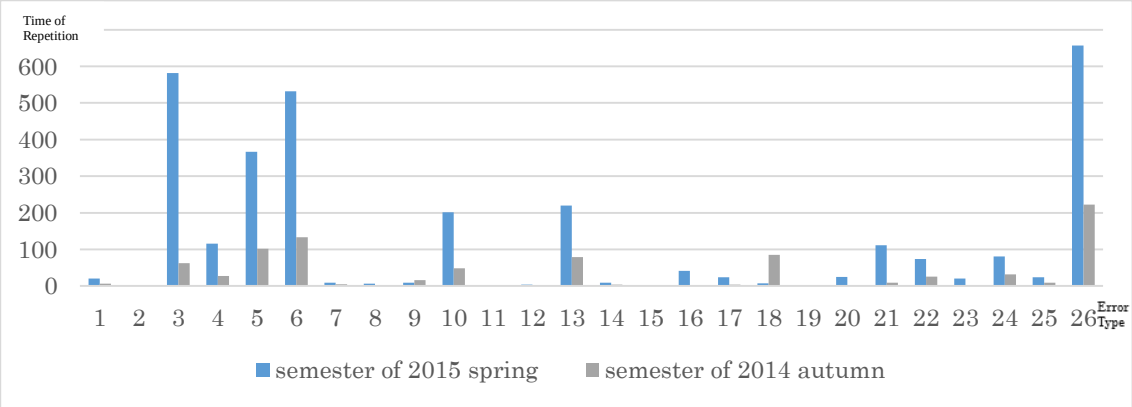


Figure 4. Reduplicative happen times of the two semesters

## *4.2 Analysis of Error Types by Topic*

Since we did not name the exercises taught during the 2014 autumn semester course, it is not straightforward to distinguish errors by course topic in that semester. We therefore use errors generated during the 2015 spring semester course for analysis in this section. Figure 5 therefore presents some typical error frequencies separated by course topic for the 2015 spring semester. Error types are the same as in section 4.1. The 2015 spring semester has currently progressed through topics 1–7; we also have some results for topic 8 and 9, as it seems that some particularly motivated students have begun these topics.

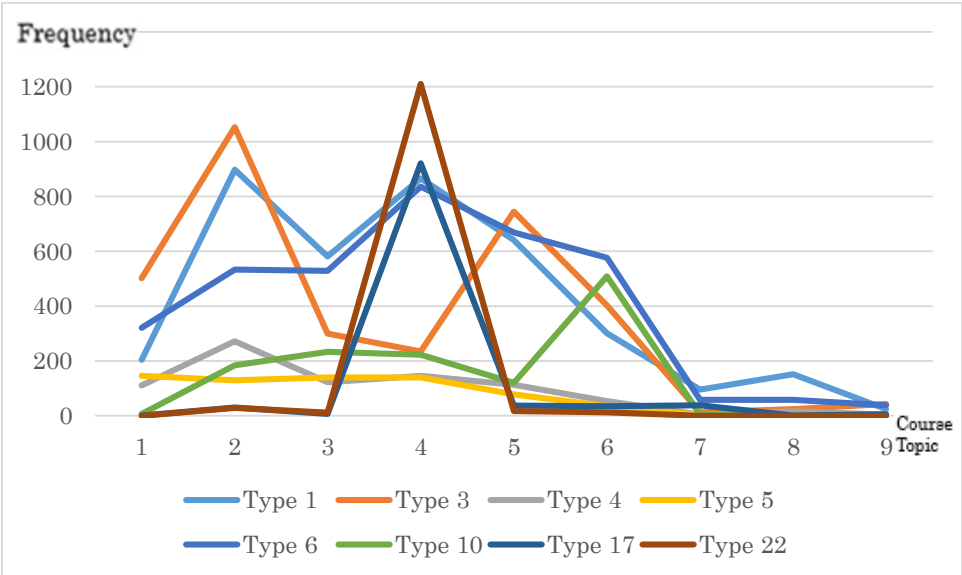


Figure 5. Typical Errors' happen times during topic 1 to topic 9

Figure 5 clearly illustrates that the frequencies of error types 1 and 6 do not decrease as the course progresses (these errors are mostly caused by missing semicolons). Teachers should therefore

emphasize these error types throughout courses. Frequency rates of error types 17 and 22 increase sharply in topic 4; topic 4 addresses mathematical functions. Figure 5 hence indicates that beginners are particularly prone to these two error types when learning mathematical functions. We will collect typical examples of these error types, and use them to make appropriate suggestions regarding more effective ways of teaching C. Further, we will list examples of these types of error on our system's web page for students who require example errors.

