WEB ENVIRONMENT TO SUPPORT TEACHING INTRODUCTORY PROGRAMMING

by

Dağhan Dinç

BS, Computer Engineering, Boğaziçi University, 2005

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering
Boğaziçi University
2008

WEB ENVIRONMENT TO SUPPORT TEACHING INTRODUCTORY PROGRAMMING

APPROVED BY:

Dr. Suzan Üsküdarlı (Thesis Supervisor)	.....
Assist. Prof. Pınar Yolum	.....
Assoc. Prof. Hamdi Erkunt	.....

DATE OF APPROVAL: 10.06.2008

ACKNOWLEDGEMENTS

I would like to thank my wife for her patience and understanding. I would like to thank my family for directing me to science. I would like to thank my advisor Dr. Suzan Üskudarlı for guiding me. I would like to thank Doc. Dr. Tuna Tuğcu for encouraging me for this project. I would like to thank you for starting reading, I wish you a joyful reading till the end.

ABSTRACT

WEB ENVIRONMENT TO SUPPORT TEACHING INTRODUCTORY PROGRAMMING

This thesis describes a Web based education support environment for supplementing the teaching of "Introduction to Programming" (code CMPE150) course at Boğaziçi University by the Computer Engineering Department. The course in question has always been a challenging one to teach due to its nature and the size of the student body. This work aims to provide a higher quality learning environment given the constraints of where it is applied. To this end a system was designed, implemented, and used in the teaching of this course. This work reports on design, development and use of this system.

This thesis will describe in detail why and how we built the system as well as our use for two semesters. In Chapter 2 we explain our design motivations. In Chapter 3 present work related to supplemental online teaching environments. In Chapter 4 we describe the system requirements. In Chapter 5 we introduce the design of the system. In Chapter 6 we provide implementation decisions and details. In Chapter 7 we explain the use of this system both from the student as well as staff perspective. In Chapter 8 we present consequences and benefits of the feedback mechanisms that we implemented. In Chapter 9 we present our evaluation regarding the use during two semesters. In Chapter 10 we outline future work. Finally, in Chapter 11 we complete the thesis with concluding remarks.

ÖZET

GİRİŞ DÜZEYİNDE PROGRAMCILIK EĞİTİMİNİ DESTEKLEMEK İÇİN WEB ORTAMI

Bu tez çalışması, Boğaziçi üniversitesi Bilgisayar Mühendisliği bölümünde verilen "Programcılığa Giriş" (kod: cmpe150) dersini destekleme amaçlı olarak geliştirilen "Giriş düzeyinde programcılık eğitimini desteklemek için web ortamı" çalışmasını anlatmaktadır. Bahsedilen ders, pek çok eğitim kurumunda yapısı ve boyutları sebebiyle öğrenciler için oldukça zorlayıcı bir ders olmuştur. Yaptığımız çalışmanın amacı dersin veriliş ortamı göz önünde bulundurularak öğretim kalitesini arttırmaktır. Bu çalışmada öğrenci ve eğitim kadrosunun kullanımına açık şekilde program yazılabilecek ve eğitim materyalleriyle zenginleştirilmiş bir web ortamı sunulmaktadır. Bu raporda tasarım, gerçekleştirim ve sistemin kullanımı anlatılmaktadır.

Bu çalışma detaylı olarak bu sistemi neden ve nasıl geliştirdiğimize değinmektedir. Ayrıca iki dönemdir kullanmakta olduğumuz sistemin tecrübe ve dönütlerine de yer verilmiştir. 2. ünitede sistemi neden tasarladığımıza değindik. 3. ünitede çalışmamızla alakalı alanlarda yapılan çalışmalardan bahsettik. 4. ünitede bu sistemin hangi özellikleri içermesi gerektiğini anlattık. 5. ünitede sistemimizin içermesi gerektiği özellikleri göz önünde bulunarak yaptığımız tasarımı sunduk. 6. ünitede tasarımımızı nasıl somut yazılıma dönüştürdüğümüzü anlattık. 7. ünitede sistemin öğrenci ve eğitim kadrosu gözünden nasıl kullanıldığını ekran görüntüleriyle tarif ettik. 8. ünitede sistemden aldığımız dönütü aktardık. 9. ünitede bu sistemi kullandığımız iki sömestir'i göz önünde bulundurarak sistemimizi değerlendirdik. 10. ünitede sisteme gelecekte yapılabilecek olası eklentileri ve geliştirmeleri belirttik. Son olarak 11. ünitede son görüşlerimizi belirttik.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iii
ABSTRACT iv
ÖZET v
LIST OF FIGURES xi
LIST OF TABLES xv
LIST OF SYMBOLS/ABBREVIATIONS xvii
1. INTRODUCTION 1
2. MOTIVATION 4
2.1. Student Perspective 4
2.1.1. Lack of Computer Background 4
2.1.2. Lack of Personal Computer 4
2.2. Instructor Perspective 5
2.2.1. Software Maintanence in Labs 5
2.2.2. Exam Difficulties 5
2.2.3. Lack of Feedback 6
2.3. Conclusion 7
3. RELATED WORK 8
3.1. Web Based Software 8
3.1.1. Email Clients 8
3.1.2. Multimedia Content 9
3.1.3. Commercial Software 9
3.1.4. Compilation 10
3.2. Web Based Programming Education 12
3.3. Introductary Programming Education 13
4. REQUIREMENTS 16
4.1. Accessibility 16
4.1.1. Ease of Access 16
4.1.2. Ease of Use 16
4.2. Primary Editor Components 16

4.2.1. Syntax Highlighting: . . . 17
4.2.2. Indentation: . . . 17
4.2.3. Automatic Bracket and Quote Closing: . . . 17
4.2.4. Undo-Redo Support: . . . 17
4.2.5. Line Numbering: . . . 18
4.2.6. Error Highlighting: . . . 18
4.2.7. Automatic Code Completion: . . . 18
4.3. Privacy and Security . . . 19
4.4. Computer Based Exams . . . 19
4.5. Course Content . . . 19
4.6. Feedback . . . 19
5. DESIGN . . . 21
5.1. Software Environment . . . 21
5.1.1. Web Programming . . . 21
5.1.2. Operating System of the Servers: Windows . . . 21
5.1.3. Compiler: Visual Studio 2005 C++ Express Edition . . . 22
5.1.4. Development Environment: PHP-MySQL-Apache . . . 22
5.1.5. No Installation - Javascript . . . 22
5.1.6. Database Use . . . 22
5.2. Software Architecture . . . 23
5.3. Tier Details . . . 23
5.3.1. Data Tier 1 . . . 23
5.3.2. Logic Tier 2 . . . 29
5.3.3. Data Tier 2 . . . 30
5.4. Hardware Environment . . . 30
5.4.1. Multiple Servers . . . 30
5.4.2. Private Computer Labs . . . 32
6. IMPLEMENTATION . . . 33
6.1. Rich Editor Support . . . 33
6.1.1. Syntax Highlighting . . . 33
6.1.2. Indentation . . . 35
6.1.3. Automatic Bracket and Quote Closing . . . 35

6.1.4. Undo-Redo . . . 35
6.1.5. Line Numbering . . . 36
6.1.6. Error Highlighting . . . 36
6.1.7. Auto Complete . . . 38
6.2. Privacy and Security . . . 38
6.3. Course Content . . . 38
6.4. Feedback . . . 38
6.5. Authentication . . . 39
6.6. Tiers in Detail . . . 40
6.6.1. Data Tier 1: Database Storage via MySQL . . . 40
6.6.2. Logic Tier 2: Database Interaction via Stored Procedures . . . 40
6.6.3. Data Tier 2: Filesystem storage . . . 41
6.6.4. Data Tier 3: Cookies . . . 41
6.6.5. Logic Tier 1: PHP coding . . . 41
6.6.5.1. PHP Classes . . . 41
6.7. Exam-Specific Attributes . . . 48
6.7.1. Exam Environment . . . 48
6.7.2. Header-Trailer Codes . . . 49
6.7.3. Security . . . 49
6.7.3.1. Exam Password . . . 49
6.7.3.2. Logging . . . 49
6.7.3.3. Exam Environment . . . 49
7. EXPERIENCE . . . 50
7.1. Student Perspective . . . 50
7.1.1. Login . . . 50
7.1.2. Writing Programs . . . 53
7.1.3. Online Course Notes . . . 60
7.1.4. Project Upload . . . 62
7.1.5. Settings - Update Password . . . 62
7.1.6. Feedback . . . 63
7.1.6.1. Report a Bug . . . 63
7.1.6.2. Make a Recommendation . . . 65

7.2. Staff . . . 66
7.2.1. Bug Reports and Recommendations . . . 66
7.2.2. Exam Passwords . . . 66
7.2.3. Error Hints . . . 67
7.3. Admin . . . 70
7.4. Exam Student . . . 71
8. FEEDBACK . . . 74
8.1. Student Recommendations . . . 74
8.1.1. Comment Colors . . . 74
8.1.2. Disappearing \n bug . . . 74
8.1.3. File Rename . . . 74
8.1.4. File Sorting . . . 74
8.1.5. Bracket and parenthesis coloring . . . 75
8.2. Top Compile Errors . . . 75
9. EVALUATION . . . 76
9.1. Comparison With Alternatives . . . 76
9.2. Computer Based Exam vs Paper Exam . . . 76
9.3. Possible Ways to Analyze Usage Data . . . 76
9.4. Survey Results . . . 79
9.5. Portability . . . 80
10. FUTURE WORK . . . 82
10.1. Teaching Staff Based Improvements . . . 82
10.1.1. Exam Reader . . . 82
10.1.2. Editable Course Notes . . . 82
10.1.3. Improved Feedback Reporting . . . 82
10.1.4. Improved Admin Interface . . . 83
10.1.5. Example Question and Quiz Creation Interface . . . 83
10.2. Unimplemented Requests From Students . . . 83
10.2.1. Turkish Character Support . . . 83
10.2.2. Link Error Bug . . . 84
10.2.3. Cross Browser Support . . . 84
10.2.4. Code Folding . . . 84

10.2.5. Offline Online Compiler 84
10.2.6. Mass Deletion of Source Files 85
10.3. Our Plans 85
10.3.1. Improved User Interface and User Experience 85
10.3.2. Social Networks 86
10.3.3. Code Tidy 86
10.3.4. Debugging Support 86
10.3.5. Automatic Evaluation Of Code 86
10.3.6. Stress Test 86
10.3.7. Usability Test 87
10.3.8. Online Compiler For The World 87
11. CONCLUSIONS 88
APPENDIX A: Stored Procedure Sources 89
A.1. bug_report_insert: 89
A.2. error_explanation_toplist: 89
A.3. error_explanation_toplist_by_student_no: 90
A.4. error_hint_insert: 91
A.5. error_log_insert: 92
A.6. file_action_log_insert: 93
A.7. page_visit_log_insert: 94
A.8. part_select: 95
A.9. recommendation_insert: 95
APPENDIX B: Javascript Sources 97
B.1. Automatic Bracket and Quote Closing 97
APPENDIX C: PHP Sources 99
C.1. PHP Compile Error Parsing 99
APPENDIX D: Survey Results 100
D.1. Compiler Use 100
D.2. CompilerFeatures 100
REFERENCES 103

LIST OF FIGURES

Figure 3.1. Screenshot of eyebot page. Eyebot is a web based compilation service for robotics source codes 10

Figure 3.2. Screenshot of jxxx page. Jxxx is a web based compilation service for java source codes including applets. 11

Figure 3.3. Screenshot of djgpp online compilation page page. Djgpp online is a web based compilation service for C programs 11

Figure 3.4. Screenshot of w3schools HTML editing page within HTML tutorial 12

Figure 3.5. Screenshot of w3schools Javascript editing page within Javascript tutorial 13

Figure 3.6. Screenshot of eloquentJavascript interactive javascript tutorial page 14

Figure 3.7. Screenshot of alice, a 3D programming education software 15

Figure 3.8. Screenshot DrScheme program 15

Figure 5.1. 3-tier architecture demonstration 24

Figure 5.2. 3-tier architecture that we used 25

Figure 5.3. Relationship schema of database tables 26

Figure 5.4. Data Tier 2 directory structure. This tier is used for storing passwords, source files, submitted projects etc. 31

Figure 6.1. Javascript regular expression source code used for syntax highlighting 34

Figure 6.2. An example of top compile errors of the week. As you notice, bracket and parenthesis errors are very common 36

Figure 6.3. Example erroneus C source file 37

Figure 6.4. Compile errors listed by gcc compiler 37

Figure 6.5. Compile errors listed by Microsoft Visual Studio 2005 C++ compiler 38

Figure 7.1. Screenshot of login page 51

Figure 7.2. Screenshot of login page with login form filled 51

Figure 7.3. Screenshot of login page, when the student tries to fill the password field with Caps Lock open, he gets a warning 52

Figure 7.4. Screenshot of update password page. The student is redirected here for the first time when loginned with default password 52

Figure 7.5. Screenshot of confirmation page 53

Figure 7.6. Screenshot of home page. 54

Figure 7.7. Screenshot of add new file form, filled at home page. 54

Figure 7.8. Screenshot of home page, after file added via add new file form. . . 55

Figure 7.9. Screenshot of editor. 56

Figure 7.10. Screenshot of editor while autocompleting. 56

Figure 7.11. Screenshot of editor after save button is clicked. 57

Figure 7.12. Screenshot of editor after compile button is clicked. 57

Figure 7.13. Screenshot of editor showing compile errors. 58

Figure 7.14. First exe running confirmation request 59

Figure 7.15. Second exe running confirmation request 59

Figure 7.16. exe running on the client computer. Note that colors of the windows command shell are inverted to improve printed quality 60

Figure 7.17. Screenshot of online course notes 61

Figure 7.18. Screenshot of online course notes, working on a dynamic example . 61

Figure 7.19. Screenshot of project upload part 62

Figure 7.20. Screenshot of settings part 63

Figure 7.21. Screenshot of update password form 64

Figure 7.22. Screenshot of feedback part 64

Figure 7.23. Screenshot of report bug form 65

Figure 7.24. Screenshot of recommendation form 66

Figure 7.25. Screenshot of staff's home page. There exists a staff link on top of the page which is not visible when a student logins 67

Figure 7.26. Screenshot of exam password list at staff page. Note that exam passwords are censored for security. 68

Figure 7.27. Screenshot of exam password finding form, which is filled to find exam password of two students 68

Figure 7.28. Screenshot of exam passwords listed after entered to exam password finding form. Note that exam passwords are censored for security . 69

Figure 7.29. Screenshot of manage error hints form 69

Figure 7.30. Screenshot of admin page 70

Figure 7.31. Screenshot of CreateOrDeleteAccounts form 71

Figure 7.32. Screenshot of login page when exam part is enabled and home part is disabled 72

Figure 7.33. Screenshot of exam page 72

Figure 7.34. Screenshot of an exam question 73

Figure 8.1. Screenshot of bracket and parenthesis coloring in action 75

Figure 9.1. Number of compiles vs. Midterm 1 grade 78

LIST OF TABLES

Table 5.1. Structure of table action 27

Table 5.2. Structure of table bug_report 27

Table 5.3. Structure of table error_hint 27

Table 5.4. Structure of table error_log 28

Table 5.5. Structure of table error_type 28

Table 5.6. Structure of table file_action_log 28

Table 5.7. Structure of table page_visit_log 29

Table 5.8. Structure of table part 29

Table 5.9. Structure of table recommendation 29

Table 9.1. Microsoft Visual Studio for C++ vs Online Compiler 77

Table 9.2. Paper Exam vs Computer Exam 78

Table 9.3. Results of *compiler use* question of survey. For full table, please refer to appendix D.1 79

Table 9.4. Results of *compiler features* question of survey. For complete table, please refer to appendix D.2 80

Table D.1. Results of *compiler use* question of survey 101

Table D.2. Results of *compiler features* question of survey 102

LIST OF SYMBOLS/ABBREVIATIONS

3D	Three Dimensional
cl	Microsoft C/C++ command line compiler
DB	Database
exe	Windows executable file
FSF	Free Software Foundation
GB	Gigabyte
gcc	GNU Compiler Collection
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
JS	Javascript
MVS	Microsoft Visual Studio
P2P	Peer To Peer
PHP	PHP: Hypertext Preprocessor
SP	Stored Procedure
SQL	Structured Query Language
utf-8	Unicode Transformation Format-8
WWW	World Wide Web

1. INTRODUCTION

This thesis describes a Web based education support environment for supplementing the teaching of "Introduction to Programming" (code CMPE150) course at Boğaziçi University by the Computer Engineering Department. The course in question has always been a challenging one to teach due to its nature and the size of the student body. This work aims to provide a higher quality learning environment given the constraints of where it is applied. To this end a system was designed, implemented, and used in the teaching of this course. This work reports on design, development and use of this system.

In order to better understand the context it will be useful for the reader to know the prior approaches to teaching this course as well as the local conditions of the University and the country.

Boğaziçi University is a government university that is located in Istanbul, Turkey. The Computer Engineering Department is part of the Engineering Faculty. The course CMPE 150: Introduction to Programming is a mandatory course for all engineering students as well a few more departments. Each semester approximately 350 students enroll in it. The course is offered by The Computer Engineering Department as a service course to all departments. There are typically 10 sections of approximately 35 students per section. Although we try to group the students according to their majors, each section often ends up as a heterogeneous group of mixed students. To complicate issues further many of the students are repeat students from prior semesters. Unfortunately, the failure ratio is relatively high (often approaching 50%). The offering of this course has varied over the years.

When the enrolled student number was not too large, each course was taught by a single faculty member. When the student body grew beyond the faculty capacity, some sections were joined during lecture hours. Problem sessions and lab work supplemented lectures. Problem sessions were used to give in depth examples of the current topic

that was covered during the lecture. Labs served as hands-on time for students to work out problems with assistance and were mandatory. The labs were held at scheduled times at the University Information Center, which presented several challenges:

- The lab officials were concerned about security issues and maintenance of the computer resources at the hands of first years students.
- The labs serve all University students and were available for limited time. The time restrictions were very significant for many students.

Many students do not own computers. They must access them in the labs and their dormitories. Given the multitude of assignments along with demands for private use, computer time is hard to get. In our case the students not only need to get computer access, but also access to compilers. This fact seriously hinders learning how to program.

While personal possession of computers is low, the youth is well familiar with computer access in terms of email, web browsing, and various messaging programs. The country has plenty of *Internet Cafe*s that provide such access.

With the expansion of the Computer Engineering Department building, 3 computer labs became available for use of teaching under our maintenance. This resolved our problems related to lab maintenance and compiler issues. But, student access time remained restricted, since we do not have the human resources to monitor the labs.

Our aim was to find a way to provide unlimited computer resources to students so that they could be assured of time to practice and learn. We wanted to present them with learning material and hands-on practice ability. Given the wide appeal and familiarity of Web use and its accessibility benefits we envisioned a Web based system to provide support both the students and the staff involved with this course.

The *Introduction to Programming* course uses the C programming language. This choice has been agreed on by the engineering faculty as a result of meeting the needs of

various department. The choice of an appropriate language in the teaching of programming is beyond the scope of this thesis and is treated as a constraint for the solution we must provide.

This thesis will describe in detail why and how we built the system as well as our use for two semesters. In Chapter 2 we explain our design motivations. In Chapter 3 present work related to supplemental online teaching environments. In Chapter 4 we describe the system requirements. In Chapter 5 we introduce the design of the system. In Chapter 6 we provide implementation decisions and details. In Chapter 7 we explain the use of this system both from the student as well as staff perspective. In Chapter 8 we present consequences and benefits of the feedback mechanisms that we implemented. In Chapter 9 we present our evaluation regarding the use during two semesters. In Chapter 10 we outline future work. Finally, in Chapter 11 we complete the thesis with concluding remarks.

2. MOTIVATION

2.1. Student Perspective

2.1.1. Lack of Computer Background

Introduction to programming course is a challenging course for most of the engineering students. Since usually they have no programming background from high school, it is difficult for them to relate to programming concepts.

Luckily, most of the freshmen university students are internet users thanks to internet cafes. Surprisingly, most of them own an email account [1] and thus they are familiar with a web page accessed via a username and a password. Moreover Boğaziçi University students register via an online system thus they definitely have experience using a web page for logging in by a username and a password.

2.1.2. Lack of Personal Computer

Our course requires the use of a C compiler for writing programs. Learning how to program requires a great deal of hands-on practice, which in turn requires access to computing services. Many students do not own computers because of relatively high computer prices in Turkey. Many students use a computer when they come to university. They access to them in labs, dormitories or other public computers. Such computers are generally not equipped with compilers. In fact, most of them do not permit the existence of a compiler due to security concerns. In previous years, our staff went to great efforts to oversee the use of Microsoft Visual Studio C++ [2] in labs. This was very problematic in that in three major aspects:

1. Student access to labs was limited and insufficient
2. Maintenance load on teaching staff was too high
3. Conflicts with university lab staff arose

Some students may have access to a friend's or their parent's computer but this usually turns to be another problematic situation due to long term working time necessity and installation requirement of bulky compiler software.

Additionally, even if the students own a personal computer, they are not proficient computer users. Thus even installing a program from CD is challenging for them.

2.2. Instructor Perspective

2.2.1. Software Maintanence in Labs

We as computer engineering department are lecturing Cmpe150 [3] course with computer aid. Additionally the exams are given on computers allowing students to use a compiler for easily examining and fixing their programming errors. Maintenance of an installed compiler in computer labs is yet another problematic issue as we mentioned above.

2.2.2. Exam Difficulties

Previously exams were given in computer labs via installed compilers on every computer. Students were asked four programming questions. Students were given four source files, namely q1.c, q2.c, q3.c and q4.c for answering the questions via writing the desired program. Sometimes source files were containing previously written code. That code sometimes used as an aid to student. For example the instructor wrote the input reading and output printing parts and asked the student to write just the intermediate algorithmic section. That code also sometimes used as an enforcing part. For example, the instructor gave the main function calling another function for some purpose and the student was asked to write that function. Writing into main was prohibited.

