A Web Support System for Submission and Handling of Programming Assignments

Kurt Nørmark

Abstract: *Individual submission of programming assignments should be considered in all introductory programming courses. We describe a custom web support system for submission and management of programming assignments in an introductory C programming course. Experience from the first time use of the system is reported. In addition, we compare the pattern of use with the results of the final exam in order to reveal a possible impact of the programming assignments. We summarize the lessons learned in preparation for improving the system prior to the next round of use in the fall of 2011.*

Keywords: *Programming assignments, web support system, C programming course.*

INTRODUCTION

This paper reports about the use of a web support system for submission and management of programming assignments an introductory C programming course. Often, a standard web support system such as Moodle [4] is used for such purposes. In the paper we argue that a custom built web support system should be developed which is tailored to the specific ideas and needs in the course.

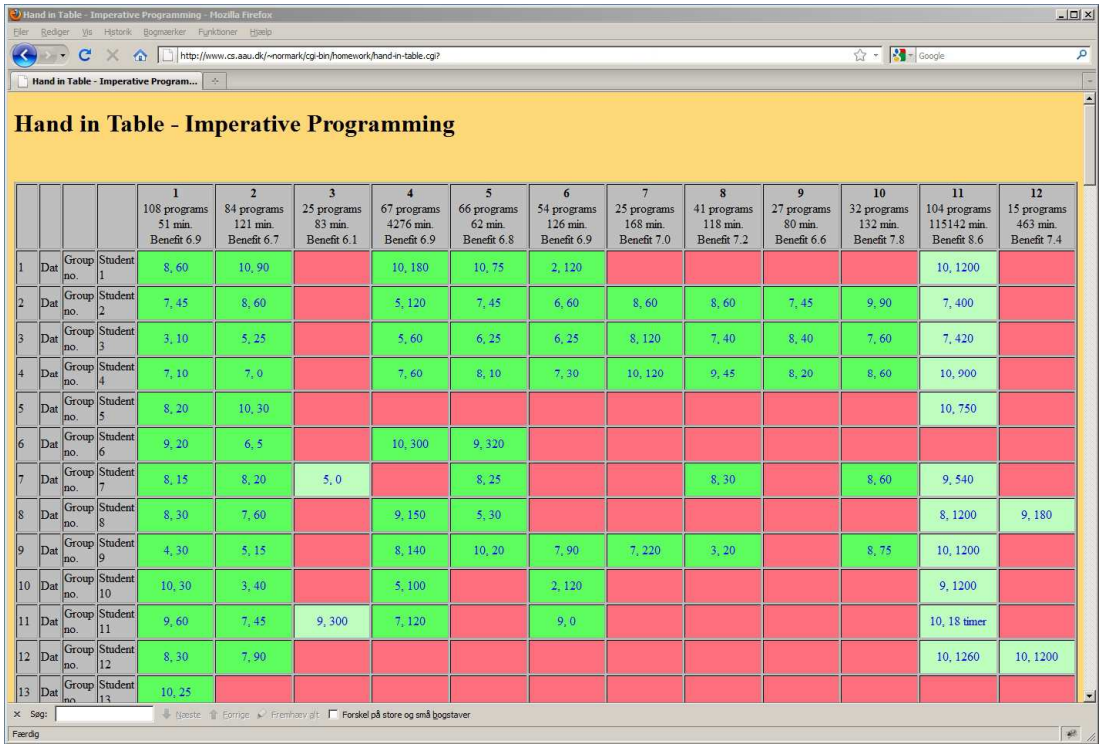
DESCRIPTION OF THE COURSE

The C programming course that we discuss in this paper has a weight of 5 ECTS points, it is structured in 15 lectures, and revamped from an earlier course with less ECTS points. As part of the course extension it was decided to introduce submission of individual home programming assignments - roughly one assignment for each regular lecture. The last such assignment was compulsory, and it was the basis of the final oral course exam. The starting point of the exam is a discussion of the submitted program. In total, 115 students joined the course at Aalborg University in the fall of 2010. In the appendix we briefly outline and characterize each of the programming assignments of the course.

From a student's point of view, a program is submitted via upload on a simple web page. For identification purposes the students must supply a valid user name and password. In addition the student is asked about the amount of time spent on the exercise (in minutes), the estimated personal benefit (on a scale from 1 to 10 where 1 means no benefit at all, 5 means a satisfactory benefit, and 10 means a large benefit), the status of the program, and a possible textual comment. To boost the motivation of the students it was decided that the students get individual feedback on each submission.