### *4.3 Strong Emphasis on Learning by Similar Examples*

As described in section 3.4, we wish for our system to promote learning through reflection. We therefore provide opportunities for students to review previous programming they have completed that contained the same error they are presently attempting to resolve. In this way, we aim to strengthen the student's awareness of that error type. Figure 6 is an example of a student's programming that contains an error. The error is highlighted; it is an example of a mismatched pair of "{}."

A given error message may sometimes cause by different errors. We intend to list typical examples of such error messages in our system's web page. Students may refer to these examples to improve their chances of locating errors in their programs. Figure 7, and appendix figures 1–3, provide examples of programming containing Type 1 errors (undeclared variables or ";" or "}" missing in previous row). Further, students may mentally review the causes of their error. This effectively strengthens students' retention of error-related information, and assists novices in noticing careless errors.

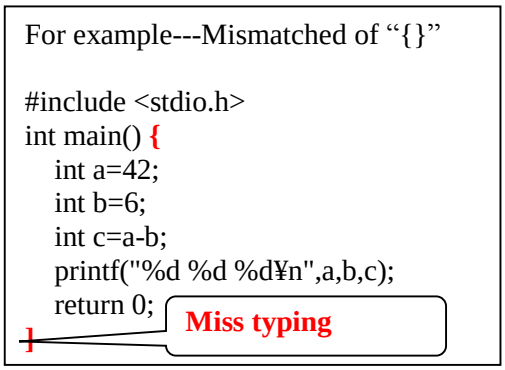


Figure 6: example of wrong programming

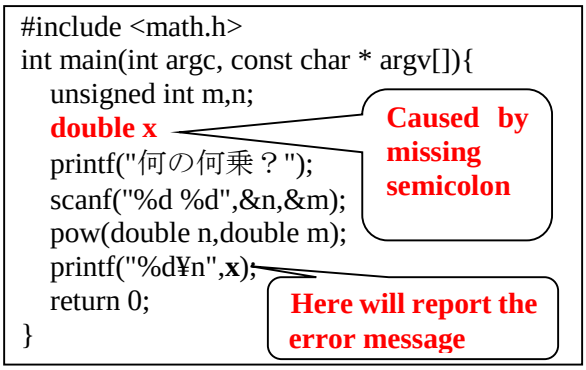


Figure 7: example 1 of Type1

### *4.4 Achieving Real-Time Analysis*

This section describes how real-time analysis was achieved. A considerable amount of extant research has discussed real-time analysis of learning (e.g., Rao and Kumar, 2008; Huang et al., 2006; Barto, et al., 1995). Real-time analysis allows effective research of programmers' learning style, and is generally beneficial to research examining learning.

Our system updates the error logs from a remote server every five minutes. As discussed in section 3.4, our system refreshes our error logs during the day. This is real-time in two aspects: First, students may access analysis results during course time. A student who encounters a problem may log into our system through a web page, and that student's recent error types will be listed on his or her personal homepage. Students may review their recent errors, and may retrieve relevant examples for the present error type. Second, teachers may access an overview of all error types with times of occurrence, and may target specific student errors at any time during teaching. Teachers may use these data to further explain specific course content to students as necessary.

## 5. Conclusions and Future Work

In this paper, we outline a syntax error analysis system for the C language. Our results may effectively assist learning of C, and optimize the C teaching process. Knowledge of common error types indicates

errors that are easily made through carelessness, and information requiring further explanation during course time.

We have collected 909 students' programming logs from programming courses conducted in autumn 2014 and spring 2015. We classified and analyzed errors in the logs. Our work makes the following four main contributions.

First, through analysis of syntax errors, we characterize difficulties that students face when learning C. To our knowledge, our study is the first to offer analysis of novices' coding errors in C, and use the results to suggest ways to optimize of education materials. Our results show the frequency of errors, and indicate the difficulty of their correction.

Second, we use real-time analysis. Students may use the system during class. Shilpi and Kumar (2008) discussed the effect of real-time programming assessment on students' Java programming style. They tracked students' coding styles in real time, and made suggestions regarding how students' coding styles may be improved. In our study, we use real-time analysis to provide students with samples and suggestions as they make errors.

Third, we use sample code taken from a student's work if a student repeats a certain error. This is a form of knowledge consolidation in learning programming.

Fourth, we statistically analyze syntactical errors, providing feedback to teachers. This feedback may guide materials development in C programming education.

Given that we currently assemble error logs though syntax error messages obtained from the compiler, the causes of errors we receive are not exact. It is very difficult to obtain exact error causes. We intend to incrementally improve our error dictionary by collecting and analyzing more error messages. This will allow more accurate categorization of errors, and may not only explain individual error messages, but also allow identification of bugs throughout complete programs.