We were facing many problems when we were using labs with installed compilers for exams:

- Distribution of source files were time consuming. They were being distributed via ftp and due to network limitations and problems it was taking 15-30 minutes.
- When the scripts were distributed via ftp, they seemed to student uneditable since the distributer of source codes seem as owner not the student. Students needed to create four new files with given name format (e.g. q1_student.c) and copy the contents of the distributed file and start working. That was another time consuming activity.
- Although the student was not allowed to change the previously given code, he had the chance to alter it by mistake. When grading student answers, those previously given sections were recovered since they were marked via separating comments. However, when the student accidentally alters the comment, it was impossible to parse those sections. Those were handled via objections and was consuming considerable amount of time.
- Since labs are general purpose places used for multiple courses and for student study, they face many configuration alterations. As a result, installed compilers sometimes fail to work on some lab computers. We needed to alter seats of many students at the beginning of exam due to faulty compiler installations.
- When a computer failed during the exam, student's study was lost. We needed to give extra time to the student to recover.
- After the exam finished, the scripts needed to be collected via ftp. Due to network limitations and problems, this operation were consuming 15-30 minutes and the end of the exam. Moreover, sometimes students shut down their computers as a habit thus their answers could not be taken. Then the staff was manually taking those files.
- Due to student population, the cmpe150 exams are given in two sessions. As a result, all above time consuming factors are doubled. The total delay was usually around 1.5 hours and sometimes it was more than 2 hours.

2.2.3. Lack of Feedback

When students work with an installed compiler on their own, their study and progress cannot be monitored by the instructors. As a result, precious information

that can be utilized to improve course content and structure is missed. Also students have no opportunity to see other students' situation.

2.3. Conclusion

For these reasons, we explored alternatives for a better solution. Considering all those factors, a web based compiler for writing C programs for students seemed appropriate. Thus students have access to any computer connected to the internet, will have an access to a compiler for writing programs. Moreover, independent of which computer they are accessing (it can be a lab computer, a friend's computer or their personal computer, whatever available), they will access to their files. Technical issues which we will discuss later, currently brought Windows [4] and Internet Explorer [5] dependency which we are planning to solve later.

3. RELATED WORK

3.1. Web Based Software

Initially the World Wide Web [6],[7] was thought as a publishing medium where web page owners put content and web page visitors see content. However, with increased accessibility, web based applications [8] became popular:

3.1.1. Email Clients

Initially most of the people were using mail clients such as telnet or outlook for reading emails. Telnet like text based clients are difficult to use compared to graphical interface and can not publish rich text content. Outlook like email clients with GUI are easier to use and can show rich text content. However they usually do not have a built in security for viruses or other kinds of malware. As a result, mails containing viruses deal massive amounts of damage. For example, the ILOVEYOU [9] virus which is spread via email infected 10% of computers in year 2000 and gave 5.5 billion dollars of damage.

Currently there are many web-based email providers such as Yahoo [10], Hotmail [11], and Gmail [12]. Even universities provide web based email interface [13]. Almost all web based email providers present a simple to use graphical interface compared to text based clients such as telnet. Moreover, most of them have a built in virus scanner which helps preventing the spread of email based viruses. Additionally, most of them present massive amounts of email storage. For example Google gives 6.6 GB and Yahoo gives limitless storage. Thus they save the clients' disk storage.

One of the problems of web based email providers were they were not appropriate for companies which desire to send their emails from their domains. However, today Gmail has a free service for companies enabling email addresses from companies' domains up to 100 all with 6.6 GB storage.

3.1.2. Multimedia Content

Previously, multimedia content such as music and video were only stored and played offline. Only P2P sharing was available and very few websites were containing content for download. Thanks to increased Internet bandwidth and advanced data compression techniques, music and video content is moved to the Internet.

There exists online radio and TV channels for almost every commercial channel [14], [15]. There also exists many video sharing websites such as Youtube [16] and Metacafe [17].

Initially internet browsers were only capable of publishing static content. Web based email clients were also relying on refreshing pages for dynamicity. Today, with enriched Javascript power and with the aid of browser plug-ins most popularly flash, almost every desktop based rich content is doable on the web. This enabled multimedia content to be published on the web.

Todays most common web limitation is for 3D graphics where performance is a big issue. That limitation currently limits mostly computer games containing 3D graphics. However even for that there exists many studies which seems promising [18], [19].

3.1.3. Commercial Software

There currently exists web based softwares for commercial business segment. A good example is a web based CRM software developed by SalesForce [20]. Thanks to visualization enhancements of the web we mentioned above, even a commercial software with graphical tools can be implemented on the web.

For business applications, accessibility is crucial. If employees are mobile which is common for especially salesmen and representatives, they might not access the company's up to date data. Alternatively expensive gadgets and software can be used for

accessibility thus improving consistency and coherency.

On the other hand a web based application is accessible via very common and cheap gadgets including simple mobile phones. This gains more importance if the employees are traveling abroad. Enabling employee access via WWW improves business connectivity and continuity.

3.1.4. Compilation

There currently exists compilation services as web pages. For example eyebot [21] is a compilation service for robotics code. The user needs to upload the source file for compilation (see figure 3.1). Similarly jxxx [22] is a compilation service for java source codes including applets. Again, jxxx requires source file upload for compilation (see figure 3.2). It has also support for viewing compiled applets online.

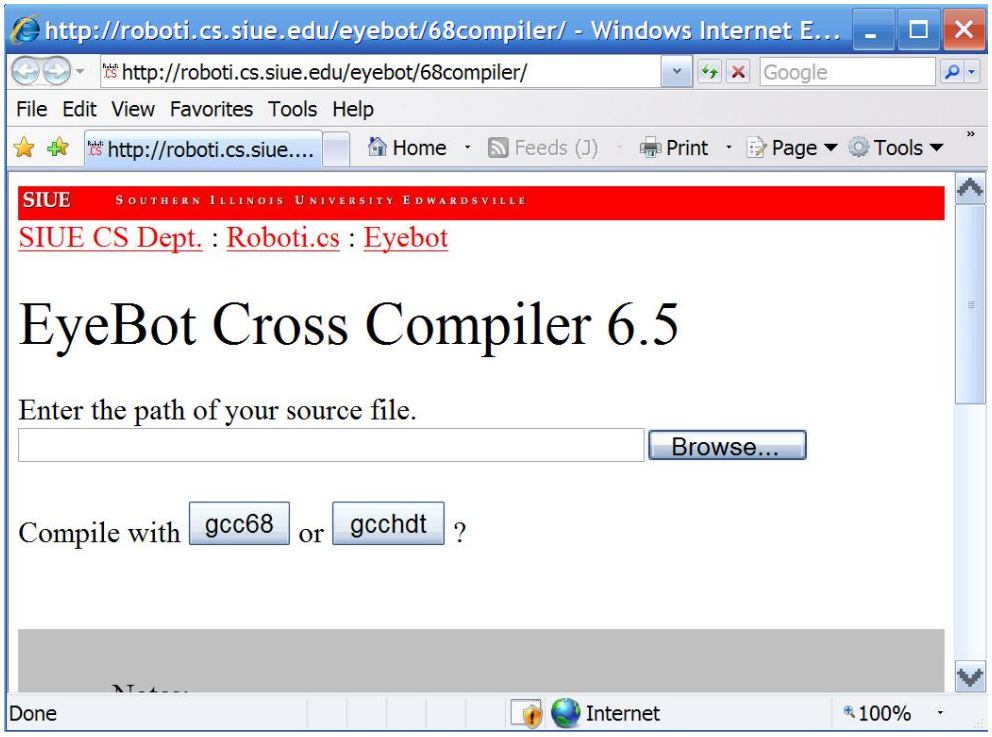


Figure 3.1. Screenshot of eyebot page. Eyebot is a web based compilation service for robotics source codes

Another compilation service is djgpp's online compilation service [23] which contains a textarea for writing C code 3.3.

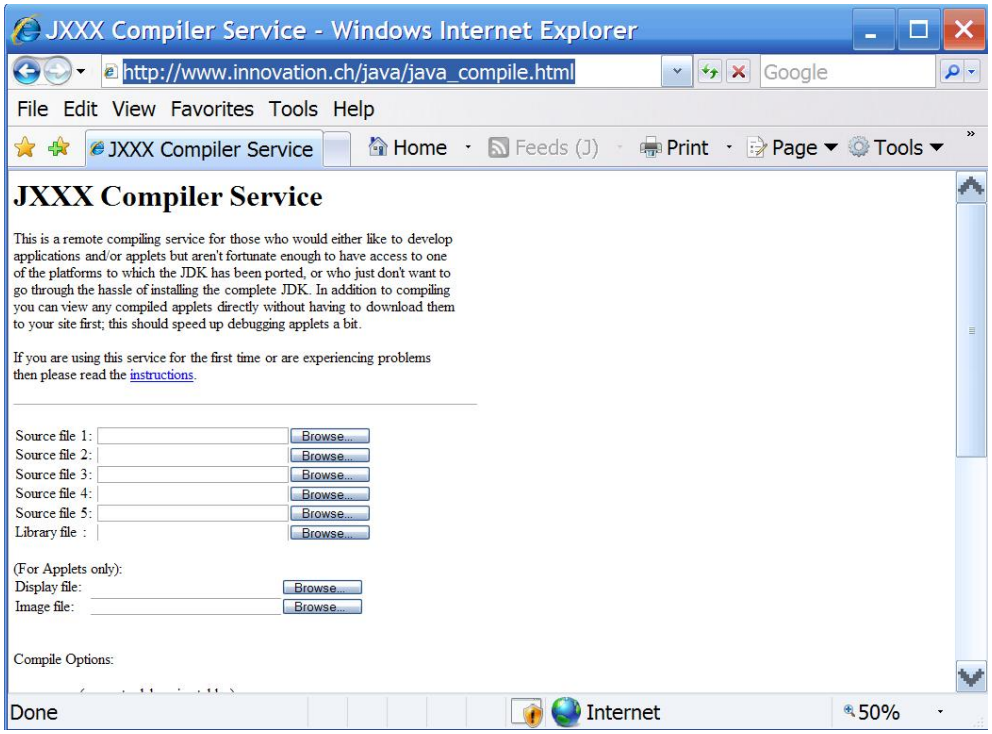


Figure 3.2. Screenshot of jxxx page. Jxxx is a web based compilation service for java source codes including applets.

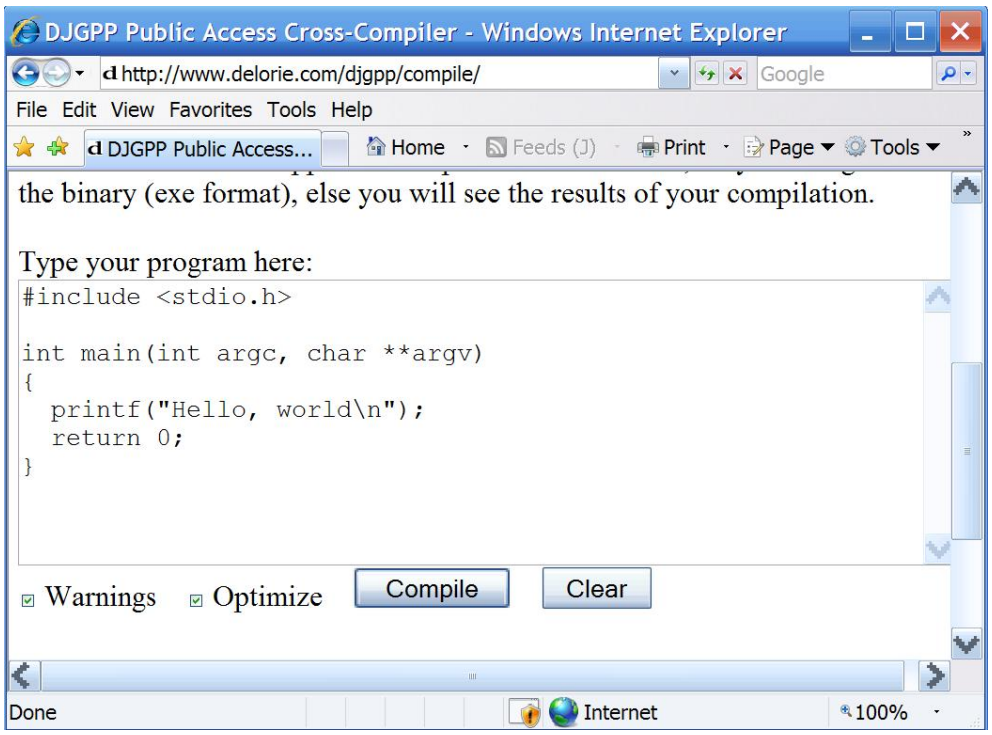


Figure 3.3. Screenshot of djgpp online compilation page page. Djgpp online is a web based compilation service for C programs

Above examples are good for seeing the Internet's potential for software development. However, eyebot and jxxx requires upload of source codes and in order to upload source codes, the user needs another program for creating and editing those source codes. Although djgpp online have a text area for input, plain textarea is inadequate for coding. Plain textarea does not have properties like syntax highlighting or indentation which are crucial while writing programs. Thus those examples are not adequate for being a software development environment.

3.2. Web Based Programming Education

There exists good HTML [24] and Javascript [25] tutorials at `http://www.w3schools.com` with textarea editing and executing. They are very good and compact tutorials which are very appropriate for starting a new language. However, their editing capability is just adequate for learning a single function, not software development because they are also plain textarea based (see figure 3.4 for HTML editing and figure 3.5 for Javascript editing).

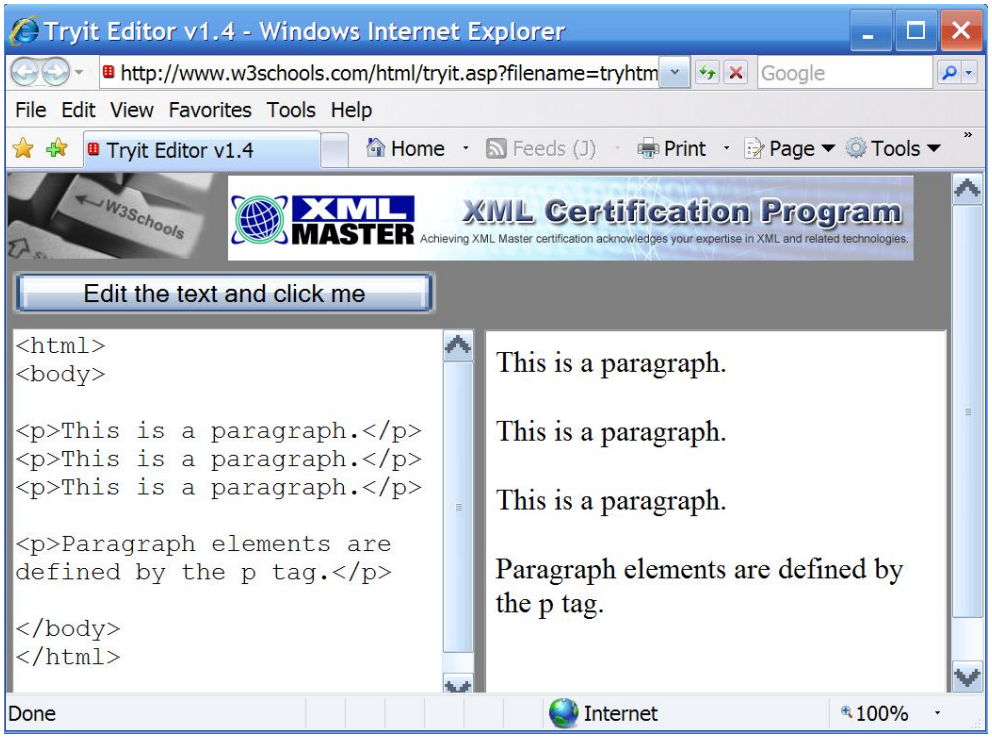


Figure 3.4. Screenshot of w3schools HTML editing page within HTML tutorial

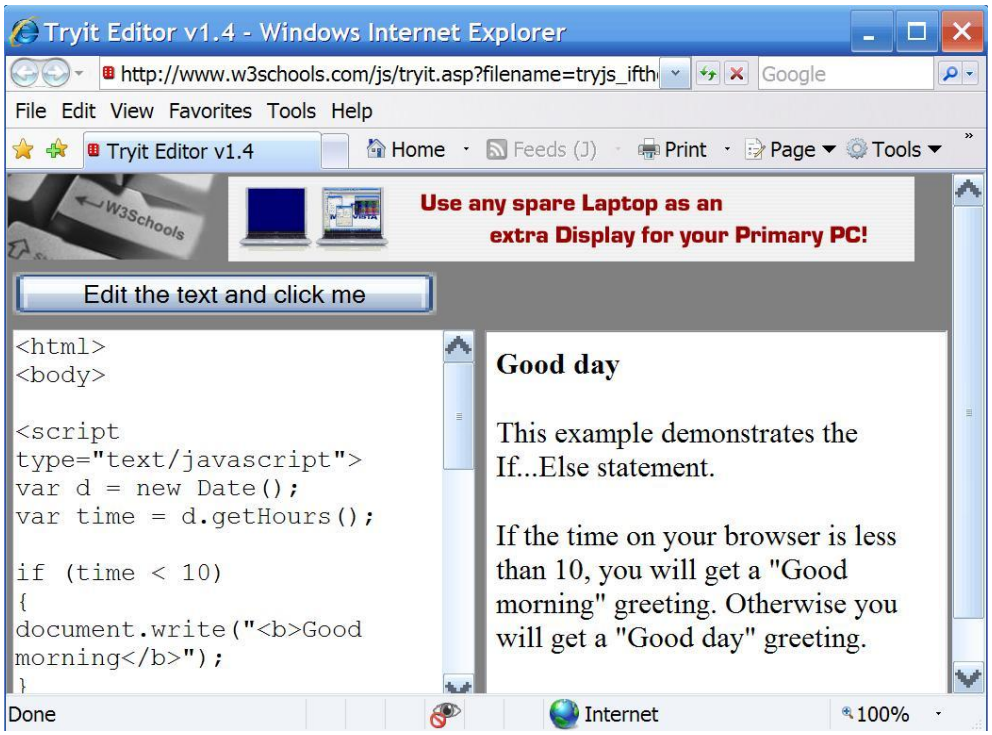


Figure 3.5. Screenshot of w3schools Javascript editing page within Javascript tutorial

There currently exist a pretty successful syntax highlighter [26] which is also cross-browser compatible. It is used within an interactive javascript tutorial [27]. In figure 3.6 you can see the javascript editor in action with syntax highlighting support. However, since we started this project two years earlier and we have more complicated needs, we cannot use that.

3.3. Introductary Programming Education

Teaching computer programming is a challenging task. Many languages and teaching environments are proposed throught the last few decades [28]. In the past, one approach was creating a simplified subset of an industrial language as an educational language. For example, Pascal [29] language was created as a subset of ALGOL 60 [30].

Another approach is creating graphical tools for programming education. For example, Alice [31] is a 3D toolbox where the user can write successive events. It

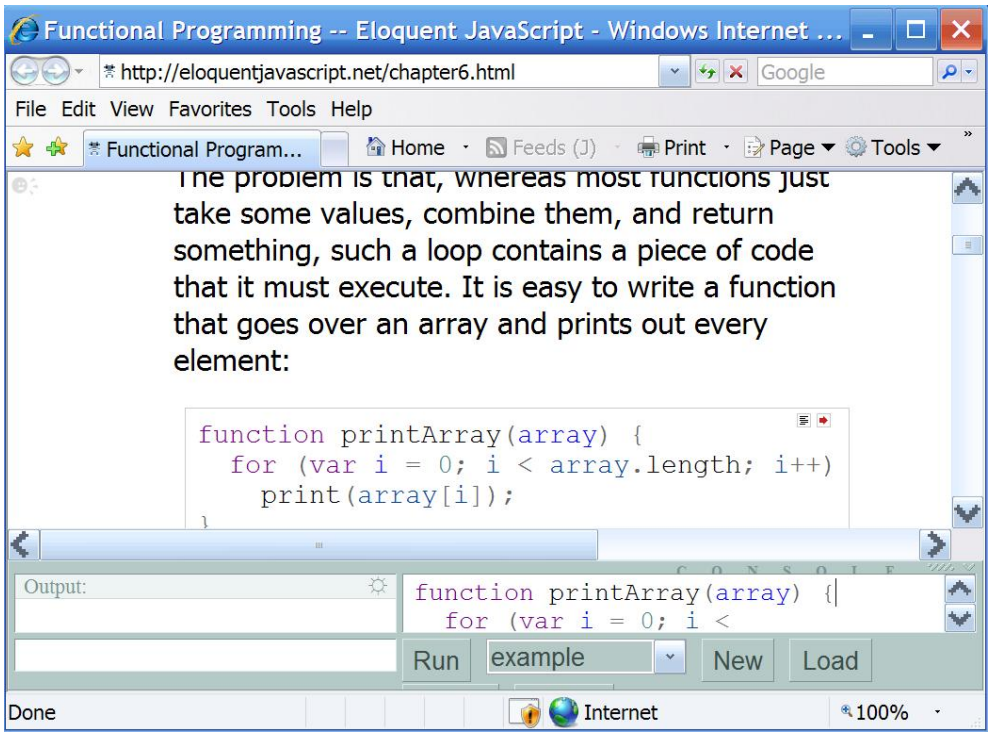


Figure 3.6. Screenshot of eloquentJavascript interactive javascript tutorial page

has support for conditional statements and looping. 3D interface is very motivating especially for middle and high school students (see figure 3.7).

Today, one very popular approach is utilizing a very simple programming language in syntax and focusing on programming and software engineering concepts rather than teaching language syntax. "How To Design Programs" [32] book is a good example of this approach. It uses Scheme [33] language and a modified version of DrScheme [34] program as programming environment (see figure 3.8). It enforces students to write comments and test cases for the program which is usually skipped while trying to teach Java or C syntax.

Although most of the Teaching Institutions prefer a simple and non-industrial programming language for teaching introductory programming [28] [32], due to the multi-disciplinary nature and mass-course requirements of Boğaziçi University, C language [35] is taught at "Introduction to Programming" course [36] to all freshmen engineering students.