From the teacher's point of view, the main interface to the system is a large table where students are listed vertically, and the exercises are listed horizontally. An example of such a table (with anonymous student data) is shown in Figure 1. The regular homework assignments, as discussed in this paper, are number 1, 2, 4, 5, 6, 7, 8, 9 and 10 as represented in the columns of Figure 1. In each of the inner table cells the status of a single program submission is summarized, indicating benefit and time consumption. The color of a cell represents feedback status. Each of the inner table cells is linked to a submission page that shows the details of a single submission. Most important, the submission page shows the C program and the status information. In addition, a text box is present which allows the course teacher to give feedback to the student. The feedback is sent via university email. If feedback has already been sent (for instance by a teaching assistant) the existing feedback is also presented on the submission page.

It is well-known that it requires a lot of resources to manually asses programming homework from a large class of students [1]. This observation has spawned a large amount of work on automatic assessment of programming assignments. (This work will be reviewed briefly in a later section of this paper). It is therefore of utmost importance to minimize the overhead related to receiving the programs, handling them, and sending the feedback. A custom system for a particular course, built to satisfy the preferences of the involved teachers, is interesting in that respect compared to use of a general-purpose E-learning system with broad applicability in mind.



				1 108 programs 51 min. Benefit 6.9	2 84 programs 121 min. Benefit 6.7	3 25 programs 83 min. Benefit 6.1	4 67 programs 4276 min. Benefit 6.9	5 66 programs 62 min. Benefit 6.8	6 54 programs 126 min. Benefit 6.9	7 25 programs 168 min. Benefit 7.0	8 41 programs 118 min. Benefit 7.2	9 27 programs 80 min. Benefit 6.6	10 32 programs 132 min. Benefit 7.8	11 104 programs 115142 min. Benefit 8.6	12 15 programs 463 min. Benefit 7.4
1	Dat	Group no.	Student 1	8, 60	10, 90		10, 180	10, 75	2, 120					10, 1200	
2	Dat	Group no.	Student 2	7, 45	8, 60		5, 120	7, 45	6, 60	8, 60	8, 60	7, 45	9, 90	7, 400	
3	Dat	Group no.	Student 3	3, 10	5, 25		5, 60	6, 25	6, 25	8, 120	7, 40	8, 40	7, 60	7, 420	
4	Dat	Group no.	Student 4	7, 10	7, 0		7, 60	8, 10	7, 30	10, 120	9, 45	8, 20	8, 60	10, 900	
5	Dat	Group no.	Student 5	8, 20	10, 30									10, 750	
6	Dat	Group no.	Student 6	9, 20	6, 5		10, 300	9, 320							
7	Dat	Group no.	Student 7	8, 15	8, 20	5, 0		8, 25			8, 30		8, 60	9, 540	
8	Dat	Group no.	Student 8	8, 30	7, 60		9, 150	5, 30						8, 1200	9, 180
9	Dat	Group no.	Student 9	4, 30	5, 15		8, 140	10, 20	7, 90	7, 220	3, 20		8, 75	10, 1200	
10	Dat	Group no.	Student 10	10, 30	3, 40		5, 100		2, 120					9, 1200	
11	Dat	Group no.	Student 11	9, 60	7, 45	9, 300	7, 120		9, 0					10, 18 timer	
12	Dat	Group no.	Student 12	8, 30	7, 90									10, 1260	10, 1200
13	Dat	Group no.	Student 13	10, 25											

Figure 1. *A table that shows an overview of the all submitted programs.*

As mentioned, the last programming assignment is an exam assignment which is larger and more complex than the regular programming assignments. (This is exercise 11 in the table shown above). The students have approximately a week for solving the exam programming exercises (side by side with doing other kinds of work). The exam program is uploaded in the same way as the other programming assignments. If students receive help from others, it is required that they carefully describe this help in a form, which they must sign and deliver to the course secretary on a sheet of paper. This kind of exam is similar to a so-called *mini project programming exam* which has been described in a separate paper [5].

The web system allows extraction of all the submitted exam programs to a local hard disk, organized in separate directories named after the (unique) user names of the students. In that way it is easy to compile, run, and annotate each of the exam programs.

An electronic annotation system has been developed, such that the teacher is able to make notes about the exam programs while evaluating these. Previously, such notes have been written on printed copies of the source program. The annotation system is developed for the Emacs text editor [3]. Each annotation consists of a short text associated to a given textual region in the program. (A region is identified with a combination of a position, textual prefix, and textual suffix). Flexible navigation in between

the annotations is provided for. The annotations are used as the starting point in the discussion of the program at the oral exam.

EXPERIENCE

In this section we will report on the experiences from the first time use of the homework web support system. This section is based on the data collected during the use of the system, juxtaposed with the grades (pass/non-pass) obtained by the participating students at the final oral exam.

Of the 115 participating students (registered at the beginning of the course) 104 handed in an exam program, 96 attended the oral exam, and 79 passed the exam. We find that this ``degression'' is acceptable.