Further, we intend to extend our study's scope to address not only frequency, but also errors' ease of commission or avoidance. To examine content difficulty level, we intend to add a function for our system to track error resolution times, and acquire average resolution times for each error type, in order to analyze errors more accurately. We intend to finish our system quickly, and deploy it in a C programming course, in order to verify its usefulness. We also plan to discuss the method which is proposed in Mouri, et al. (2014, 2015).

Finally, as students in C programming courses are a mixture of computing majors and general education students, we wish to analyze potential differences in rates of error types between majors, and adapt course teaching accordingly to each major. We intend to implement an online system for all novices studying C programming, and collect and examine more error messages and error types. It is our hope that our system may serve a greater number of learners, and be useful in more complex programs.

## Acknowledgement

This research work was supported by the Grant-in-Aid for Scientific Research No. 25282059, No. 26560122, No. 25540091, and No. 26350319 from the Ministry of Education, Science, Sports, and Culture in Japan and "Research and Development on Fundamental and Utilization Technologies for Social Big Data" (178A03), the Commissioned Research of the National Institute of Information and Communications Technology (NICT), Japan.

## References

A. G. Barto, S. J. Bradtke, S. P Singh (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, Vol. 72, pp.81-138.

A. Robins, J. Rountree, and N. Rountree (2003). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education*, Vol. 13, No. 2, pp. 137-172.

D. Perkins, C. Hanconck, R. Hobbs, F. Martin and R. Simmons (1989). Conditions of learning in novice programmers. *In Soloway & Spohrer: Studying the Novice Programmer*, pp. 261-279.

G.-B. Huang, Q.-Y. Zhu and C. Keong (2006). Real-Time Learning Capability of Neural Networks. *IEEE Transactions on neural networks*, Vol. 17, No. 4. , pp. 863-878.

H. P. Thomas , B. Dorn and A. Forte (2015). An analysis of HTML and CSS syntax errors in a web development course. *ACM Trans. Comput. Educ*. Vol. 15, No. 1, pp. 4:1-4:21.

K. Ala-Mutka (2004). Problems in Learning and Teaching Programming - a literature study for developing visualizations in the Codewitz-Minerva project. *Codewitz Needs Analysis*. Literature Study.

K. Mouri, H. Ogata, N. Uosak (2015). Ubiquitous learning analytics in the context of real-world language learning. *Proceedings of LAK15*, pp. 378-382.

K. Mouri, H. Ogata, N. Uosaki, S. Liu (2014). Visualization for Analyzing Ubiquitous Learning Logs, *Proceedings of the 22nd International Conference on Computers in Education (ICCE 2014)*. pp. 461-470,

K. Nagao, N. Ishii. (2003). Evaluation of Learning Support System for Agent-Based C Programming. Knowledge-Based *Intelligent Information and Engineering Systems Lecture Notes in Computer Science* ,Vol. 2774, pp. 540-546

S. Rao and V. Kumar (2008). A theory-centric real-time assessment of programming. *In proceedings of the Eighth IEEE International Conference on Advanced Learning Technologies (ICALT)*, Cantabria, Spain, July, pp. 139–143.

W.-Y. Hwang, R. Shadiev, C.-Y. Wang and Z.-H. Huang (2012). A pilot study of cooperative programming learning behavior and its relationship with students' learning performance. *Computers & Education*, pp. 1267-1281.

Appendix table 1. Weekly Overview of the Course Schedule

| Week | Topics | Assessments |
|---|---|---|
| 1 | Introduce to C Language | 3 Exercises of Using "Printf" Statement |
| 2 | Variables | 6 Exercises of Variables |
| 3 | Functions | 6 Exercises of Using "Scanf" Statement and Operator Symbol like "+", "-", "*", "++". |
| 4 | Mathematical Functions | 6 Exercises of Mathematical Functions |
| 5 | Decision Making Structures If-else | 6 Exercises of If-else |
| 6 | Multiple If-else and Switch-case | 6 Exercises of Multiple If-else and Switch-case |
| 7 | For Loop | 6 Exercises of For Loop |
| 8 | Array | 6 Exercises of Array |
| 9 | Multi-dimensional Array | 6 Exercises of Multi-dimensional Array |
| 10 | Multiple For Loop | 6 Exercises of Multiple For Loop |
| 11 | While--Do-while Loop | 6 Exercises of While--Do-while Loop |
| 12 | String Functions | 6 Exercises of String Functions |
| 13 | User-defined Functions | 6 Exercises of User-defined Functions |
| 14 | File I/O | 6 Exercises of File I/O |
| 15 | Programming Practise | 1 Exercises of Final Test of Semester |