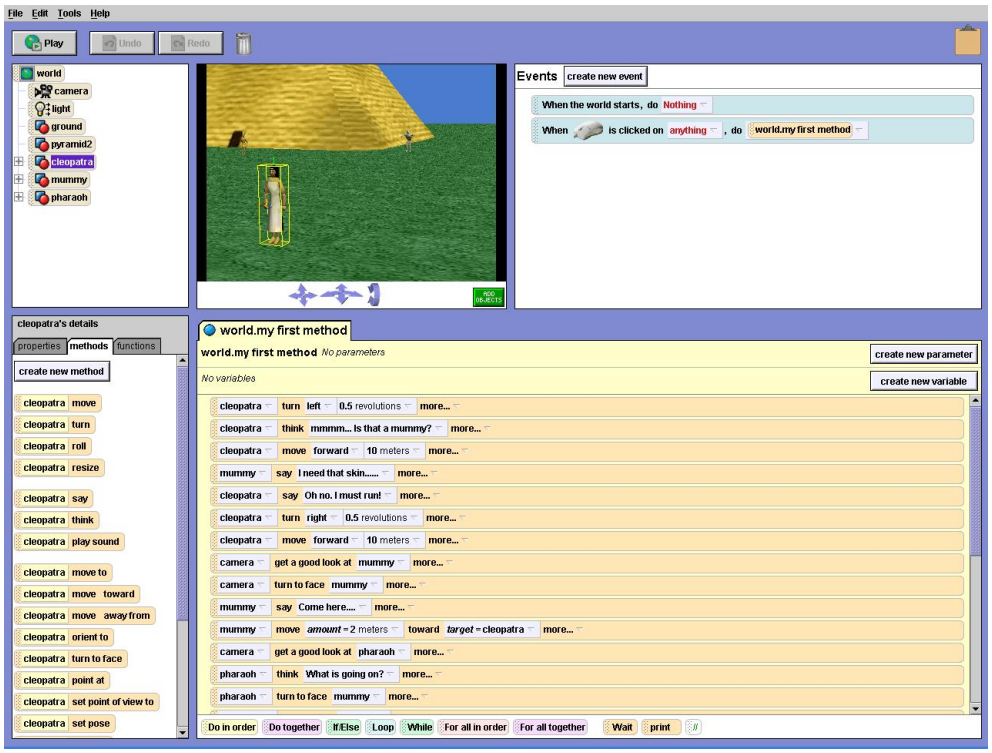


Figure 3.7. Screenshot of alice, a 3D programming education software

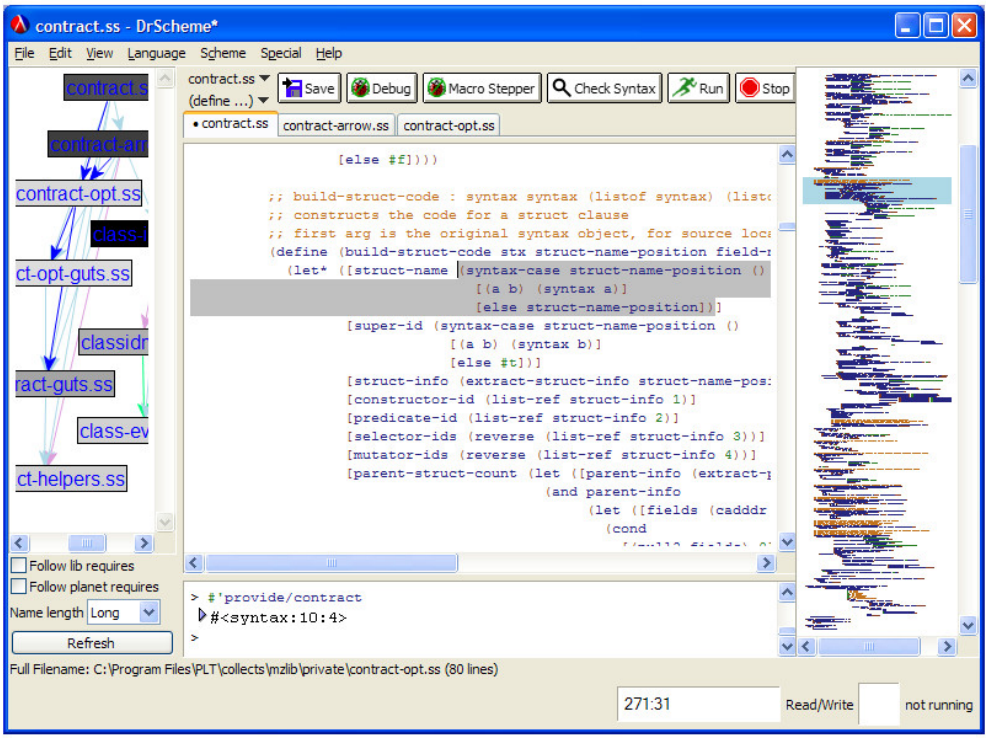


Figure 3.8. Screenshot DrScheme program

4. REQUIREMENTS

In this section, we outline the system requirements for an online teaching supplement for "Introduction to Programming".

4.1. Accessibility

4.1.1. Ease of Access

As we mentioned in 2.1, accessibility of individual computer with installed compilers is weak. The compiler should be accessible from as many places as possible including labs in University, personal computers, internet cafes etc. For our case, the compiler and all the course resources should be accessible by a web browser.

4.1.2. Ease of Use

Many students who met computer before university are only Internet users. Thus they are not experienced computer users in order to work with compilers with complicated interfaces such as Microsoft Visual Studio, Borland etc. Therefore, simpler interfaces similar to web-based email clients are thought to be more suitable.

4.2. Primary Editor Components

Earlier programming editors did not have features like syntax highlighting [37], indentation [38] or mouse support, but only provided a bare environment for program writing. Now we have even freeware editors with syntax highlighting and indentation support features, which are common programming editor components. We listed some of the common programming editor components as requirements below.

4.2.1. Syntax Highlighting:

Syntax highlighting is a feature of some text editors that displays text especially source code in different colors and fonts according to the category of terms [37]. Almost all programming editors provide syntax highlighting. It both improves readability and prevents many syntax errors. Thus syntax highlighting is desirable. Note that, text highlighting means altering the background of the text. In syntax highlighting, usually the color of the font is changed but the background may remain the same.

4.2.2. Indentation:

In computer programming, an indent style is a convention governing the indentation of blocks of code to convey the program's structure [38]. Almost all programming editors provide automatic indentation support which is crucial for following code hierarchy. Thus indentation should be implemented.

4.2.3. Automatic Bracket and Quote Closing:

Modern editors provide automatic closing of brackets ({,(,[) and quotation marks (",') in order to prevent the coder from forgetting them to close. This both prevents syntax errors and eases writing. Automatic bracket and quote closing should be implemented.

4.2.4. Undo-Redo Support:

Undo is a command in most creative programs that erases the last change done to the document reverting it to an older state [39]. The opposite of undo is redo, which in many PC programs corresponds to the Ctrl-Y keyboard shortcut. The redo command reverses the undo or advances the buffer to a more current state [39]. In order to be able to rollback the code, undo-redo ability should be implemented.

4.2.5. Line Numbering:

In computing, a line number is a method used to specify a particular sequence of characters in a text file. The most common method of assigning numbers to lines is to assign every line a unique number, starting at 1 for the first line, and incrementing by 1 for each successive line [40]. This is a simple but important property especially for tracing syntax errors. Numbering code lines should be implemented.

4.2.6. Error Highlighting:

Compiler errors can be followed from reports. However, marking erroneous lines helps the programmer to fix the errors more quickly and with more ease. Error highlighting should be implemented.

4.2.7. Automatic Code Completion:

Autocomplete is a feature provided by many source code text editors, word processors, and web browsers. Autocomplete involves the program predicting a word or phrase that the user wants to type in without the user actually typing it in completely [41].

For today's complicated and long programs, meaningful variable and function naming is crucial. Consequently, shortened variable names are not preferred. Long variable names consisting of one or more concatenated words via underscore (_) or camel case [42] improves readability a lot. However, it hardens writing and easies typo mistakes. As a solution, auto-completion of common words, variable and function names helps a lot while coding. Thus automatic code completion should be implemented.

4.3. Privacy and Security

In our programming course, we give take-home assignments. A working environment which enables students to keep their work secure and private should be implemented.

4.4. Computer Based Exams

Our exams are computer based requiring a proper environment for exam taking. Realizing computer based exams is risky since results of failures can cause delay or loss of data. In addition to every-day use, for safety of the exams, robustness and stability are crucial elements. The system should be robust and stable enough for both daily work and giving exams.

Cmpe150 course is given to more than 300 students every semester. This means that more than 300 students will enter the exam. Exams can be given in two sessions. This means that the system must be able to handle more than 150 students at the same time when giving an exam.

4.5. Course Content

Since our system will be used for educational purposes, it should have more than a mere documentation. Course notes and exercises should be added.

4.6. Feedback

We can classify feedback mechanisms of the system into two classes:

1. **Active feedback mechanisms:** There can be forms where students or staff can enter their opinions, feature-requests or complaints about the system.
2. **Passive feedback mechanisms:** Student behaviors and actions can be logged and investigated for evaluating the system.

In order to validate our system, both feedback mechanisms should be implemented.

5. DESIGN

5.1. Software Environment

5.1.1. Web Programming

As we mentioned in 4.1, ease of access and ease of use are crucial. One of the strongest benefits of web applications' [8] is their accessibility. Thus we chosed to develop a web application to facilitate access to our compiler. The strength of using a web application strategy lies in the fact that students who are already familiar with web use (see 2.1.1) will experience minimal difficulty working with this interface (see 4.1.2).

5.1.2. Operating System of the Servers: Windows

As we mentioned in 4.1.1, one of our most prior requirements is accessibility. Since most of the computers are Windows based, and it is almost standard for internet cafe's and University lab's, we focused on Windows-based client computers. In order to produce Windows executable, we used Windows operating system on the server.

For accessibility of web applications, cross-browser compatibility is an important issue. Sadly, in order to satisfy requirements for "primary editor components" (see 4.2 and 6.1) in a limited time, this property was overlooked. Our compiler requires Microsoft Windows Operating System [4] and Internet Explorer 6.0 [5] or higher as a browser on the client side. However, since our University computers as well as most public computers supply Microsoft Windows and Microsoft Internet Explorer, we have decided this support to be sufficient. Moreover, most of other common operating systems such as Linux or BSD distributions come with a C compiler. Nevertheless, our system is not only a remedy for the lack of built-in compiler software in Windows operating system but also a software both supplying students a familiar web interface for programming and enabling staff to get feedback.

5.1.3. Compiler: Visual Studio 2005 C++ Express Edition

Since Microsoft distributes this compiler for free, this is easily accessible for students who are Windows users if they want to use an off-line compiler. For compatibility, on the server, we decided to use the command line compiler of the Microsoft Visual Studio 2005 C++ Express Edition [43].

5.1.4. Development Environment: PHP-MySQL-Apache

We preferred PHP[44]-MySQL[45]-Apache[46] trio for development since it is open-source, widely used and have enormous community for forums, blogs and support. It also have the possibility to run on multiple platforms which might be a necessity in the future. We chosed wampserver [47] since it automatically installs, configures and integrates PHP, MySQL and Apache.

5.1.5. No Installation - Javascript

Since we aim to make our system to work at most computers as possible, we avoided any client side installations even Flash [48] or Java Applets [49]. As a result, we used javascript for client-side dynamic properties.

5.1.6. Database Use

For logging purposes, we use a MySQL [45] database. However primary use without any data logging does not require database. Thus for critical cases, we can close the database access completely and compiler continues to operate. During exams, for robustness (see 4.4), we stop logging. Although exam log is a precious data, 360 freshmen engineering students' non-problematic C exam is more valuable.

5.2. Software Architecture

As a common approach for web based software, we used 3-tier architecture [50] (see figure 5.1 taken from http://en.wikipedia.org/wiki/Three-tier_(computing)). 3-tier architecture is a design choice which is based on splitting the software into 3-tiers:

1. **Data tier:** Used for storing and retrieving data. Usually a relational database is used such as MySQL, MsSQL, Oracle etc.
2. **Logic tier:** This layer coordinates the application processes and commands. This layer is usually implemented with a programming language such as PHP, ASP, C# etc.
3. **Presentation tier:** This tier is the user interface. For web applications, it is usually implemented via HTML, CSS and Javascript in combination.

The reason for chosing 3-tier architecture is, it enables to write system parts in separate and specified programming languages and environments. For example, data storage is implemented via a relational database and SQL commands, interface is implemented via HTML and logic is implemented via PHP. As a result, code readability and maintanence is improved. For example when the programmer sees an HTML code, he knows that code is for interface and not for database or logic. Moreover, alteration of a tier technology usually does not effect other layers.

However, since for business layer we used both PHP and stored procedures of MySQL, we do not have chance to benefit the easy-platform-change feature of 3-tier for data tier (see figure 5.2).

5.3. Tier Details

5.3.1. Data Tier 1

This tier is implemented via MySQL. It consists of nine tables. In figure 5.3 you can see the relationships between tables.

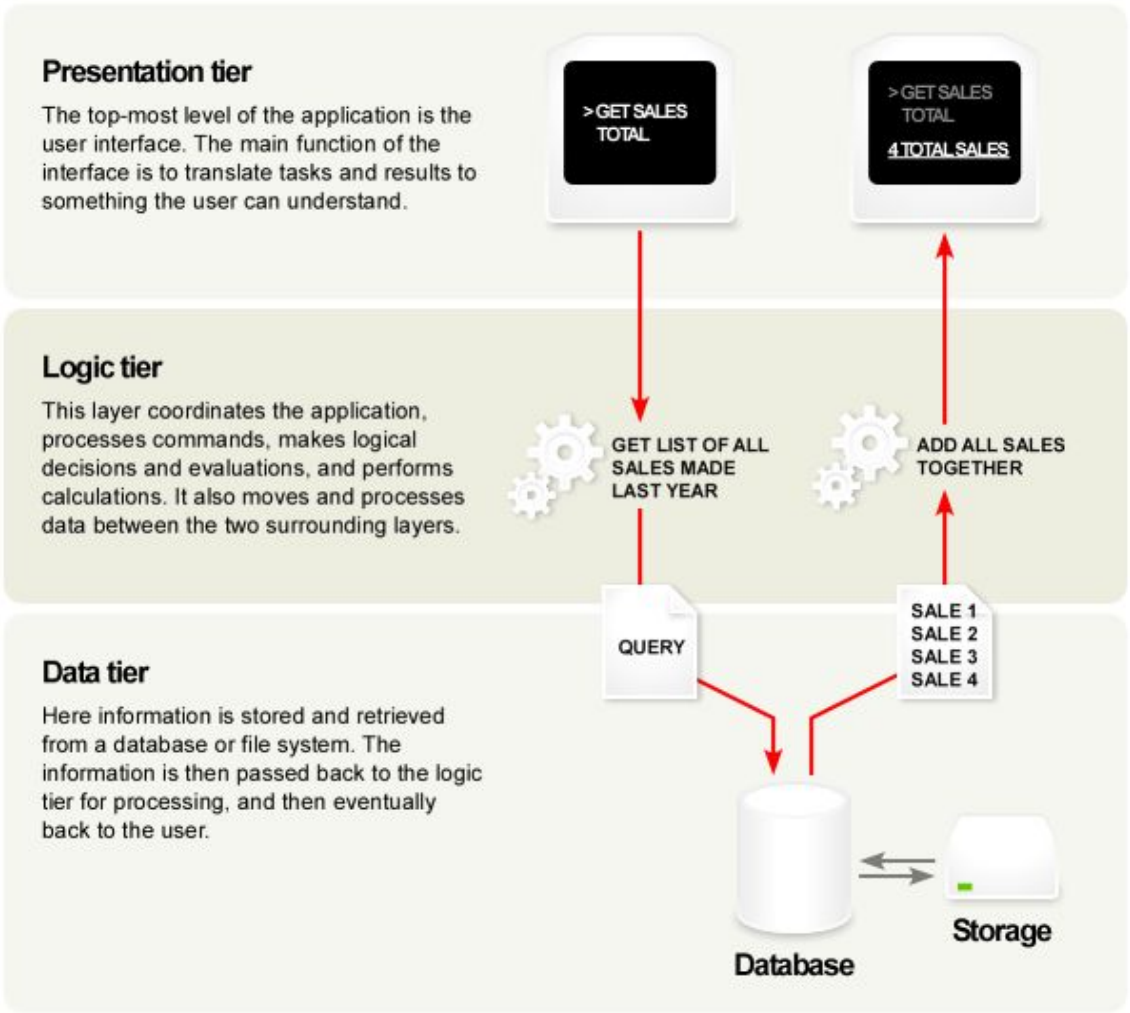


Figure 5.1. 3-tier architecture demonstration

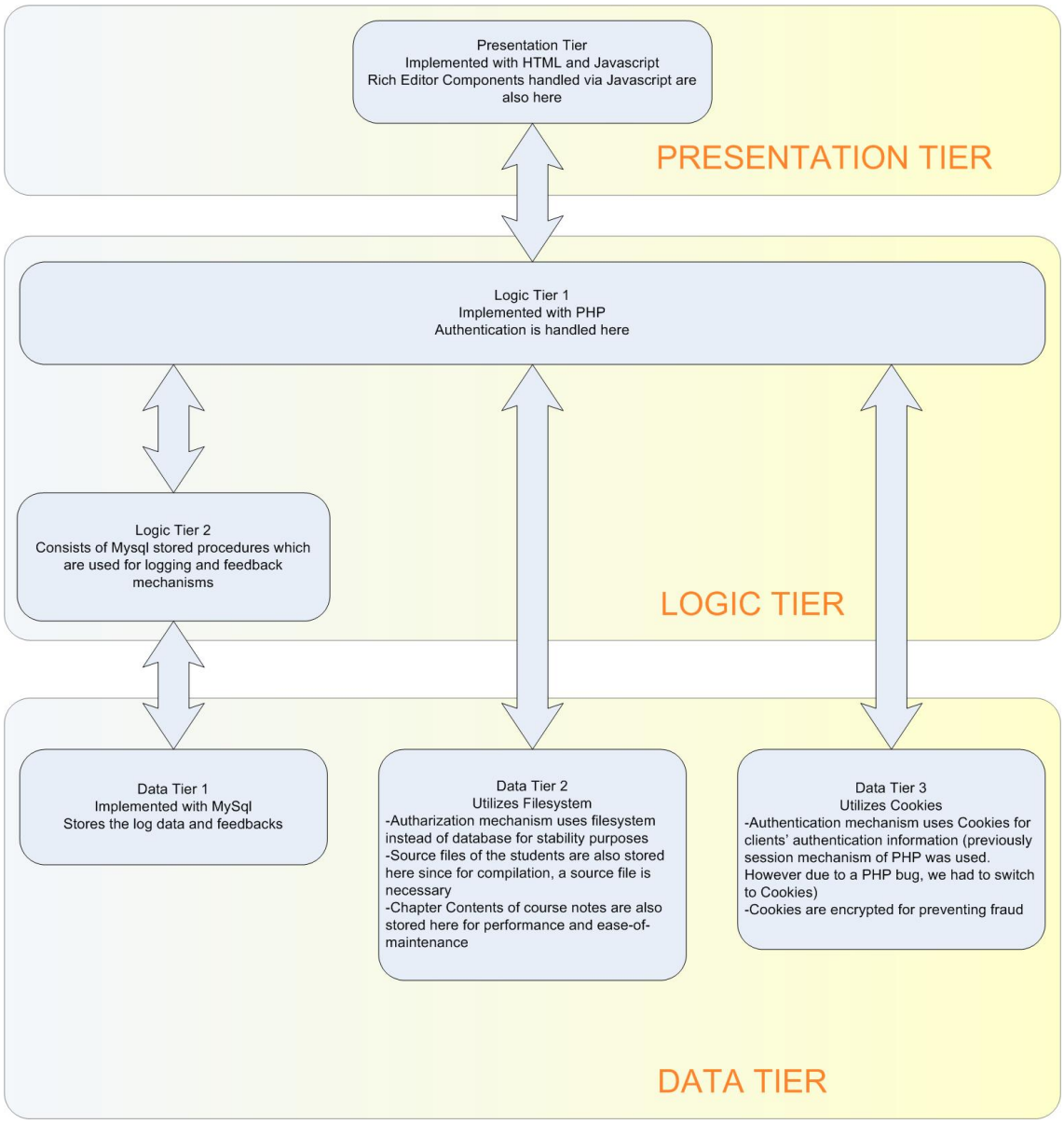


Figure 5.2. 3-tier architecture that we used

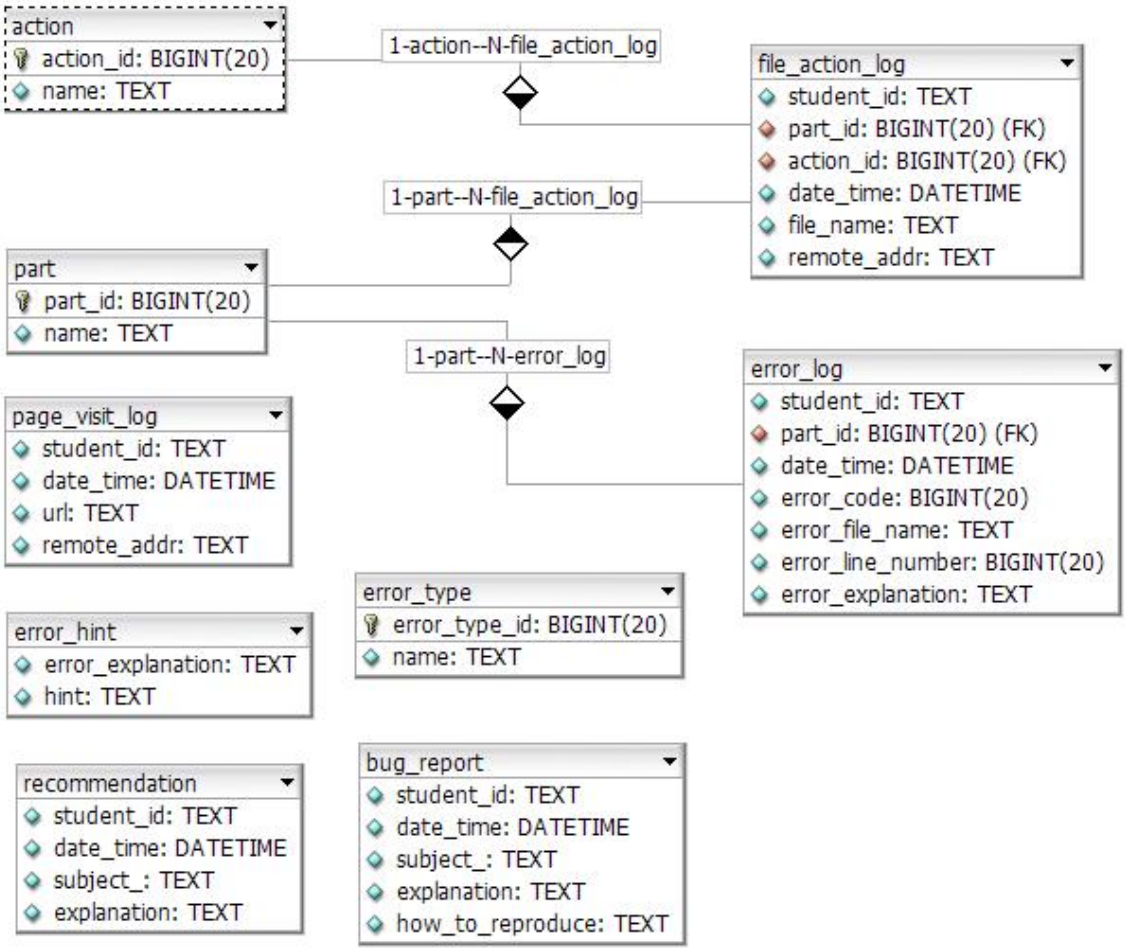


Figure 5.3. Relationship schema of database tables

Table 5.1. Structure of table action

Field	**Type**	**Null**	**Default**
action_id	bigint(20)	Yes	NULL
name	text	Yes	

Table 5.2. Structure of table bug_report

Field	**Type**	**Null**	**Default**
student_id	text	Yes	
date_time	datetime	Yes	
subject_	text	Yes	
explanation	text	Yes	
how_to_reproduce	text	Yes	

Tables from 5.1 to 5.9 shows detailed structure of the tables. Bold fields are primary keys.