Exercise	Number of students	Average Benefit	Median Time Consumption
1	107	6.9	30
2	83	6.7	35
4	66	6.9	100
5	66	6.8	35
6	54	6.9	120
7	25	7.0	120
8	41	7.2	90
9	27	6.6	75
10	32	7.8	120
11 (Exam)	104	8.6	1200

Figure 2.

Number of exercises submitted	Passed the final exam	Failed the final exam
0	0	0
1	3	6
2	9	8
3	5	6
4	16	1
5	13	1
6	7	1
7	10	1
8	9	1
9	7	0
Sum	***79***	***25***

Figure 3.

During the course the participating students submitted 501 individual programs in total (not counting the exam programs). Of these submissions, 484 had a self-estimate of benefit for the student, and 474 contained a reasonable self-estimate of the time consumption. The second column of the table in Figure 2 shows the number of students submitting programs for each of the exercises. The third column shows the average benefit on a scale from 1 to 10. The fourth column shows the median of time consumptions (in minutes). The benefit and the time consumptions are taken from the students own declarations, as provided when uploading the program. It should be noticed that only 87 (of 104) students provided information about the time consumption of the exam exercise.

It is clear that almost all students were eager to submit a program in the first couple of lectures. After this good start, less than half of the students participated. This is not satisfactory. It also appears that the students, consistently through all exercises, evaluate their benefit between 6.6 and 7.8. We are pleased with these numbers. The time consumptions quite naturally vary between the exercises (cf. the appendix).

With respect to the exam exercise, the median time consumption is in the neighborhood of 20 hours. The benefit score of 8.6 indicates that the students learn a lot

from the exam exercise. It is very satisfactory that the many hours spent on this programming exercise are evaluated as returning a high yield.

The table in Figure 3 shows the distribution of the 104 students who submitted the final exam exercise. The students are distributed according to two dimensions: (1) The number of exercises they have submitted (vertically) and (2) their grade (passed/not passed) after the final exam (horizontally). Thus, each of the 104 students who submitted the exam program is only counted once in the table cells of column 2 and 3 in Figure 3. So, for instance, we see from the table that among the students, who only submitted a single exercise, 3 passed the final exam and 6 failed. It should be noticed that among the 25 students listed as not passing the oral exam, 17 failed at the oral exam, and 8 students did not show up for the exam.

It is clear from the table that the majority of the students (20 out of 25), who did not pass the final exam, have submitted less than four programs during the course. It can also be seen that more than 90% of the students, who solved four or more homework exercises during the course, passed the final exam. From this it may be tempting to conclude that the impact of the homework program exercises has been quite substantial, relative to the exam results. At least, in the fall of 2010, there is a strong correlation between 'submitting many programming assignments' and 'passing the final oral exam'.

LESSONS LEARNED

In this section we will describe the lessons we have learned from using the system in the fall semester of 2010. We also describe some improvements that we have introduced as preparation for use of the system in future semesters.

It is time consuming to provide individual feedback on each program, which has been submitted by a large class of students. The feedback we offer is a few qualitative remarks about the submitted programs, which only can be provided by actually reading them. After getting started (after having seen the first few programs) it takes only a few minutes per program to provide this feedback. We do not think it is possible to automate this part of the system. It may, however, be possible to provide for easy inclusion of 'standard feedback', selected from a (more or less constant) list of formulations. Such a feature has been introduced prior to the next round of use.

It is very important to streamline the teacher's working process with respect to giving feedback. The web system is able to send the feedback by email to the students. It would have been too time consuming to do this from a standard email client. In addition we have provided for direct navigation to the submission of 'the next student in line'. With this addition, it is not necessary to bring up the table as shown in Figure 1, scroll the table, and navigate to the next student who have submitted an assignment.

In the fall of 2010 we did not compile and execute many of the submitted programs. Instead, we read the program in preparation for giving qualitative feedback. However, in a few occasions (for instance if the student reflects a problem with compilation or execution) it is useful or necessary to download the program, compile it, and (if possible) to run it. It takes additional time and effort to manage this download process. Therefore, we have provided for *bulk download* of all submitted programs, from the repository of the web system to directories that reflect the student-ids, and to C source files with fixed names. In that way, it becomes much more realistic to compile and run a program, if it turns out to be necessary. It should be noticed - as a security measure - that we never execute a program before having inspected it.