Appendix table 2. Error types with their frequency and reduplicative happen times

| | Error Type | Semesters of 2015 spring(topic from 1 to 9) | | Semesters of 2014 autumn (all topics) | |
|---|---|---|---|---|---|
| | | Frequency | Times of Repetition | Frequency | Times of Repetition |
| Type 1 | Undeclared variables, or ";" or "}" is missing in previous row | 6276 | 20 | 1929 | 6 |
| Type 2 | Statement is out of main class | 52 | 0 | 19 | 0 |
| Type 3 | Full-width characters are used | 3318 | 582 | 583 | 62 |
| Type 4 | Missing punctuation (e.g. "=" or "," or ";" or "asm" or "__attribute__")—semicolons were most frequently missing | 1557 | 116 | 330 | 27 |
| Type 5 | Mistyping of standard library | 1405 | 367 | 414 | 102 |
| Type 6 | Missing semicolon | 7068 | 532 | 1872 | 133 |
| Type 7 | Redeclaration of variables | 581 | 9 | 235 | 5 |
| Type 8 | Redeclaration of variable type | 143 | 6 | 49 | 1 |
| Type 9 | Undeclared variables (particularly, mismatched symbols and mistyping of symbols) | 517 | 9 | 274 | 16 |
| Type 10 | Syntax errors (invalid operand or invalid suffix) | 2122 | 201 | 370 | 48 |
| Type 11 | Mismatch of "{" (particularly, "{" after main) | 108 | 0 | 61 | 0 |
| Type 12 | Mismatch of quote marks, or mismatch of "<>" | 1200 | 4 | 178 | 0 |
| Type 13 | Missing "}" at the end of code | 1146 | 220 | 394 | 79 |
| Type 14 | Two main classes in one program | 185 | 9 | 13 | 4 |
| Type 15 | Missing semicolon before "return" | 164 | 0 | 26 | 0 |
| Type 16 | Misuse of switch statement (with or without use of "break") | 368 | 41 | 104 | 0 |
| Type 17 | Misuse of mathematical functions | 1395 | 24 | 336 | 4 |
| Type 18 | Mistakes on array declaration | 49 | 7 | 185 | 85 |
| Type 19 | Unmatched variable type for array | 11 | 0 | 20 | 2 |
| Type 20 | Mismatch of "{}" | 334 | 25 | 91 | 1 |
| Type 21 | Mismatch of "{}": missing semicolons or comma before "{" | 295 | 111 | 59 | 9 |
| Type 22 | Misuse of pow function | 1636 | 74 | 492 | 26 |
| Type 23 | "," used after variables, not "." | 176 | 20 | 26 | 0 |
| Type 24 | Missing semicolon or comma | 527 | 81 | 221 | 32 |
| Type 25 | Unmatched data type | 246 | 24 | 91 | 9 |
| Type 26 | Other error type | 11045 | 657 | 3209 | 222 |

```
#include <stdio.h>
#include <math.h>

int main(){
   dobule a, b, c, k, x1, x2;
   printf("Enter 3 coeefficents of a quadratic equation, a, b, c = ");
   scanf("%f %f %f", &a, &b, &c);

   k = pow(b, 2)-4*a*c;
   if (k <= 0){
      printf("no real root");
   }
   else if (k == 0.0){
      x1 = -1.0*b/(2*a);
      printf("dobule root, x = %f", x1);
   }
   else {
      x1 = (-1.0*b+sqrt(k))/(2*a);
      x2 = (-1.0*b-sqrt(k))/(2*a);
      printf("x1 = %f, x2 = %f", x1, x2);
   }
   return 0;
}
```

**Caused by miss typing of double. This is a very simple error. However sometimes is not easy to notice.**

Appendix figure 1: example 2 of Type1

```
#include <stdio.h>

int main(){
   int i, j, k, m[3][3], n[3][3];

   for (i=0; i<3; i++){
      for (j=0; j<3; j++){
         printf("(%d, %d) = ", i, j);
         scanf("%d", &m[i][j]);
      }
   }
   for (i=0; i<3; i++){
      for (j=0; j<3; j++){
         n[i][j] = 0;
         for (k=0; k<3; k++){
            n[i][j] += m[i][k]*m[k][j];
         }
      }
      printf("%d %d %d¥n", n[i][0], n[i][1], n[i][2]);
   }
   return 0;
```

**Caused by mismatched of "{}", the program will stop here. When try to write a long program, it is very necessity for the match of "{}". Furthermore standard program written format is very important.**

Appendix figure 2: example 3 of Type1

```
#include <math.h>;
int main(int argc, const char * argv[]){
   unsigned int m,n,x;
   a=pow(double x,double y);
   printf("%d¥n",a);
   return 0;
}
```

**Use variable of a before declared it.**

Appendix figure 3: example 4 of Type1