- **action:** This table contains the types of possible actions for movement logging, namely save, compile, run and view.
- **bug_report:** This table is used within feedback mechanism. Bug reports coming form students and staff are stored here and published within staff section of online compiler as a reference.
- **error_hint:** This table stores the error hints entered by instructors for top errors published within online compiler (see figure 6.2).
- **error_log:** This table stores the log of the compile errors that students make. These are used for feedback. Most relevant use is publishing of top compile errors of the week list. Although top compile errors of the week is just a feedback about recent information, error logs are being stored for more than one year and continuing.

Table 5.3. Structure of table error_hint

Field	**Type**	**Null**	**Default**
error_explanation	text	Yes	
hint	text	Yes	

Table 5.4. Structure of table error_log

Field	Type	Null	Default
student_id	text	Yes	
part_id	bigint(20)	Yes	
date_time	datetime	Yes	
error_code	bigint(20)	Yes	
error_file_name	text	Yes	
error_line_number	bigint(20)	Yes	
error_explanation	text	Yes	

Table 5.5. Structure of table error_type

Field	Type	Null	Default
error_type_id	bigint(20)	Yes	NULL
name	text	Yes	

- **error_type:** This table stores the types of errors, namely warning and error. Currently warning logging is not used but this table is kept as a reference for the future.
- **file_action_log:** Student actions on source files such as save, compile, run and view are logged into this table. Currently run actions are not logged since they are directly sent to exe files instead of a php page. Currently, there does not exist any interface for reporting but file actions are being logged for more than one year and continuing.
- **page_visit_log:** Student movements at portions of the system such as online

Table 5.6. Structure of table file_action_log

Field	Type	Null	Default
student_id	text	Yes	
part_id	bigint(20)	Yes	
action_id	bigint(20)	Yes	
date_time	datetime	Yes	
file_name	text	Yes	
remote_addr	text	Yes	

Table 5.7. Structure of table page_visit_log

Field	Type	Null	Default
student_id	text	Yes	
date_time	datetime	Yes	
url	text	Yes	
remote_addr	text	Yes	

Table 5.8. Structure of table part

Field	Type	Null	Default
part_id	bigint(20)	Yes	NULL
name	text	Yes	

course notes are logged into this table.

- **part:** Stores the names of the parts: home, exam and course_notes.
- **recommendation:** This table is used within feedback mechanism. Recommendations and feature requests coming form students and staff are stored here and published within staff section of online compiler as a reference.

5.3.2. Logic Tier 2

Consists of MySql stored procedures. All connection from PHP scripts to database is via stored procedures. Implementation details will be explained in Implementation section. (see 6.6.2).

Table 5.9. Structure of table recommendation

Field	Type	Null	Default
student_id	text	Yes	
date_time	datetime	Yes	
subject_	text	Yes	
explanation	text	Yes	

5.3.3. Data Tier 2

Stored within file system. It is made up nested directories and files. The directory hierarchy can be seen in figure 5.4

- **passwords** folder: Stores authentication information. Under this folder, there exists directories named with student numbers or staff names which are users of the system. Under every user directory, there exists a password file which stores the hashed output of password. Thus login authentication except exam part are validated via here.
- **home** folder: Stores students' source files. After compilation, resultant obj and exe files are also stored here. Every student's c source files are stored under a directory whose name is student's number.
- **admin** folder: If there is a directory named "foo" under this folder, the account with username "foo" gets admin priviledges. Admin rights are given to instructors and one research assistant who is the online compiler coordinator every semester.
- **course_notes** folder: Under *course notes* part of the system, there exists C code examples which students can run and alter. If a student works on an example here, the resultant altered source file and exe files are stored under this folder.
- **project** folder: Stores students' submitted projects.
- **exam** folder: During exams, students' answers which are C source files are stored under this directory.
- **editableCourseBook** folder: This is not currently used. It is planned that this will contain a course book which instructors can edit via a web interface.

5.4. Hardware Environment

5.4.1. Multiple Servers

We currently use three servers for this application. One server is the public server which is used for students for studying. Two servers are used for realizing exams. We do not use the public server for exams since it is open to the Internet and thus open to

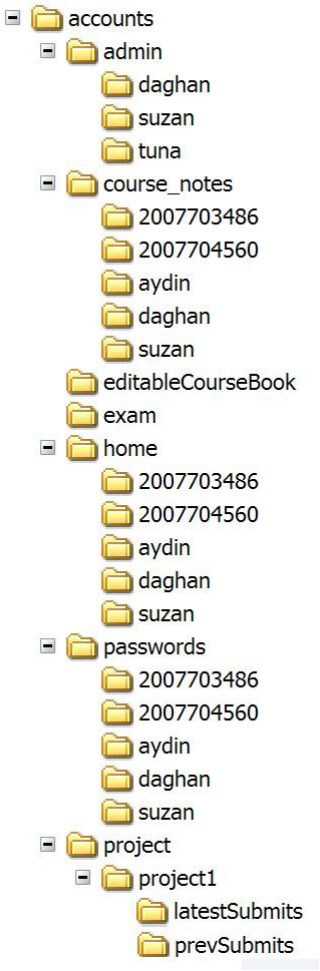


Figure 5.4. Data Tier 2 directory structure. This tier is used for storing passwords, source files, submitted projects etc.

attacks. We use two servers during the exam because compilation is a resource hungry operation and 180 students simultaneously having exam on one server sometimes results in system hangs. Moreover if one server fails, the second server stands as a backup server. We newly learned that because of a PHP bug, when session is used, using exec command results in system hangs. We newly abandoned session based authentication and now servers seem to be much more robust.

5.4.2. Private Computer Labs

Previously we were realizing exams on University's other labs. That was pretty problematic and insecure because of the distributed location of the labs and the installed software management of the labs was not specifically configured for C education but for general lab use as expected. After the online compiler, although we were still realizing exams on University's other labs, exams became much less problematic since there was no need for software maintanence. Currently we have three computer labs at Computer Engineering building with total of around 200 computers where we lecture two mass computer-based courses, namely "Cmpe150 introduction to programming" [3] and "Eng101 introduction to computers" [51] and realize our computer-based exams. There also is still chance of accessing the public server for studying from outside.

6. IMPLEMENTATION

6.1. Rich Editor Support

Within HTML, there exists a textarea [52] component. Although it is suitable for ordinary text entry for forms, it is far from adequate for coding. We utilized contentEditable implementation of Microsoft Internet Explorer in order to achieve the following rich text editing features via HTML [53], CSS [54] and Javascript [55].

However, plain contentEditable implementation and its built-in functions are not adequate. For that reason, we needed use 2169 lines of Javascript and CSS code where 1651 lines are purely written by us.

6.1.1. Syntax Highlighting

There are many ways of achieving syntax highlighting via Javascript. A very popular one published in Google Code [56] by Alex Gorbatchev [57] uses multiple regular expressions [58] ordered by priority and executed via loops. Although effective and portable, this technique is very slow. It is most commonly used in forums and web pages for publishing code. It is used in one Javascript based code editor [59]. It basically publishes the code. For editing, when a line is clicked, a single line editor with no coloring is created for altering that line. Although it works, it is too complicated and thus buggy. In eloquentJavascript tutorial [27] a top-down parser implementation is used. It is again a little slow and lags while coding.

In order to achieve syntax highlighting for C language with high performance, we used a pretty complicated (long) regular expression as a very simple LL(1) [60] parser without recursion. It traverses the code from left to right once and adds appropriate HTML tags for coloring (see figure 6.1).

This regular expression consists of multiple statements concatenated with OR

```
Match:
/(\b(if|else|for|while|do|goto|switch|case|default|extern|static|const|
volatile|sizeof|entry|typedef|struct|union|return|break|continue|enum|
unsigned|signed|auto|void|long|short|int|float|double|char|register|
printf|scanf)\b)|(\#\D.*)|(\/\/.*)|('(\\.|[^'\\])')|(\/\*[\s\S]*?\*\/)|
("(\\.|[^"\n\r\\])*?")|(\b\d+(\.\d+)?[a-zA-Z_]+)/g
Replace:
'<SPAN class="reserved">$1</SPAN>'+
'<SPAN class="preproc">$3</SPAN>'+
'<SPAN class="comment">$4</SPAN>'+
'<SPAN class="character">$5</SPAN>'+
'<SPAN class="comment">$7</SPAN>'+
'<SPAN class="string">$8</SPAN>'+
'<SPAN class="error">$10</SPAN>'
```

Figure 6.1. Javascript regular expression source code used for syntax highlighting

(—) operators. All those statements match a token which is of type reserved word, preprocessor, comment etc. When it finds one token, it puts that token inside appropriate span tags with given class to make them highlighted (font color is set to some color other than black). Then it continues searching where it left due to the "g" flag at the end.

In a compiled language like C or C++, this regular expression may work slower than an efficient top-down parser implementation even recursion is used. However in interpreted languages such as Javascript, decreasing number of statements or function calls may improve the performance dramatically. Probably due to this reason, our complicated regular expression works faster than other top-down highlighting implementations since it is only a single statement running once.

This regular expression for syntax highlighting is executed whenever the source code is altered via editing.

6.1.2. Indentation

Implementation of indentation is straightforward: preserve upper line's indentation, increase indentation at the beginning of blocks and decrease at the end of the blocks. This is implemented via Javascript by tracking newline (enter) keys.

One special situation is, tab character moves focus out of the field but it is used for indentation by most of the programmers. Thus we caught tab events via javascript, killed those events and converted them to indentation. Multi-line indent via tab and de-indent via shift-tab are also supported.

6.1.3. Automatic Bracket and Quote Closing

Forgetting to close brackets and quotes is very common for beginner programmers (see figure:6.2). In order to prevent this, we implemented automatic bracket and quote closing via javascript.

When the user opens a bracket (() or opens a quote (",'), the system automatically puts a closing one ()," or '). If the user accidentally puts the closing bracket or quote after auto close, it is deleted. In this case, the system must identify whether the user entered the closing parenthesis or quote after auto close or the user really wants to enter that. The javascript source code is given in appendix B.1

6.1.4. Undo-Redo

Rich content editor provided by Internet Explorer cannot keep track of undo-redo feature because altering innerHTML source via Javascript results in loss of undo-redo stack. We do innerHTML source alteration for rich code editing actions such as syntax-highlighting and indentation. Thus we needed to implement undo-redo ourselves. We achieved this by managing a 100-state stack which stores text and cursor position for at most 100 states and restores it for undo and redo actions. Although there is no theoretical restriction that limits the stack for at most 100 states, it seemed suitable

Top Compile Errors Of The Week

error_explanation	frequency	hint
syntax error : '}'	4837	
syntax error : ')'	4012	
syntax error : 'return'	3888	
illegal break	3441	
'anykey' : redefinition; different basic types	3336	
illegal else without matching if	3309	
illegal case	2780	
syntax error : missing ';' before '}'	2425	
syntax error : missing ';' before ')'	2350	
syntax error : missing ';' before 'type'	1552	

Figure 6.2. An example of top compile errors of the week. As you notice, bracket and parenthesis errors are very common

when practicality and space complexity are considered simultaneously (e.g. 10 states might be too few to be useful and 1000 or unlimited number of states might result in heap overflow at the client computer).

6.1.5. Line Numbering

This is a simple task which is also realized via Javascript. Just number of lines within the code is calculated and printed next to code. The line number portion is updated when the number of lines within the code changes.

6.1.6. Error Highlighting

Normally, editors show errors on the bottom and when the user clicks an error message, the related line is marked or highlighted. In our editor, all erroneous lines are marked. Moreover, if you move the mouse cursor over the line number of an erroneous line, it shows the related errors. We got a lot of positive feedback about this property.

In order to achieve this, we needed to parse the output of the Visual Studio 2005 C++ Express Edition command line compiler, in short cl . In contrast with

gcc [61], the output of cl is not fully structured and requires some exception handling while parsing. For the C source code shown in figure 6.3 the output of gcc compiler (see figure 6.4) is more structured compared to the output of Microsoft Visual Studio 2005 C++ compiler (see figure 6.5). For the former, only splitting from ":" symbols is enough for getting the file name, error line and the syntax error whereas for the latter, additional parenthesis and additional ":" symbols inside error messages should be taken into consideration. For the parser code in PHP see appendix C.1

```
#include<stdio.h>

int main(){
    int ali ,veli;
    pr intf();
    alili=6;
    4=ali;

    anykey();
    return 0;
}
```

Figure 6.3. Example erroneus C source file

The output of gcc compiler is:

```
compileErrors.c:7: error: 'pr' undeclared (first use in this function)
compileErrors.c:7: error: (Each undeclared identifier is reported only once
compileErrors.c:7: error: for each function it appears in.)
compileErrors.c:7: error: syntax error before 'intf'
compileErrors.c:8: error: 'alili' undeclared (first use in this function)
compileErrors.c:9: error: invalid lvalue in assignment
```

Figure 6.4. Compile errors listed by gcc compiler

Where you just split from : symbol to get file name, error line and the syntax error. Whereas in Microsoft Visual Studio 2005 C++ the output is:

```
compileErrors.c(7) : error C2065: 'pr' : undeclared identifier
compileErrors.c(7) : error C2146: syntax error : missing ';' before identifier 'i
compileErrors.c(8) : error C2065: 'alili' : undeclared identifier
compileErrors.c(9) : error C2106: '=' : left operand must be l-value
```

Figure 6.5. Compile errors listed by Microsoft Visual Studio 2005 C++ compiler

6.1.7. Auto Complete

Our auto-complete implementation is pretty primitive compared to many editors. Our editor just prints the list of fitting words or functions when a new word is started to be typed. It does not yet provide support for members of struct variables. However, it has support for functions, where it autocompletes functions with its parameters.

6.2. Privacy and Security

Similar to a web based email client, our system is accessed via a username and a password. All students use their student numbers as username. They set their private passwords at the beginning of the semester. Keeping their password is their responsibility again similar to a web based email client.

6.3. Course Content

In addition to our coding environment, we put course notes on the web within our system. Those course notes also contain editable and executable C programming examples which enable students to execute and alter them and see their results within the same interface. We also implemented a project submission interface within the compiler which is similar to an email attachment interface.

6.4. Feedback

We implemented simple forms for students to report bugs and enter their requests about the system.

The comment color was previously light green. One student stated that light green on white background is unreadable. Thus we altered comment color to dark green. Two students requested a bracket matching feature. We added bracket coloring where matching brackets are colored with the same color (see figure 8.1).

Additionally, the system logs almost every possible information that can be gained from the use of the system. Students' compile errors, created and edited source files, visited compiler parts etc. are all logged with timestamps. Currently, only a little portion of this log is used for feedback which is the publishing of top compile errors of the week based on compile error logs. However, this data can be used for other purposes as well.

6.5. Authentication

At Boğaziçi University, every student is given a unique student number by the registration office. Similar to a web based email client like Yahoo [10], Hotmail [11] or Gmail [12], we developed an authentication system which accepts student number as username and a private password. Students first login with a default password, then they set themselves a private password.

Previously, session feature of PHP was being used for storing authentication info on the server side. However, due to a PHP bug [62], using session and exec results in random hangs. In other words, when the session is enabled, an exec command within PHP code may create a critical race condition pausing but not ending the script execution. We need exec command for compiling C sources to exe files. Random hangs are pretty problematic especially during exams. In order to overcome this, currently session info is kept via Cookies at client side. In order to prevent fraud, Cookies are encrypted.

6.6. Tiers in Detail

6.6.1. Data Tier 1: Database Storage via MySQL

For detailed information about database architecture, please refer to 5.3.1.

6.6.2. Logic Tier 2: Database Interaction via Stored Procedures

Consists of MySQL stored procedures. Those are mainly used for logging and feedback mechanisms. Authentication mechanism is independent of database, thus also independend of this tier.

Used stored procedures and their purposes are listed below. For their source codes, see appendix A.

- **bug_report_insert:** Used for inserting a bug report into database
- **error_explanation_toplist:** Used for selecting "top compile errors of the week".
- **error_explanation_toplist_by_student_no:** Used for selecting "my top compile errors of the week". In other words, selects that student's top compile errors of the week.
- **error_hint_insert:** Used for inserting error hints.
- **error_log_insert:** Used for inserting compile errors of the students.
- **file_action_log_insert:** Used for logging/inserting students' actions on source files such as view, save, compile and run.
- **page_visit_log_insert:** Used for logging visited urls bu students.
- **part_select:** Used for converting part name to part id within other stored procedures.
- **recommendation_insert:** Used within feedback mechanism. Used for inserting recommendations and feature requests given by the students.

6.6.3. Data Tier 2: Filesystem storage

Used for storing source files, student passwords etc. Detailed information about Data Tier 2 can be found in Architecture chapter (see 5.3.3).

6.6.4. Data Tier 3: Cookies

Used for authentication validation. Since Cookies are client based storage, in order to prevent fraud, we utilized encryption and hashing mechanisms for security. We will not go into details due to security reasons.

6.6.5. Logic Tier 1: PHP coding

6.6.5.1. PHP Classes.

- **Config:** Contains system configuration constants. However, in order to achieve concatenations of some string config values or calculated values, instead of constants, functions are used. Members of Config class are:
  - **home_disabled():** Boolean function that determines whether home, course_notes, project, feedback and settings sections are disabled or not. Set to true during exams to prevent access to students' work source files or online course notes.
  - **staff_disabled():** Boolean function that determines whether staff section is disabled or not. Since staff have access to exam passwords, this is set to true on exam servers before the exams to prevent any possible leakage of exam passwords to public. During exams, this is set to false to enable staff to lookup students' exam passwords during the exam if necessary.
  - **exam_disabled():** Boolean function that determines whether exam section is disabled or not. Set to true on public server always since exam is not realized on public server. It is set to true before the exams on exam servers, set to false during the exams to enable access.
  - **exam_password_key():** Students can determine their passwords on public

server for studying. However, on exam server, they are given another password just for loginning the exam section. This is done to prevent password sharing. Exam passwords are determined just before the exam uniquely. This function returns the exam password private key which determines all students' exam login password. Altering this key alters all the exam passwords which is done before every exam.

- **database_disabled():** Determines whether database is disabled or not. Since database is not crucial for systems execution, but only used for logging, disabling it does not crash the system. Database is disabled on exam servers due to stability and performance issues (see 5.1.6).
- **database_disabled_return():** When the database is disabled, calling database functions returns the value determined by this function instead of returning resultsets etc. Set to true for a flawless database disabling.
- **database_host():** Web address of database server. Currently the database is stored in public server.
- **database_username():** Username for connecting to database.
- **database_password():** Password for connecting to database.
- **database_database():** Name of the database.
- **current_project():** Every semester, three projects are given, namely project1, project2 and project3. This function returns "project1" during the submission period of project1 etc. A folder under accounts/project folder is created for each project and submissions are stored there.
- **date_time_format_filename():** Submitted projects are renamed with submission date and time and student's number. This function returns a date and time format which is used within PHP's date function [63] and returns a date appropriate to use within filenames. e.g. it does not contain characters like : or /. Currently it returns Y_m_d__H_i_s which gives something like 2008_04_20__21_21_11.
- **date_time_format_sortable():** Files under home section are sortable via creation date. Creation dates are published in such a format that ordinary string sort results in date sort. This function returns a date-time format which is used within PHP's date function and returns a date appropriate to

use within sortable lists. It returns Y.m.d H:i:s which gives something like 2008.04.20 21:21:11.

- **date_time_format():** When sorting or file naming is not an issue, date format returned by this function is used (fed to PHP's date function). This function returns d.m.Y H:i:s which gives something like 20.04.2008 21:21:11.
- **default_password():** When the accounts are first created for students, they login with a default password, then they alter this password to something private. Currently it is "deneme".
- **default_non_student_password():** When the accounts are first created for instructors or assistants, they are given a default password which they change later. This is not equal to default password given to students. It is usually changed every semester and it is not published here for security reasons.
- **accounts_folder():** For using file system for storage (which is Data Tier 2), the root directory must be given. This function gives the root directory for Data Tier 2.
- **role_accounts_folder(**
  $userRole): For a user "foo" to enter "admin" section, there must exist a directory like accounts_folder/admin/foo. Formally, for a user X to have authorization to enter part Y, there must exist a directory like accounts_folder/Y/X. At authorization, this function is used for checking whether that directory exists or not. This function returns accounts_folder/ $userRole where user role is Y (e.g admin).
- **password_files_folder():** Returns the folder where password hashes are stored. This is used for authorization.
- **password_file_folder():** Returns the password file folder for currently loginned student. Used within update password section
- **password_file():** Returns the file which contains the password hash. This is used within update password section.
- **source_files_folder():** Returns the folder where source files (C files of the student) are stored.

- **Cookie:** Authorization information is stored within Cookies. This class handles

Cookie input and output. It handles encryption-decryption issues also which is crucial for security. It manages Cookie as a hashtable with string keys and values. Members of Cookie class are:

- **set(**
  $key,
  $value): Sets the given value to the given key. Normally, set Cookie values can be read after a refresh or redirect. However, this set operation synchronizes $_COOKIE [64] array of PHP and thus guarantees synchronized read without the need for refresh or redirect.
- **get(**
  $key): Returns the value of the given key from Cookie.
- **delete(**
  $key): Deletes the value for the given key. Deletion operation is made by setting the expiration time to a past time.
- **exists(**
  $key): Checks whether a value for the given key is set or not.
- **clear():** Deletes all the key-value pairs in Cookie.
- **renew():** Cookies are set with a timeout for security. However, at any user action (e.g. go to another page, save or compile the source etc.), Cookies should be renewed in order to prevent timeout during working. For now, for performance reasons, Cookie timeout is around one week and renew function is not used.
- **encrypt(**
  $str): Private function which is used for encryption of Cookie content. Details will not be explained due to security reasons.
- **decrypt(**
  $str): Private function which is used for decryption of Cookie content. Details will not be explained due to security reasons.