It is problematic that the time and efforts used to comment on a single student's program is not used for the benefit of many more students. It is typically the case that many of the problems encountered in a single program are repeated in a number of programs from other students. We would like, in a smooth and flexible way (semi-automatically if possible) to be able to choose some programs and the accompanying feedback, and to display these for all students. Until now, we have only done this to a limited degree, as part of the course lectures (reflecting on common problems in the latest homework assignment). It is problematic, however, to expose the weaknesses of single student to all fellow students (even if it is done without mentioning names or other id). It would be a serious concern if students refrain from submitting programs because they fear to see their programs 'on public display'. In the future we will add a check box via which each student can give permission to use the program for public (but anonymous) scrutiny.

Finally, but not least, we have observed that reading and studying many programs - written by students - is extremely valuable for tuning the teaching to the actual needs of the students. As the course teacher and lecturer, you get very useful information about the actual skills of the students from the submitted programming assignments. As such, we hypothesize, that it will be possible to improve the teaching considerably on the basis of the insight harvested from reading a lot of program submissions.

RELATED WORK

E-learning systems, such as Moodle [4], typically support a general type of assignment that includes file upload coupled with some kind of grading. As argued in this paper, we find it necessary to specialise this facility, in order to streamline the work process of the teacher.

A lot of work has been carried out in the area of automatic assessment of programming assignments. A recent review of this work exists [2]. Automatic assessment has not been a direct theme in the work described in this paper. Rather, a smooth manual review process carried out by the course teachers has been chosen. Nevertheless, use of automatic assessment, as a supplement to manual assessment, could undoubtedly be attractive in a future version of the system.

An interesting paper by Ala-Mukta et al. [1] recommends mandatory submission of programming assignments, and it emphasizes the need for a combination of automatic assessment and feedback from experts. In addition, this paper explains an approach where students can learn from automatic feedback as often as they desire.

CONCLUSIONS

From the experience gained in 2010 the main conclusions are that (1) submission of many programming assignments seems to enhance the likelihood of passing the C programming course, (2) a flexible teacher interface to the system is crucial for giving qualitative feedback to each student, and (3) the time spent on reading many programs submitted by the students can be used to adjust the teaching to actual needs.

REFERENCES

[1] Kirsti Ala-Mutka and Hannu-Matti Järvinen. Assessment process for programming assignments. In Proceedings of the IEEE International Conference on Advanced Learning Technologies, pages 181–185. IEEE Computer Society, 2004.

[2] Petri Ihantola, Tuukka Ahoniemi, Ville Karavirta, and Otto Seppälä. Review of recent systems for automatic assessment of programming assignments. In Proceedings of the 10th Koli Calling International Conference on Computing Education Research, Koli Calling '10, pages 86–93. ACM, 2010.

[3] Bil Lewis, Dan LaLiberte, and Richard Stallman. GNU Emacs Lisp Reference Manual. The Free Software Fundation Inc, July 2009. Related to version 23.3 of GNU Emacs.

[4] Moodle. http://moodle.org.

[5] Kurt Nørmark, Lone Leth Thomsen, and Kristian Torp. Reflections on the teaching of programming, chapter Mini Project Programming Exams, pages 229–243. Springer Verlag, LNCS 4821, 2008.

APPENDIX

The following programming homework assignments were part of the 2010 version of the course, which we have described in this paper. The numbering of the assignment corresponds to the numbering used in Figure 1 and Figure 2.

1. A C program that converts a number of seconds to normalized hours, minutes and seconds. Trains integer divisions / with remainders %.
2. A continuation of the previous assignment, where nice and natural output is emphasized. Trains conditional control structures and expressions.
4. A program that calculates a 'leaving time' given an 'arrival time', speed, and distance. Trains simple functions with parameters.
5. A program for a simple ATM that calculates which bills to dispense when a given amount of money is requested. Trains output parameters (call by reference).
6. A program that calculates the area under a curve by use of the trapezoidal rule, and a given formula. Trains divide and conquer decomposition and function parameters.
7. A function that merges two sorted arrays of different lengths. Trains array handling.
8. A program that decomposes a text string with a product code into its textual constituents. Trains handling of text strings.
9. A program with both an iterative and a recursive palindrome predicate for text strings. Trains programming of recursive functions.
10. A program that constructs a playing card and deck of 52 cards in addition to a number of jokers. Trains the use of structs, and array of structs.
11. The exam assignment: Based on game results from 198 games the highest ranking football tournament in Denmark (the Super League), a number of questions should be answered. In addition, the final ranking of the teams should be produced in terms a sorted table. This assignment involves reading input from the text file, representation of the games with use of adequate data structures, and sorting by means of qsort from the C standard libraries.

ABOUT THE AUTHOR

Associate Professor Kurt Nørmark, PhD, Department of Computer Science, Aalborg University, Selma Lagerlöfs Vej 300, DK-9220 Aalborg, DENMARK. Phone: +45 9940 8896, E-mail: normark@cs.aau.dk. Homepage: http://www.cs.aau.dk/~normark/.