- **DBAccess:** Contains methods for accessing the database. Although it has both methods for directly executing SQL commands and executing stored procedures, only stored procedures are used within the program. Thus Logic Tier 1 only interacts with Logic Tier 2 (stored procedures), not directly the Data Tier 1

(MySQL interaction via SQL). Members of DBAccess class are:

- **__construct():** Constructor method for DBAccess object. It connects to the database.
- **connect():** Private method which connects to database. Called within constructor.
- **executeQuery(**
  $query): Executes the given SQL query. Used for executing SELECT queries. Returns the result of SELECT query as an array whose members are records as associative arrays. Since Logic Tier 1 only interacts with Logic Tier 2, direct SQL execution is not realized thus this method is not used directly.
- **executeUpdate(**
  $query): Executes the given SQL query. Used for executing INSERT, UPDATE and DELETE queries. If the executed query is an INSERT query and an autoincrement key is generated, this method returns it. Since Logic Tier 1 only interacts with Logic Tier 2, direct SQL execution is not realized and thus this method is not used directly.
- **executeQueryStoredProcedure(**
  $stored_procedure_name,
  $arr_parameters): Executes the given stored procedure with given parameters. It is assumed that last statement of the given stored procedure is a SELECT query and this method returns the result of SELECT query as an array whose members are records as associative arrays. This method calls executeQuery method.
- **executeUpdateStoredProcedure(**
  $stored_procedure_name,
  $arr_parameters): Executes the given stored procedure with given parameters. If the last statement of the given stored procedure is an INSERT statement and an autoincrement key is generated, this method returns it. This method calls executeUpdate method.
- **close():** Private method which closes the database connection. Called within destructor.

  - **__destruct():** Destructor method for DBAccess object. Closes the database connection.

- **rsToHtml:** executeQuery and executeQueryStoredProcedure methods of DBAccess class returns resultsets. This class contains methods for converting resultsets to HTML entities such as tables. Mainly used for printing resultsets as HTML tables. Members of rsToHtml class are:
  - **getColumnNames(**
    $rs): Returns the column names for the given resultset. e.g. for a query "SELECT student_id,name FROM students", after the returned resultset is fed to this method, this medhod will return an array containing "student_id" and "name" strings.
  - **rsToTable(**
    $rs,
    $is_header_exist,
    $table_attributes): Prints the given resultset as an HTML table.
- **Authorization:** Class for handling authorization mechanism. This class manages student and staff logins and authorizations on various system parts. Previously $_SESSION feature of PHP was used. However, due to a php bug [62], we needed to abandon $_SESSION use. Now, Authorization class uses cookies for authorization info via the Cookie class we described above. Members of Authorization class are:
  - **setUserRoles(**
    $arrUserRoles): Sets the loginned user's roles if the login is valid. User roles are: student, exam_student, staff and admin. Each role have authorization to enter to various system parts.
  - **getUserRoles():** Returns the current user's roles.
  - **deleteUserRoles():** Deletes the current user's roles. Used for logout operations.
  - **loginError():** If the login is invalid, this method redirects the user to login screen. Login can be invalid in three situations:
    1. If the user enters a wrong username-password pair

2. If the user tries to access to a page that he is not authorized
3. When the user logouts

Appropriate error messages are printed and user is redirected to login page in those situations.

- **checkLoginLogout():** Checkes whether the user attempted to login or logout. If so, it executes the appropriate action.
- **home_loginOperations():** Handles login operations for home, project, staff and admin parts of the system. Assigns the user appropriate roles and inserts necessary key-value pairs into cookie.
- **home_logoutOperations():** Handles the logout operations from home, project, staff and admin parts of the system. Clears the cookies appropriate key-value pairs.
- **exam_loginOperations():** Handles login operations for the exam part. Logging for suspicious cheating attempts are also handled here.
- **exam_logoutOperations():** Handles the logout operations from exam part. Clears cookies appropriate key-value pairs.
- **exam_password():** Generates the exam password for the user. Used for both checking exam logins and publishing exam passwords on staff part.
- **__construct():** Constructor method for Authorization class. For authorization purposes, creating a new instance of Authorization class, thus invoking this method is enough. This handles and manages all other operations.

• **CodeSubmitHandler:** Handles save and compile requests. Compile error logs for feedback and error highlighting used within compiler interface are also handled by this class. Members of CodeSubmitHandler class are:

- **__construct(**

  $db): Constructor of CodeSubmitHandler class. Database object is required for logging actions (for Logic Tier 2 (stored procedures) interaction).
- **printDate():** Prints current date and time. Used at frame where compile or save actions are submitted.
- **fileNameSecurityCheck(**

  $student_file_name): Used to check whether the given file name is valid or not. It is used for preventing attacks that can be made by modifying post

data via altering the source file names.

- **getLastFolder():** Returns the current scripts last folder. Last folder determines the part of the system thus this method is used for getting the part of the system.
- **save(**
  $student_file_path,
  $student_file_name_without_extension,
  $student_file_extension,
  $header_code,
  $trailer_code): Handles the save request of the user. Saves the source file. Also logs save action for feedback.
- **compile(**
  $student_no,
  $student_file_path,
  $student_file_name_without_extension,
  $student_file_extension): Handles the compile request of the user. Compiles the given source file. Emits compile errors on screen and highlights erroneous lines of the source code if any. Also handles compile error logging and compile action logging for feedback.
- **action():** This is the method which handles all code submit actions including save and compile. This method manages (calls) the upper appropriate methods for appropriate actions.

6.7. Exam-Specific Attributes

6.7.1. Exam Environment

For the exams, students see a very similar interface to ordinary environment with a difference that questions are given as previously created as C files and student does not have the right to add a new question or delete one question. This is adequate since at cmpe150 course we do not require students to write their own libraries.

6.7.2. Header-Trailer Codes

Another requirement is, some questions contain some code at the beginning and at the end of the question which should not be modified by the student (e.g. student is required to write a function which is used within main where main is previously written and untouchable). Therefore, we have header and trailer code support. Header code is the code portion at the beginning of the source file which cannot be modified. Trailer code is the code portion at the end of the source file which cannot be modified. Students are asked to edit only in between (see figure 7.34).

6.7.3. Security

6.7.3.1. Exam Password. For ordinary compiler use, students have the right to set a password for themselves. However, for exam login, students are given another password which is only valid for the exam during the exam. This prevents unethical swapping of passwords among exam takers.

6.7.3.2. Logging. Even exam passwords may become inadequate since some students can still see others exam passwords or swap exam passwords during the exam. As a further measure of security, students' login actions are logged with IP numbers of the computers they use during the exam. Thus if a student logins to multiple accounts from the same computer, it is determined.

6.7.3.3. Exam Environment. During exams, we close the public server in order to prevent students from accessing their studying files. Additionally, at our computer labs where we give exam, accessing outside of the local network is prohibited. Those precautions are taken to give exams in an isolated environment.

7. EXPERIENCE

7.1. Student Perspective

Students use our system for writing programs, accessing to course notes for studying etc. First step for any operation is to log in similar to all personalized web portals. In order to log in to the system, they need to have access to a computer with Windows operating system and Internet connection.

7.1.1. Login

In order to login the system, students should either have access to a computer with Windows operating system and Internet access, or they must be in cmpe labs. Students need to open an Internet Explorer browser. Then, if the student is in the lab, he must enter 172.16.1.233 as the address and press enter. If the student is not in the lab, he must enter `http://cmpe150-1.cmpe.boun.edu.tr` as the address and press enter.

After entering the address and pressing enter, they see the login page (see figure 7.1).

They enter their student no and password to form (see figure 7.2). At the beginning of the semester, an account is created for every student with default password "deneme" thus for their first login they enter "deneme" as password.

There is also a javascript based check for Caps Lock which can result in erroneous password entrance if accidentally left on (see figure 7.3).

For the first entrance with "deneme" password, the system forces the student to alter his password for security (see figure 7.4).

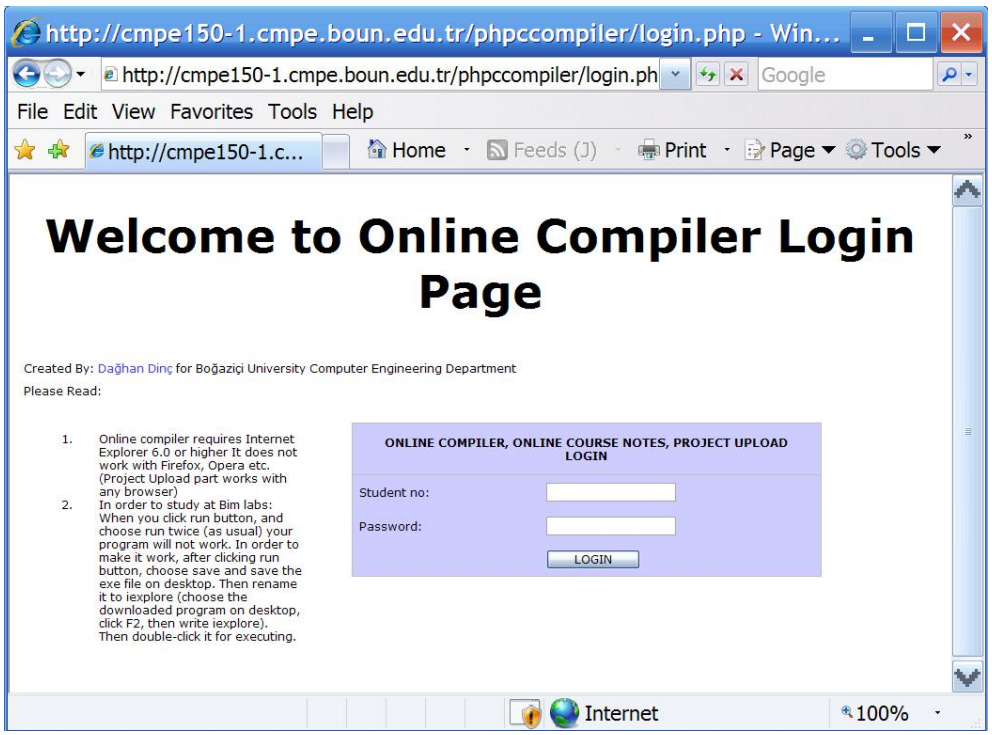


Figure 7.1. Screenshot of login page

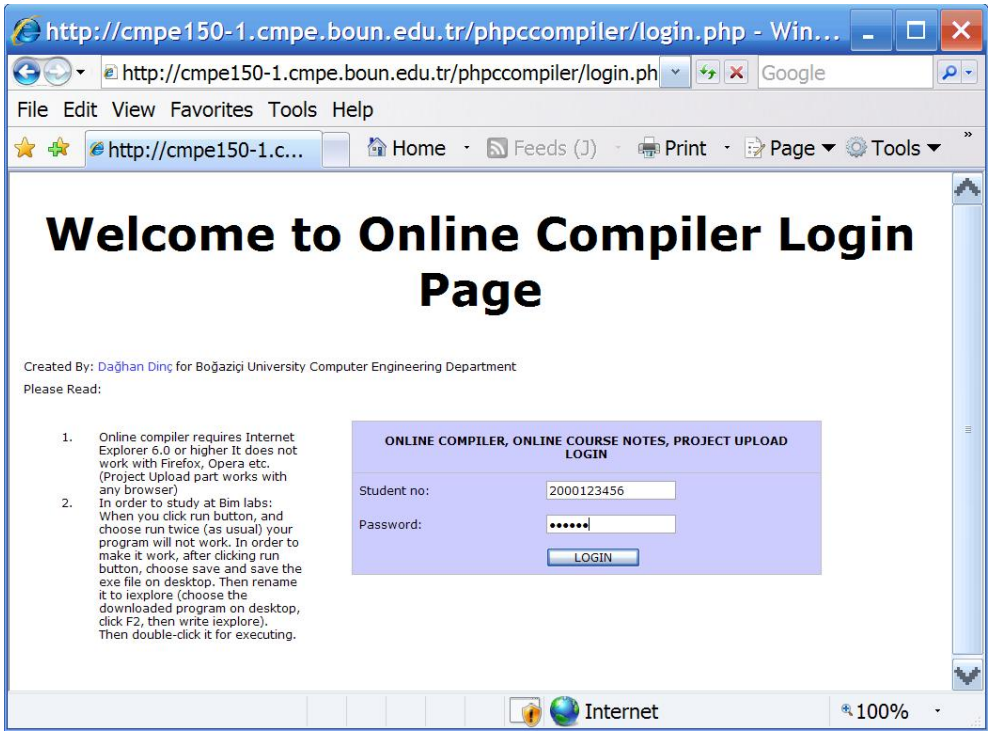


Figure 7.2. Screenshot of login page with login form filled

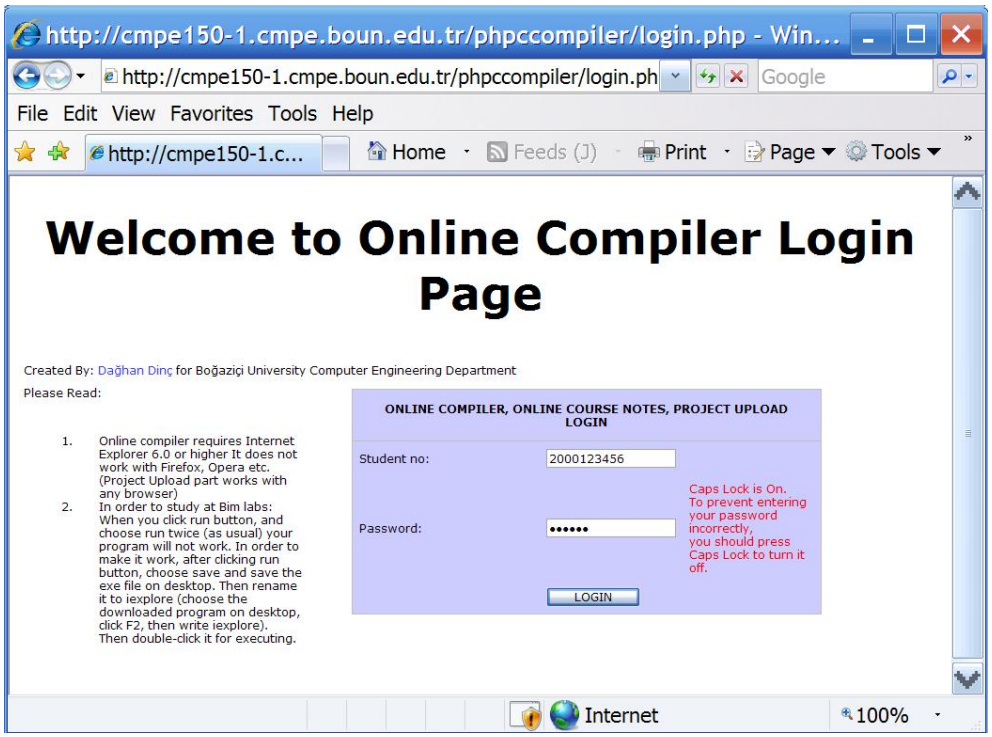


Figure 7.3. Screenshot of login page, when the student tries to fill the password field with Caps Lock open, he gets a warning

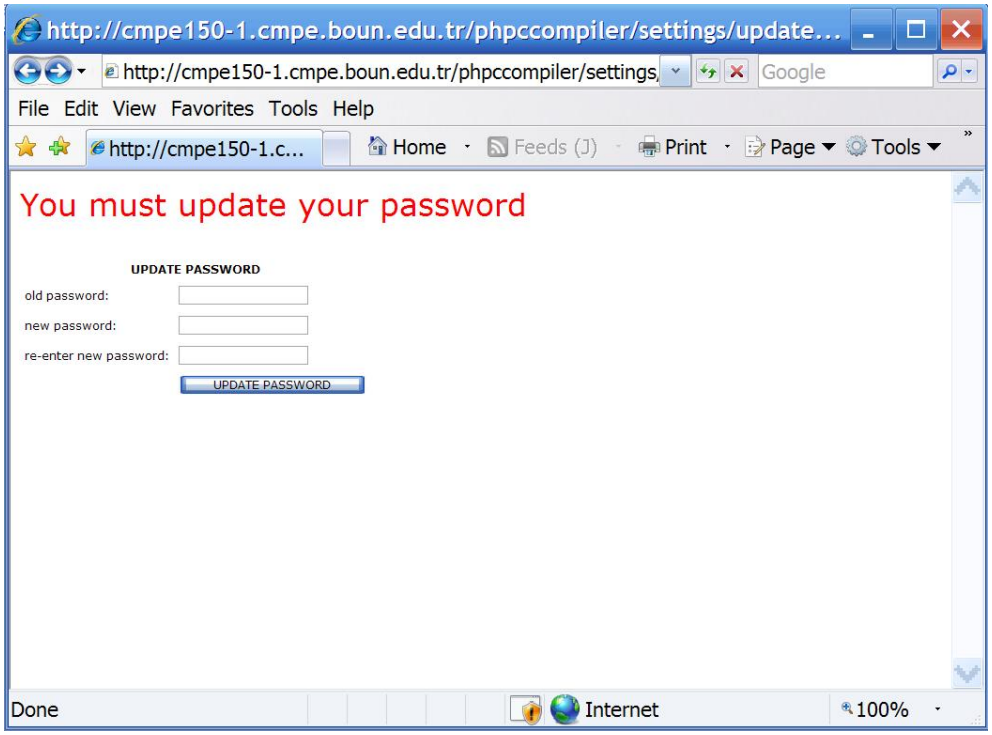


Figure 7.4. Screenshot of update password page. The student is redirected here for the first time when loginned with default password

After changing the password, the student is shown a confirmation (see figure 7.5) and redirected to home section.

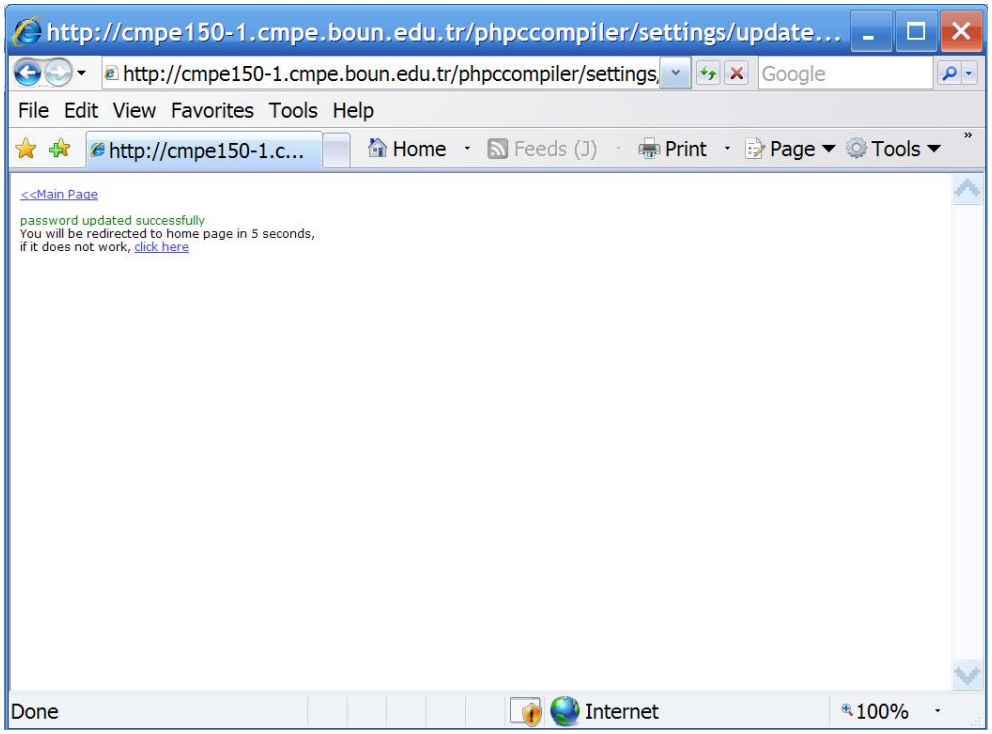


Figure 7.5. Screenshot of confirmation page

Home section (see figure 7.6) is the page where student creates and manages his source files. Also home page contains "top compile errors of the week" and "my top compile errors of the week" lists.

7.1.2. Writing Programs

For writing programs, the student first must add a new source file using the "add new file" form on top-left corner of the page. The student writes the name of the source file (see figure 7.7) and press enter or clicks the "add file" button for adding a new file (see figure 7.8).

For added source files, delete and rename operations are supported as you can see in the screenshot (see figure 7.8). For writing a program, the student must click the open button to see the editor interface (see figure 7.9). As a default source, an empty but compileable C source is printed. Here, you can see line numbering, syntax

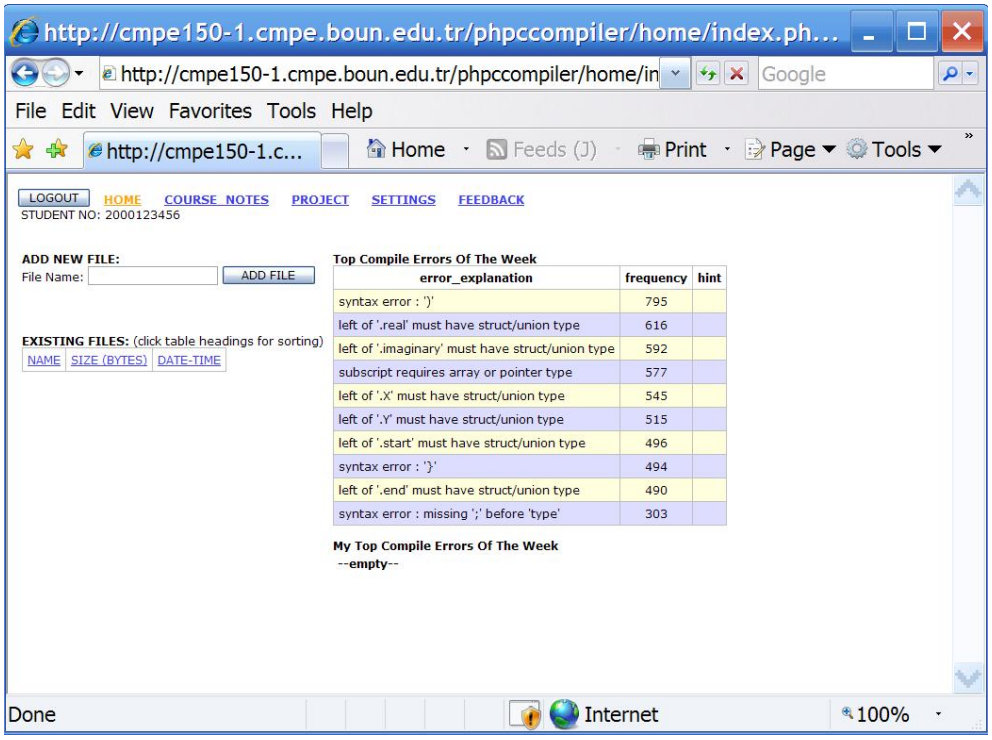


Figure 7.6. Screenshot of home page.

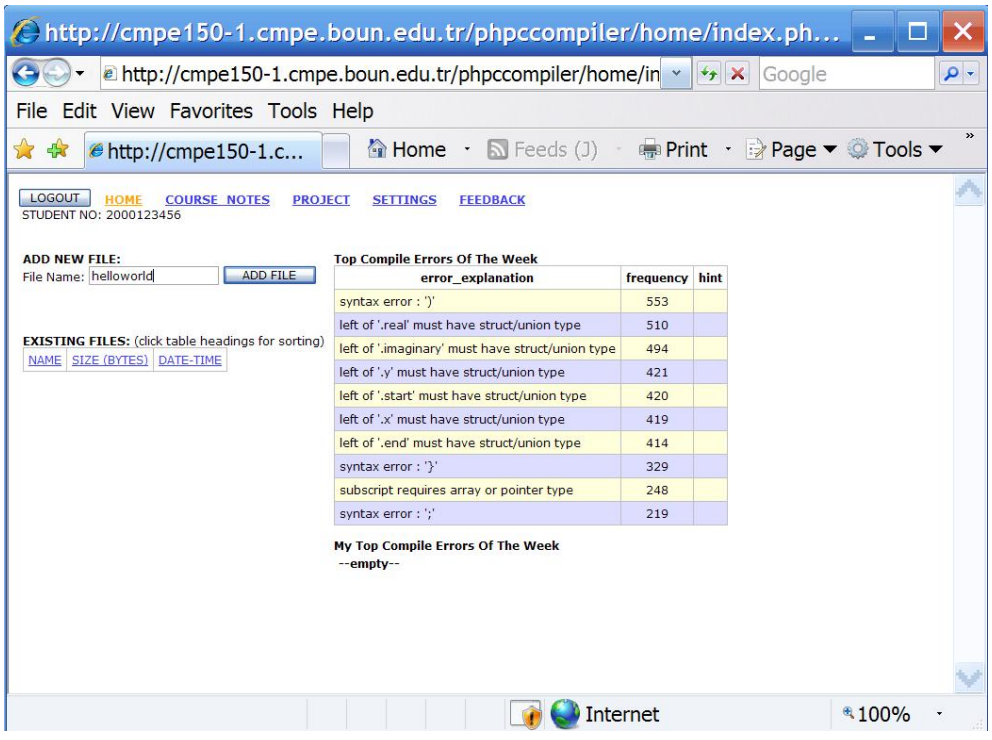


Figure 7.7. Screenshot of add new file form, filled at home page.

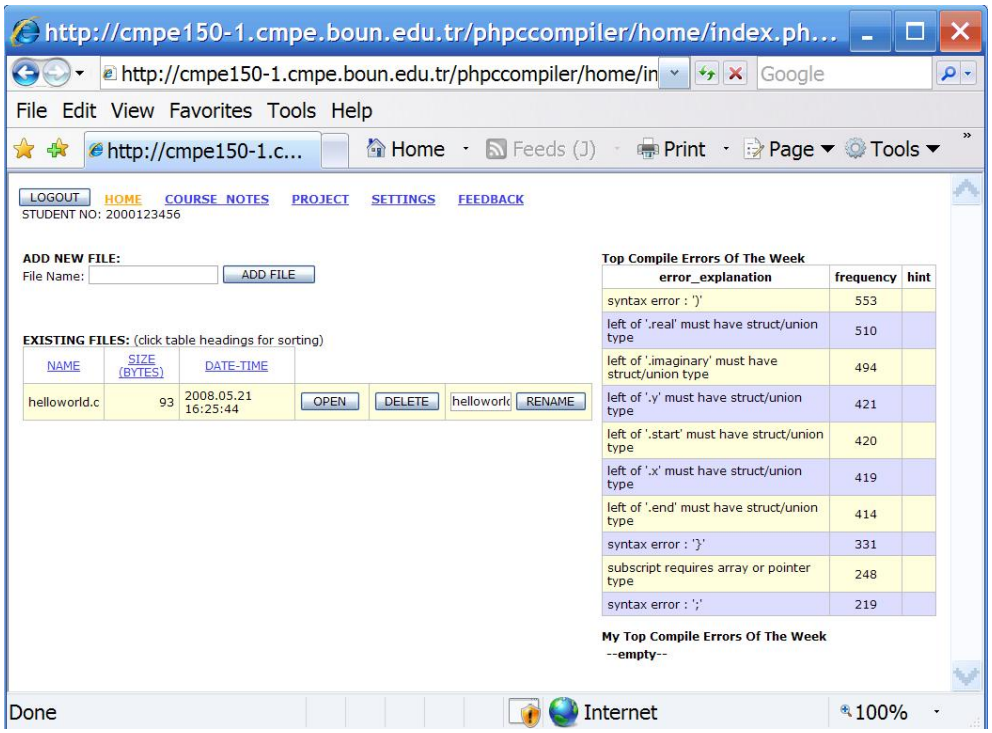


Figure 7.8. Screenshot of home page, after file added via add new file form.

higlighting, indentation in action.

This editor interface handles save, compile and run actions for the program. Also you can see undo redo buttons here. There is also support for automatic code completion for variables and common functions (see figure 7.10)

When the student clicks save button, the source is saved and a proper message is shown below (see figure 7.11). Compile button is normally deactivated. It activates when save button is pressed and deactivates when compile button is pressed. This is for preventing successful compile requests although source code is not altered. Since compilation is a resource hungry operation, eliminating unnecessary compilations decreases server load. This alteration improved exam performance of the servers dramatically.

When compile button is pressed, compilation takes place. On successful compilation, a green message and compiler output is shown (see figure 7.12).

If there are syntax errors, a red compilation message is shown with compiler

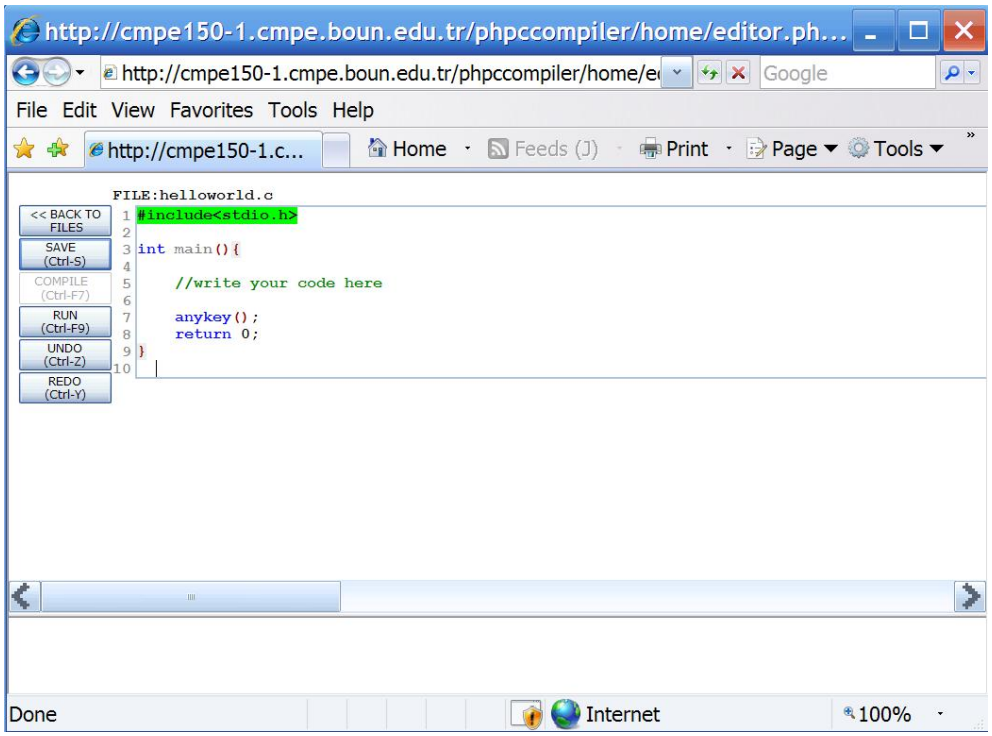


Figure 7.9. Screenshot of editor.

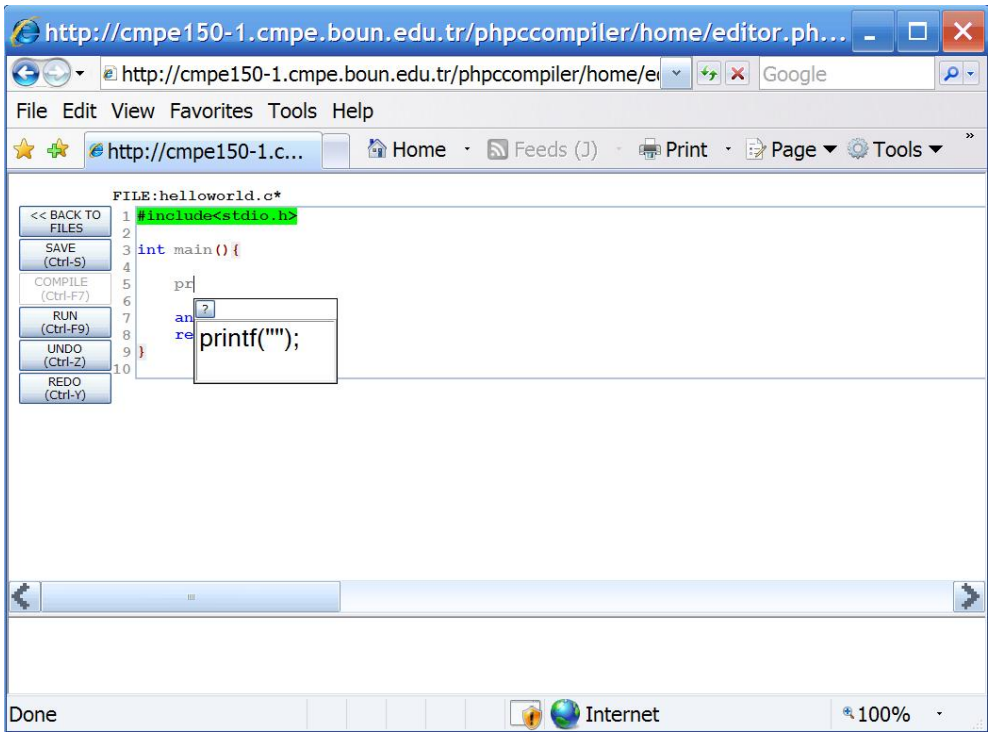


Figure 7.10. Screenshot of editor while autocompleting.

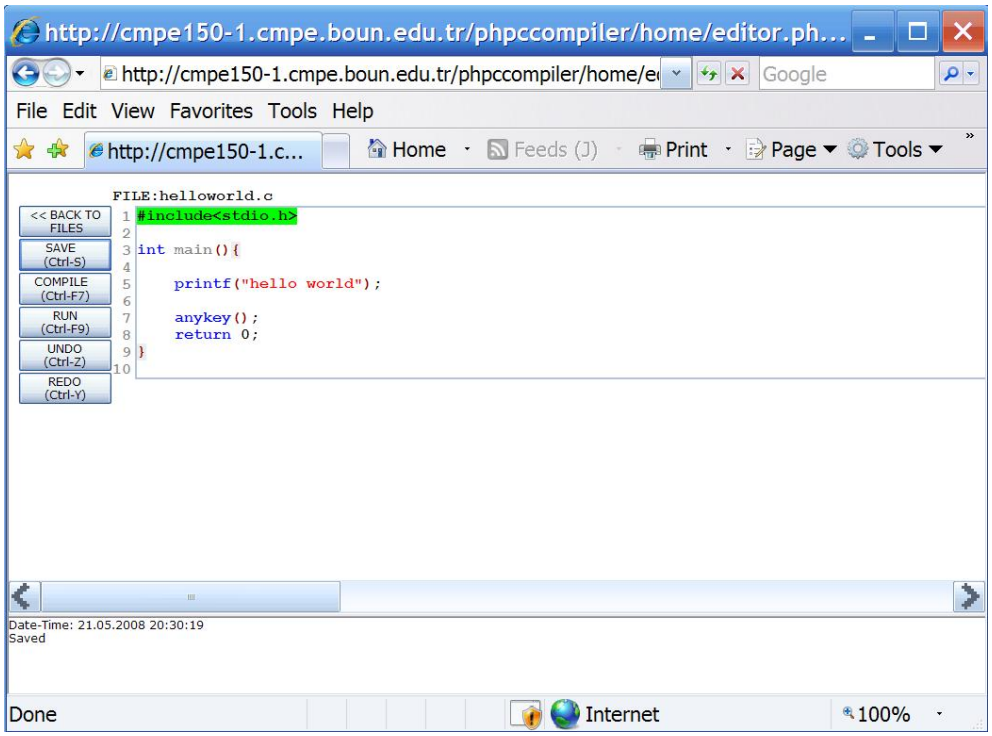


Figure 7.11. Screenshot of editor after save button is clicked.

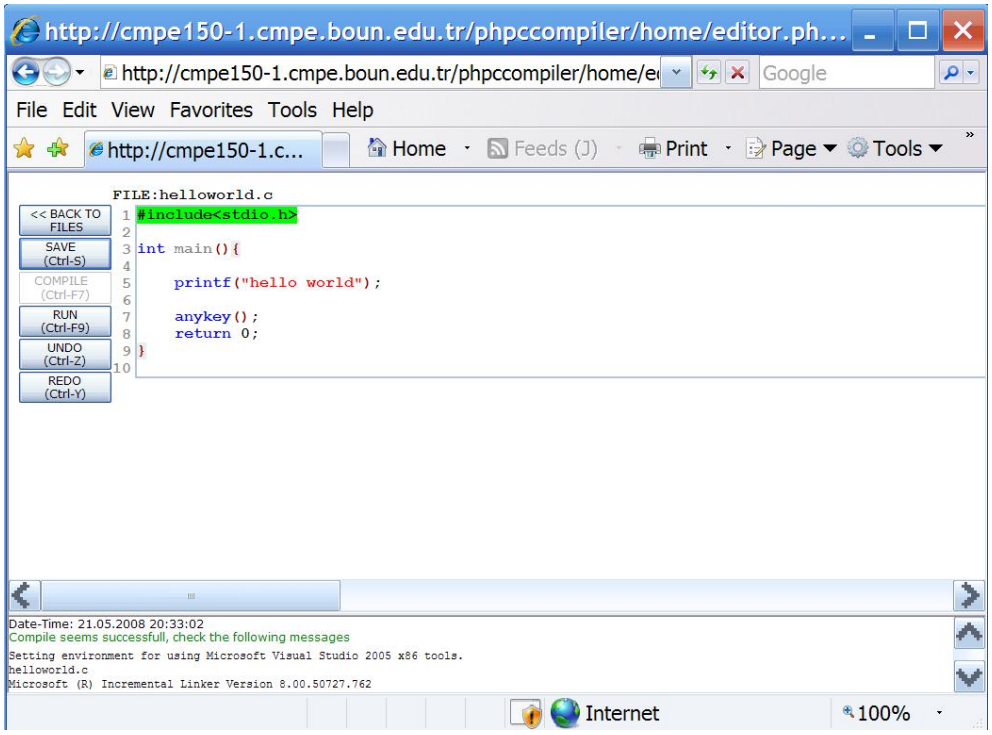


Figure 7.12. Screenshot of editor after compile button is clicked.

output listing compile errors following. In order to help student to find and fix errors more easily, erroneous code line numbers are marked with red. Additionally, if the student moves the cursor on erroneous line number, error is shown within a title box (see figure 7.13).

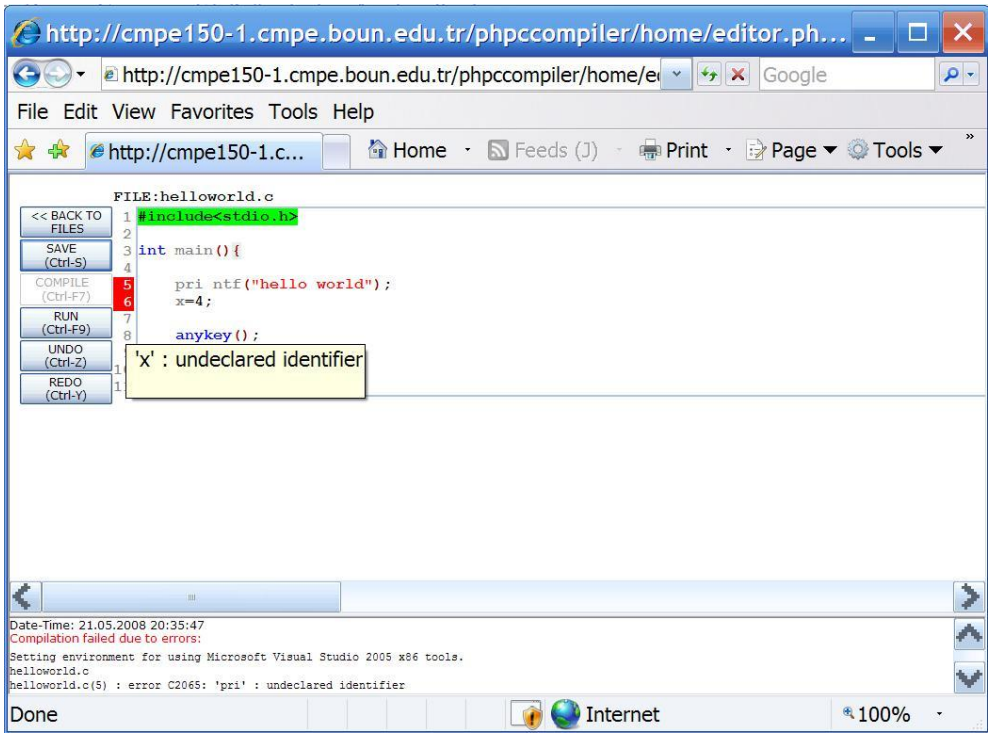


Figure 7.13. Screenshot of editor showing compile errors.

In order to run the program, the student presses run button. Running a compiled exe on the server is very risky. Additionally input output for the exe running on the server is another problem. For overcoming these factors, instead of running the exe on the server, run button simply sends the exe to the client for running. More clearly, run button is nothing but a link to the created exe. Windows asks two confirmations before running the exe after clicking the run button (see figures 7.14 and 7.15).

After confirming two confirmations, exe is run on the client (see figure 7.16).

Note that there is a function just over the "return 0;" statement within main with name "anykey()". This is not a standard C function. We added it into stdio.h for ensuring the command shell to wait for the user after showing the output. If anykey() is not used, shell window exits immediately if it is not waiting for scanf. Therefore

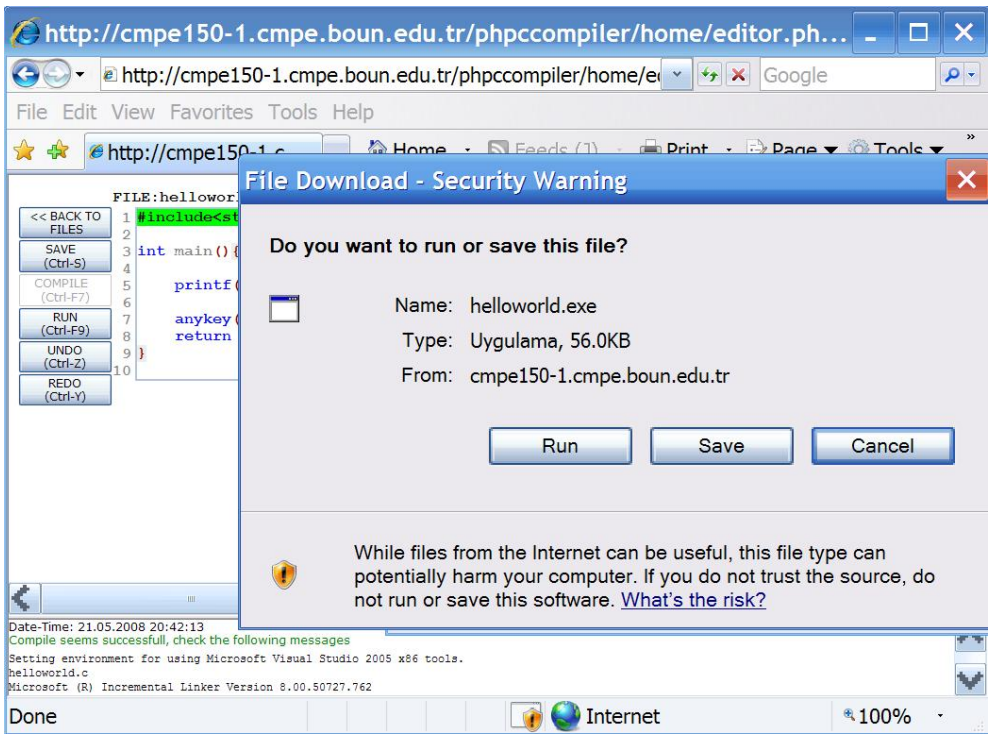


Figure 7.14. First exe running confirmation request



Figure 7.15. Second exe running confirmation request

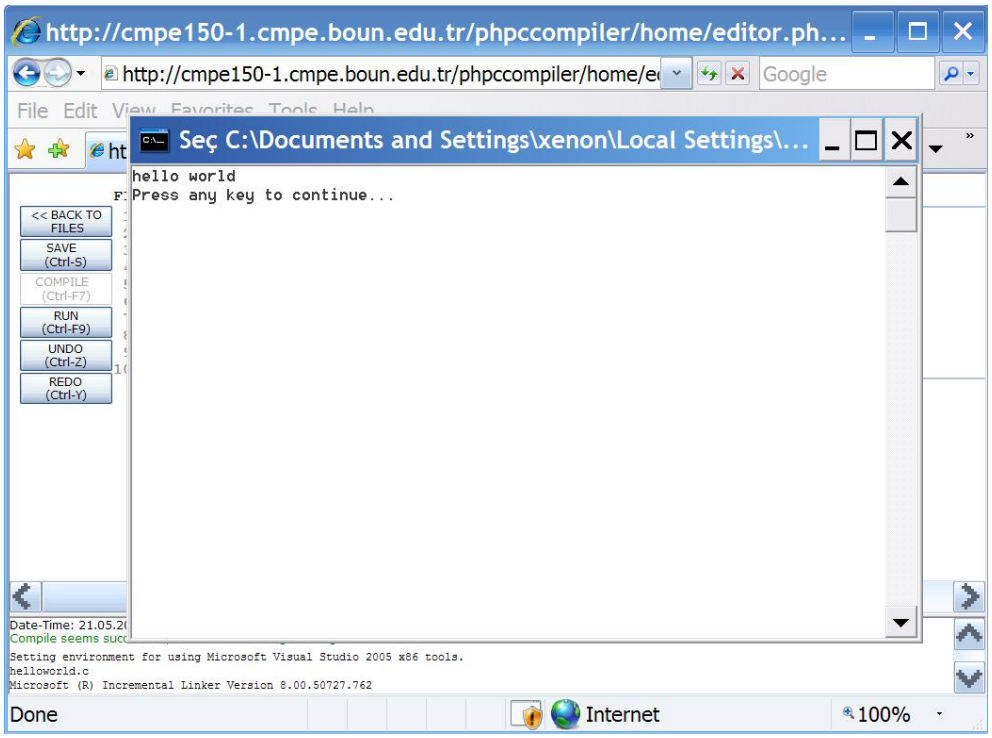


Figure 7.16. exe running on the client computer. Note that colors of the windows command shell are inverted to improve printed quality

student cannot see the program's output.

7.1.3. Online Course Notes

Course notes of cmpe150 course are available in pdf format. Students can print them for studying. Advantage of printed course notes is portability. However they cannot benefit the advantages of hypertextbooks [65] such as inner links, animated images etc. Moreover, printed source codes are impractical since for trying them, the student must type them which is cumbersome. We ported pdf course notes in HTML format as a hypertextbook where students can access to course notes and examples (see figure 7.17). Moreover, we embedded programming examples as executable and modifiable entities which enables students to easily run and alter book examples. In figure 7.18 you can see a book example as a dynamic modifiable and executable entity. All functionalities of normal editor interface are also available here. This is a highly approved property. Please note that alterations to course notes are privately stored for each student. None of the students have the right to realize global alterations.

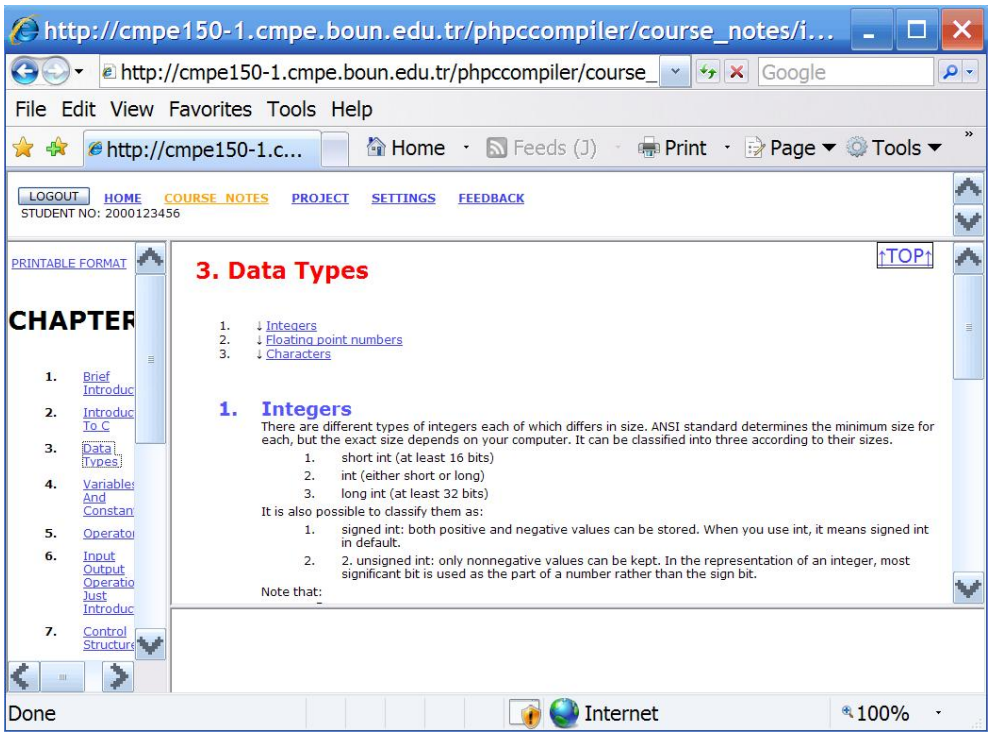


Figure 7.17. Screenshot of online course notes

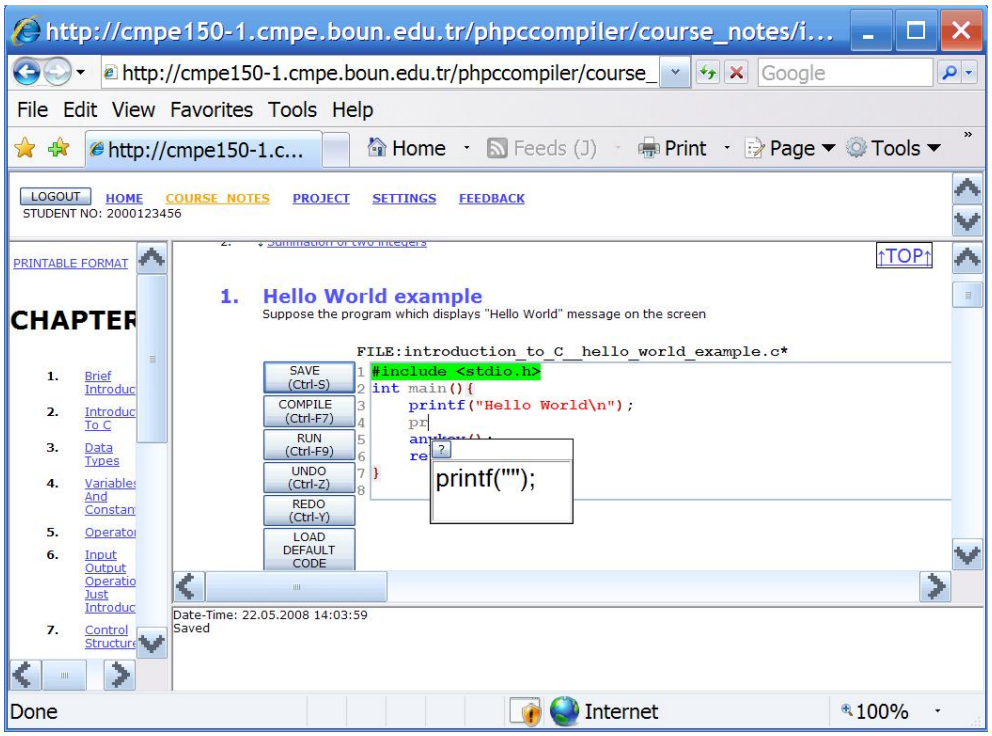


Figure 7.18. Screenshot of online course notes, working on a dynamic example

7.1.4. Project Upload

Students can submit their completed projects assignments using the system. Under project section, there is the opportunity to upload projects via a very simple form. Students have the chance to upload their projects multiple times. Only the last upload is valid, however previous uploads are also stored and listed as a reference. In order to prevent erroneus uploads, uploading of empty files and very big files are not allowed. Also only .c, .cpp and .txt extensions are allowed (.txt extension is used when uploading a project which is coded via online compiler. Students copy-paste their project source into notepad and then submit that) (see figure 7.19).

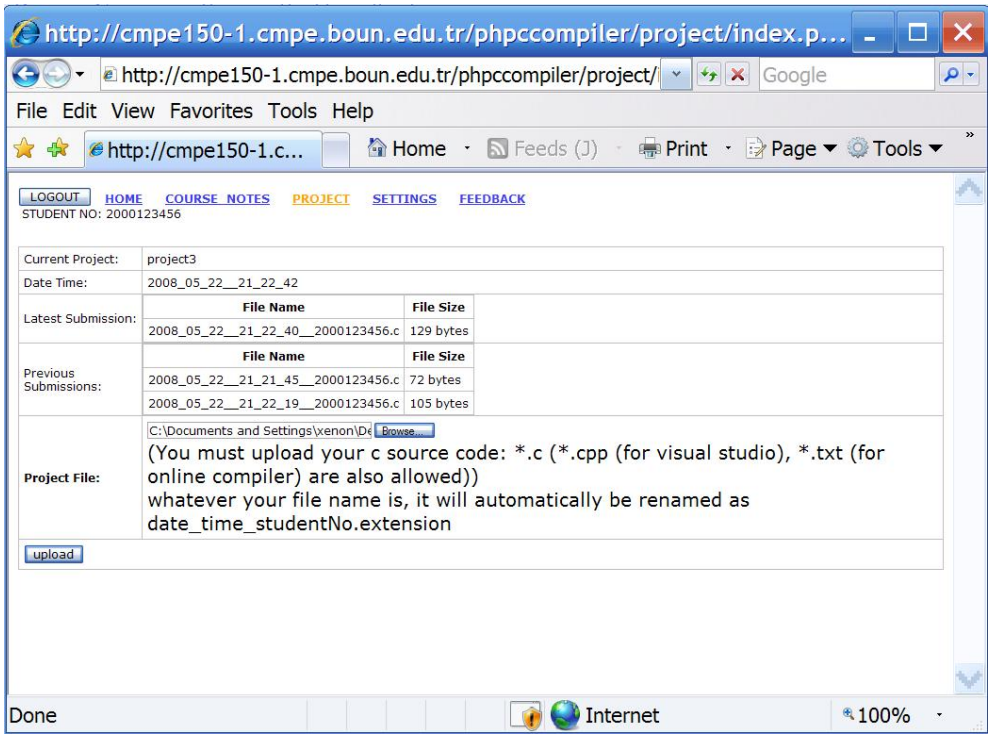


Figure 7.19. Screenshot of project upload part

7.1.5. Settings - Update Password

There is a settings section for students which currently only contains update password feature (see figure 7.20)

When the student clicks to update password link, a password update form is presented. This is the same form as mandatory update password form shown after

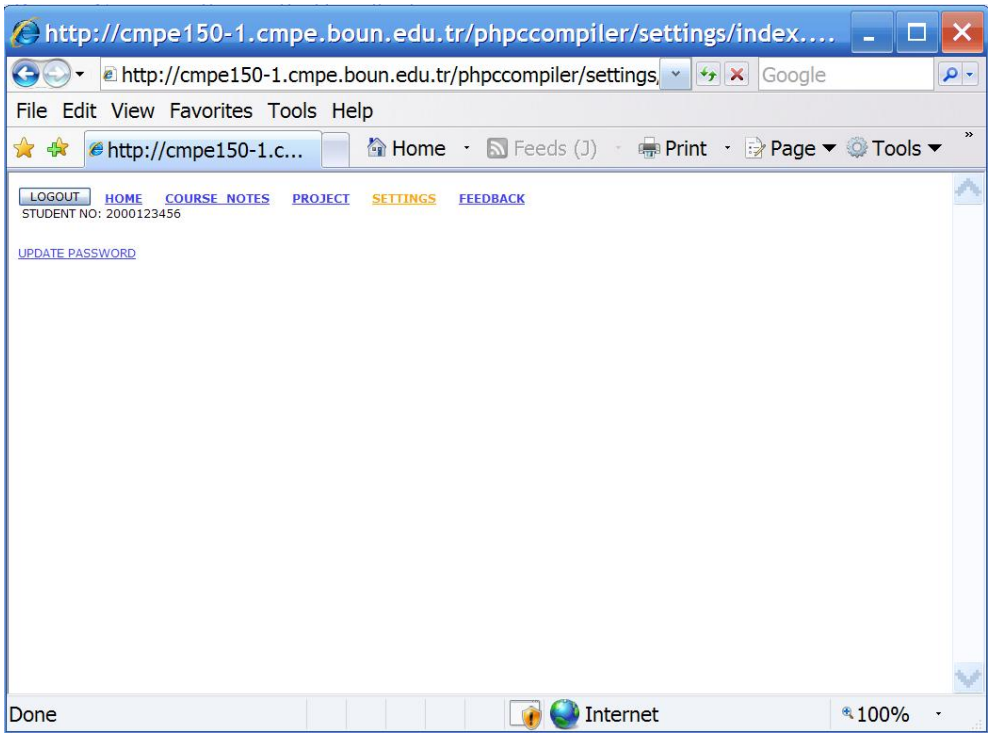


Figure 7.20. Screenshot of settings part

the first login with "deneme" password with only difference that big warning message is not shown (see figure 7.21). After password is updated, a confirmation message is shown and the user is redirected to home section (see figure 7.5).

7.1.6. Feedback

Student have the chance to enter feedback in two ways (see figure 7.22):

1. Report a bug
2. Make some recommendations

7.1.6.1. Report a Bug. In order to report a bug, student clicks the report bug link, fills the report bug form and submits it (7.23). Fields of report bug form are:

1. Subject: A brief entry for the bug
2. Explanation: Detailed description of the bug
3. How to reproduce: Steps for reproducing the bug

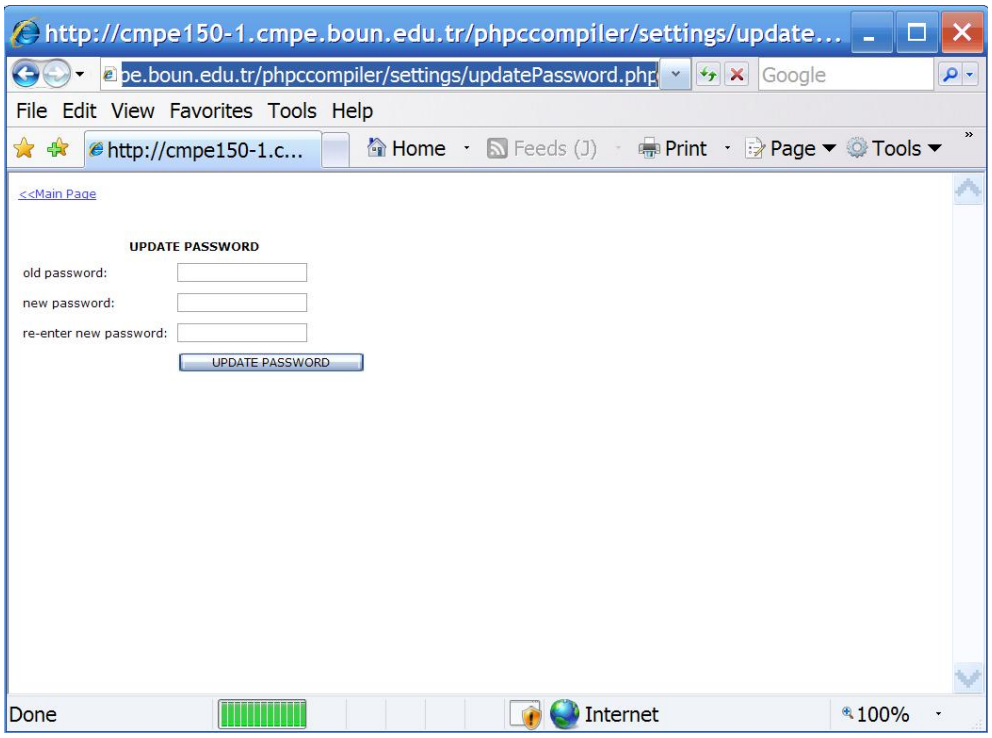


Figure 7.21. Screenshot of update password form

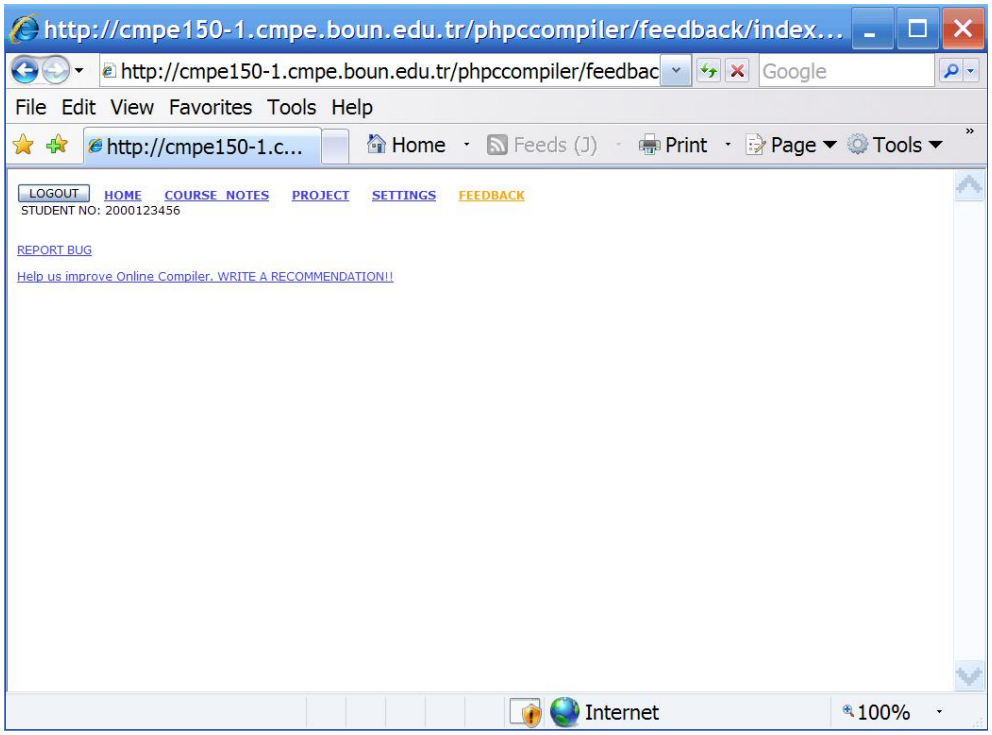


Figure 7.22. Screenshot of feedback part

These three fields are chosen to easily determine and fix the bug. Reported bugs can be seen under staff section which we will explain later (see 7.2). Bug fixes realized due to students' bug reports will be discussed under Feedback section (see 8)

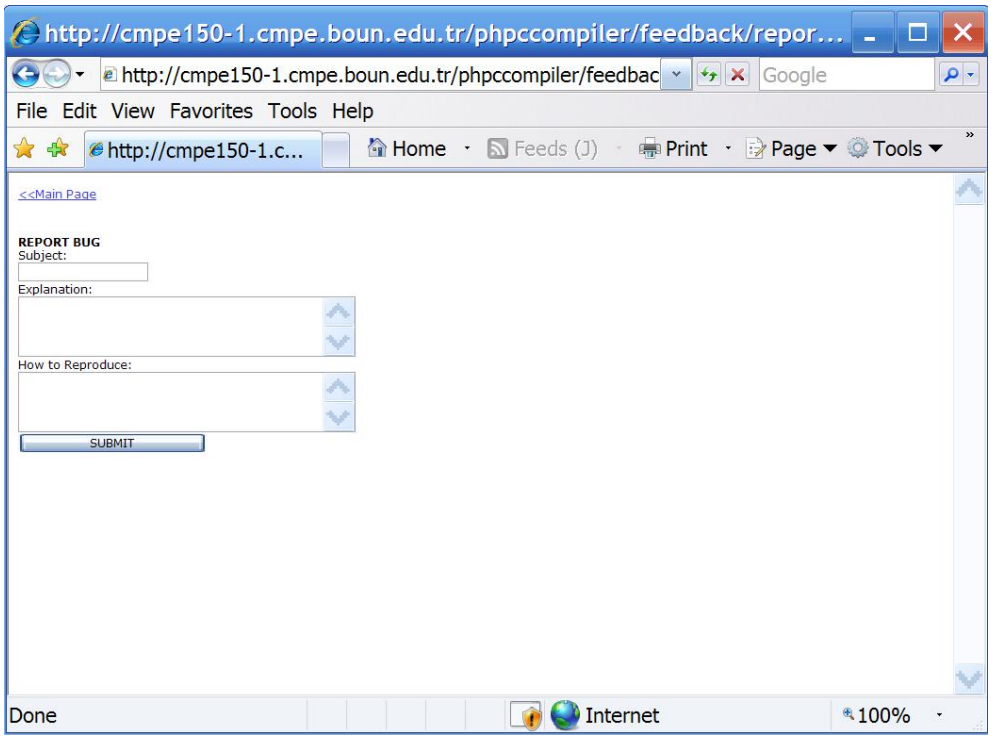


Figure 7.23. Screenshot of report bug form

7.1.6.2. Make a Recommendation. In order to make a recommendation, student clicks the make a recommendation link, fills the recommendation form and submits it (ref-fig:recommendationForm). Fields of recommendation form are:

1. Subject: A brief entry for the recommendation
2. Explanation: Detailed description of the recommendation

Recommendations can be seen under staff section which we will explain later (see 7.2). Additional features added due to students' recommendations will be discussed under Feedback section (see 8)

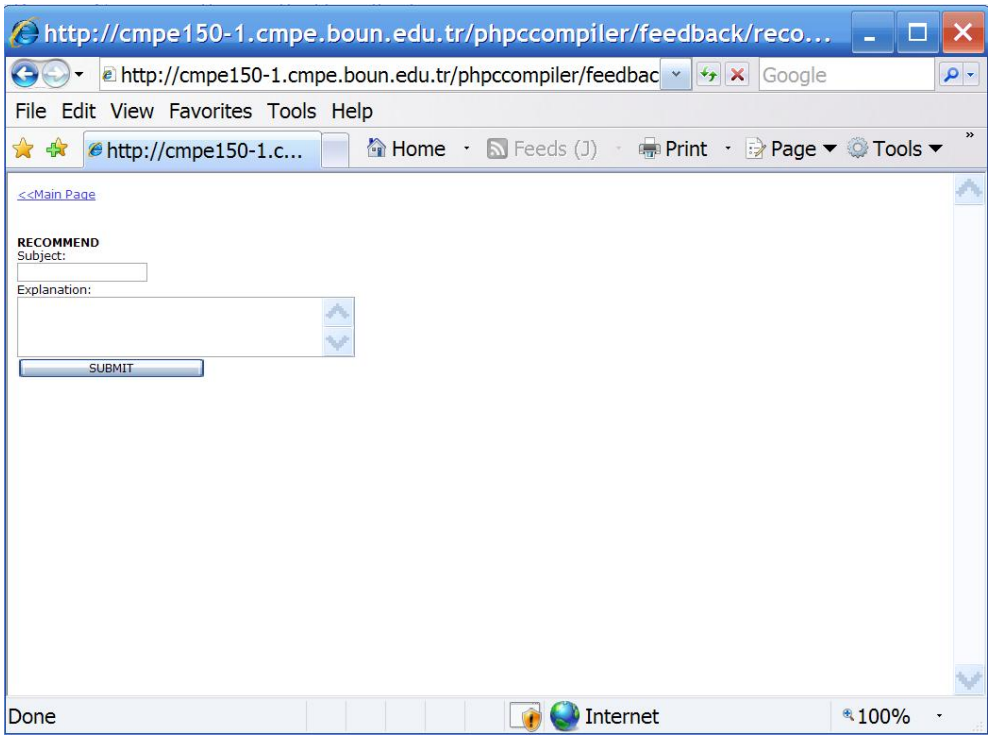


Figure 7.24. Screenshot of recommendation form

7.2. Staff

Staff section is available for teaching staff including teachers, assistants and student assistants. Students does not have access to staff section. When a staff is loginned, staff link is shown on top which is not shown to students (see figure 7.25)

7.2.1. Bug Reports and Recommendations

Staff users can see bug reports and recommendations on their section (see figure 7.25). These are used for fixing many major and minor bugs and also used for adding new features to our system. We will discuss these in more detail under Feedback section (see 8).

7.2.2. Exam Passwords

They also have right to see exam passwords of the students which they use when entering the exam. There are two ways of seeing exam passwords. They can either

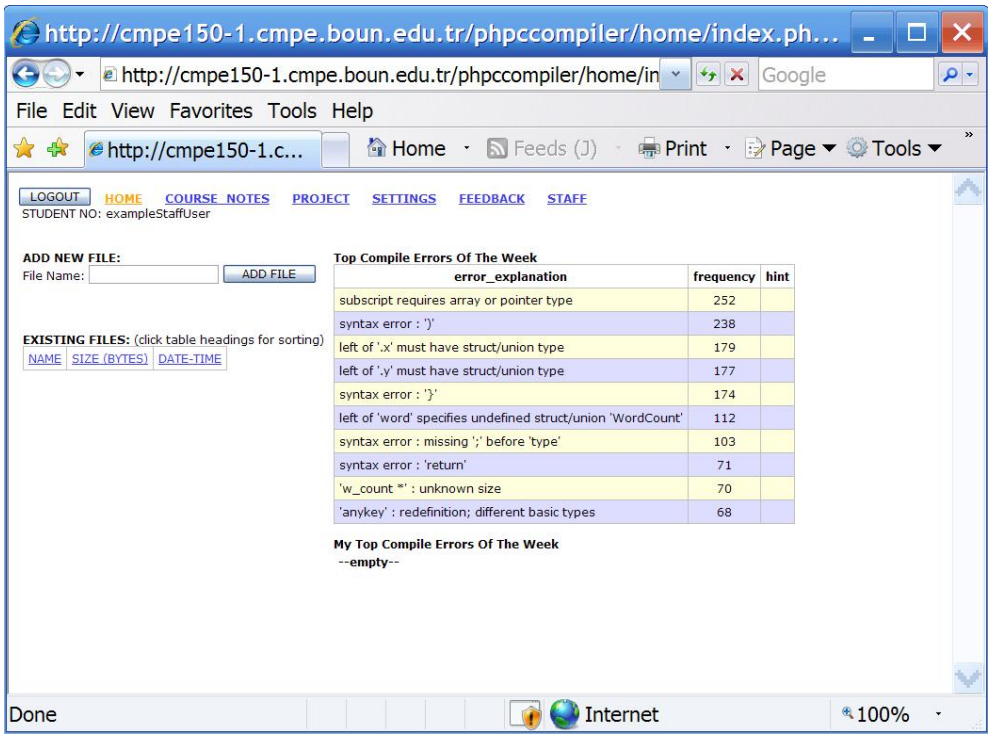


Figure 7.25. Screenshot of staff's home page. There exists a staff link on top of the page which is not visible when a student logins

list all the students exam passwords via exam password lister link (see figure 7.26), or they can list the exam passwords they want for specific users by entering them into the form under exam password finder link (see figure 7.27). In this page, only entered exam passwords are listed (see figure 7.28)

All listed exam password lists are HTML tables, which enables to copy-paste them to Microsoft Excel [66] for further working when the page is displayed with Microsoft Internet Explorer [5].

7.2.3. Error Hints

Staff users have right to enter error hints to errors that are listed as top compile errors on home page to aid students. In order to do that, staff user clicks the manage error hints link on staff page and utilizes the INSERT/UPDATE/DELETE ERROR HINT form (see figure 7.29).

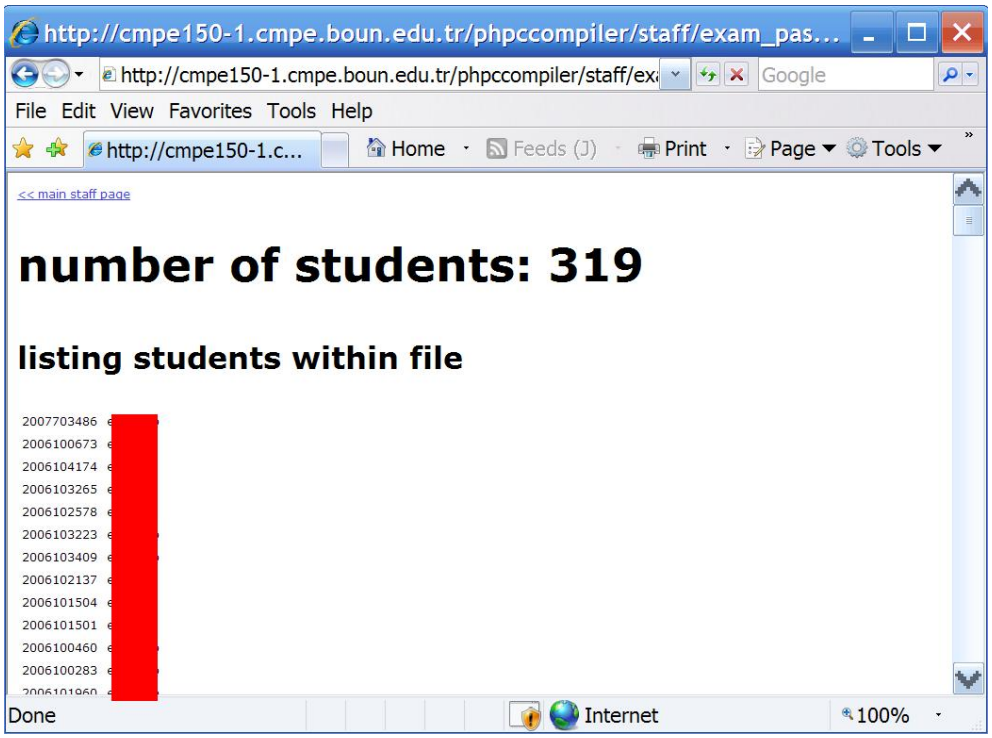


Figure 7.26. Screenshot of exam password list at staff page. Note that exam passwords are censored for security.

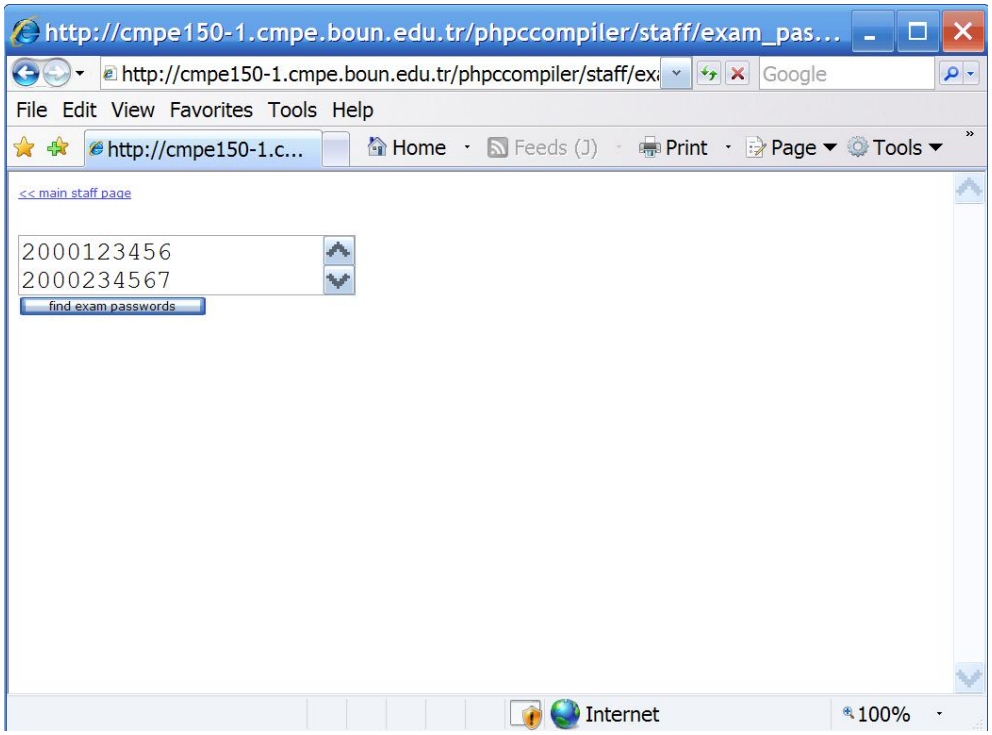


Figure 7.27. Screenshot of exam password finding form, which is filled to find exam password of two students

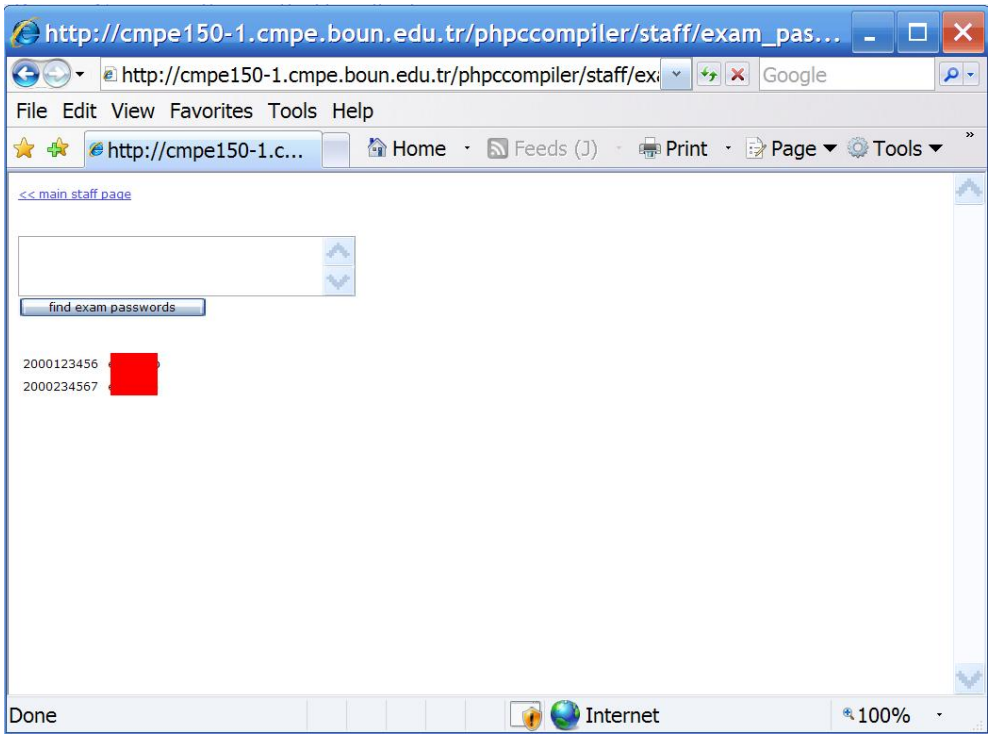


Figure 7.28. Screenshot of exam passwords listed after entered to exam password finding form. Note that exam passwords are censored for security

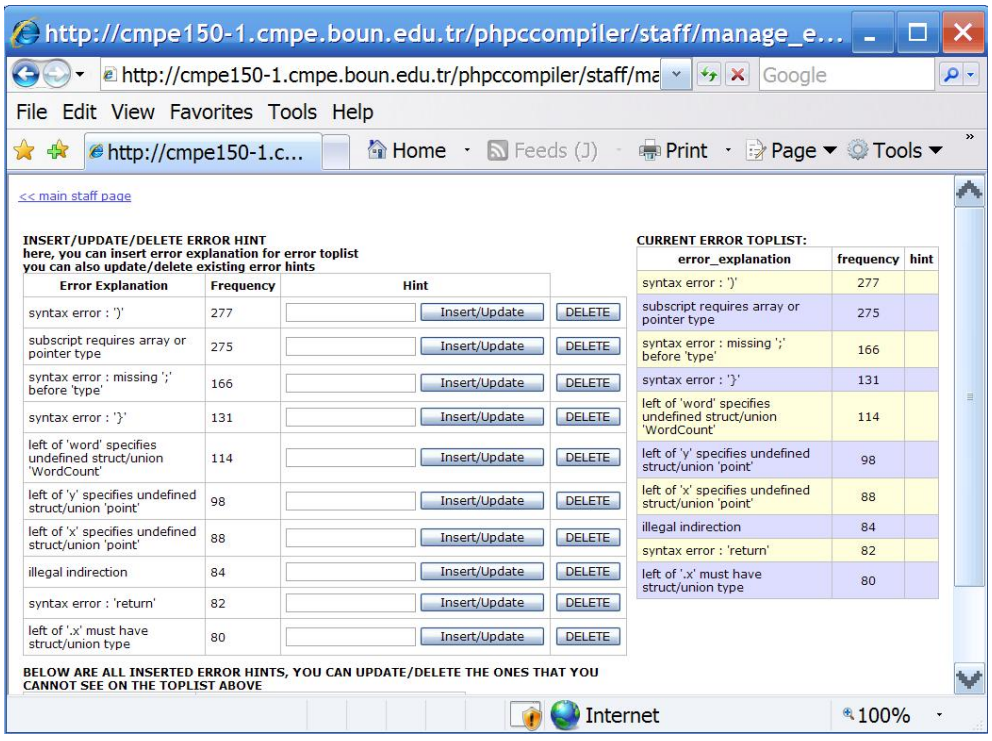


Figure 7.29. Screenshot of manage error hints form

7.3. Admin

Every semester, instructors and the assistant responsible from the online compiler is given admin rights. If a user with admin rights logs in, he sees the admin link on top and have access to admin interface (see figure 7.30).

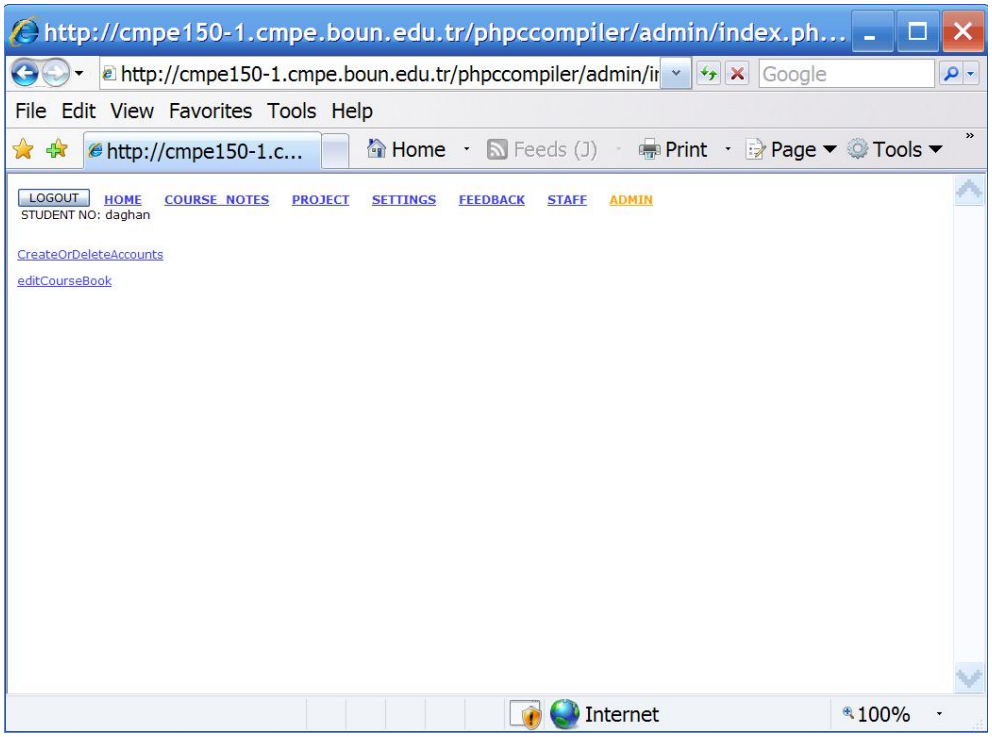


Figure 7.30. Screenshot of admin page

Admin users currently have only one additional ability: They can create or delete user accounts. As a future work, admin users will have the right to edit course notes which we will discuss within the future work section (see 10).

When the admin user clicks the CreateOrDeleteAccounts link, the admin form for creating and deleting accounts are shown. In figure 7.31 you can see the screenshot. Via the textarea on top, the admin user can add new users to the system. Below that textarea, there is a list containing every user at the system with checkboxes. Admin user can delete users or reset users' passwords to default password using this form. In order to ease selecting multiple users, there are javascript buttons for selecting all students, selecting all non-students (staff), selecting all and releasing all.

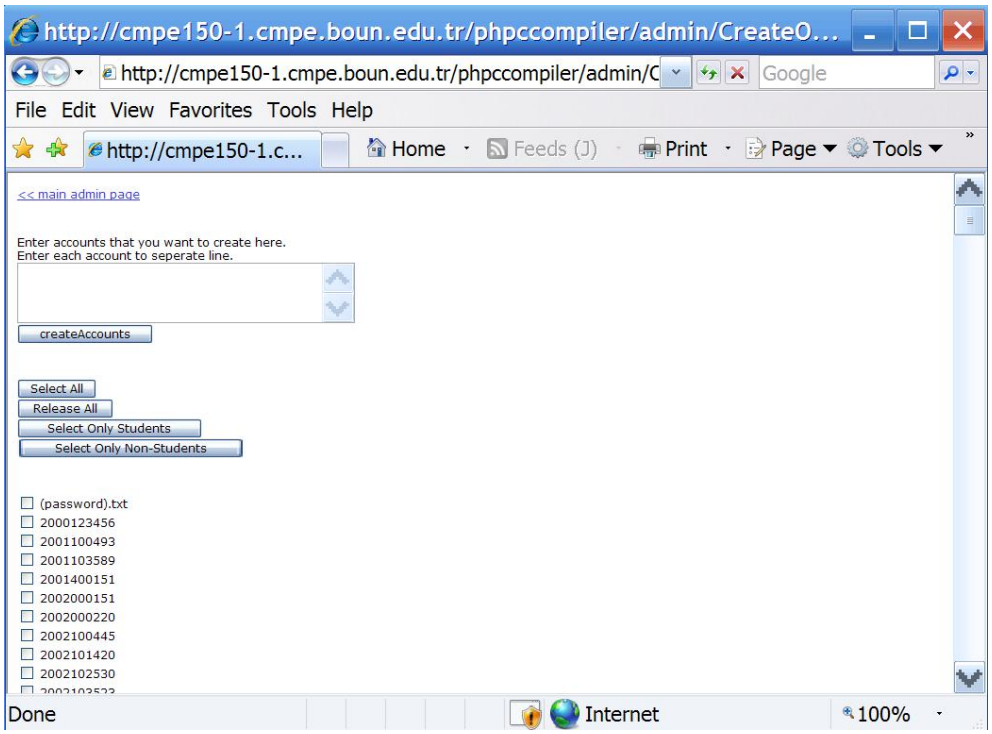


Figure 7.31. Screenshot of CreateOrDeleteAccounts form

7.4. Exam Student

Exam part is normally disabled. During exams, exam section is enabled. Also during exams, home section is disabled to prevent students to look at course notes for cheating. When the exam part is enabled and home part is disabled, on the login page, the student sees only the exam login form (see figure 7.32).

For exam log in, normal password does not work. Students are given exam passwords just before the exam (at the exam classroom) in order to prevent cheating by sharing passwords. Exam passwords are renewed before every exam.

When the student logs in, he sees the list of questions as files similar to file list at home section. Different from home section, student cannot add a file, or cannot rename/delete a file. Only opening is allowed (see figure 7.33).

When the student opens a question, page for answering the question (writing the asked program) is shown. Additional to home part, exam questions usually contain

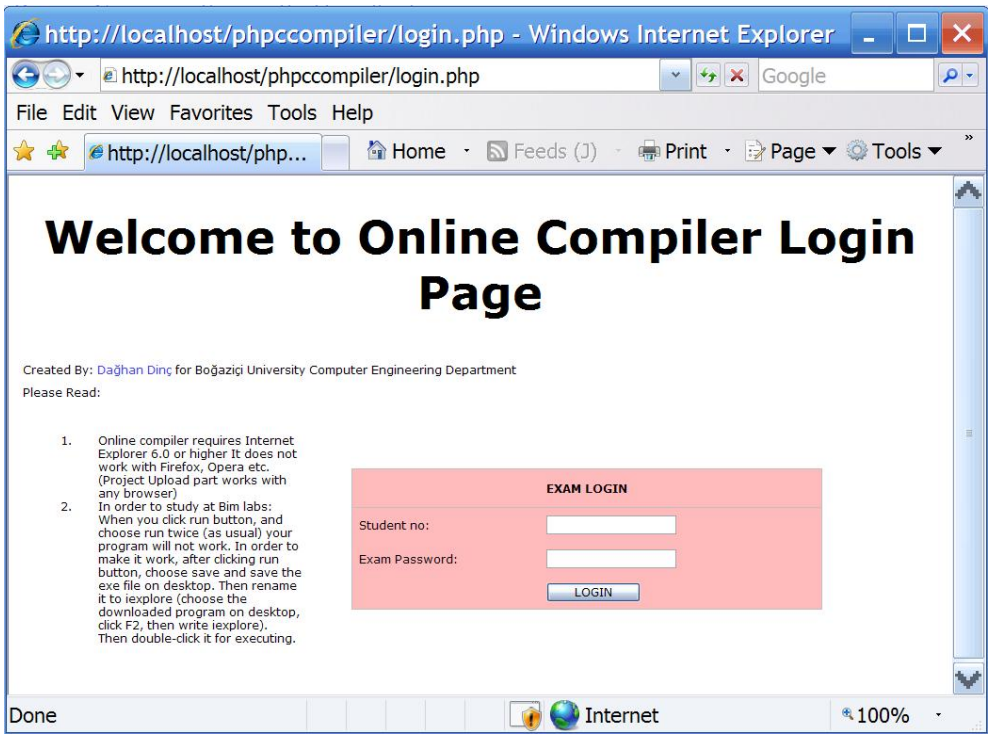


Figure 7.32. Screenshot of login page when exam part is enabled and home part is disabled

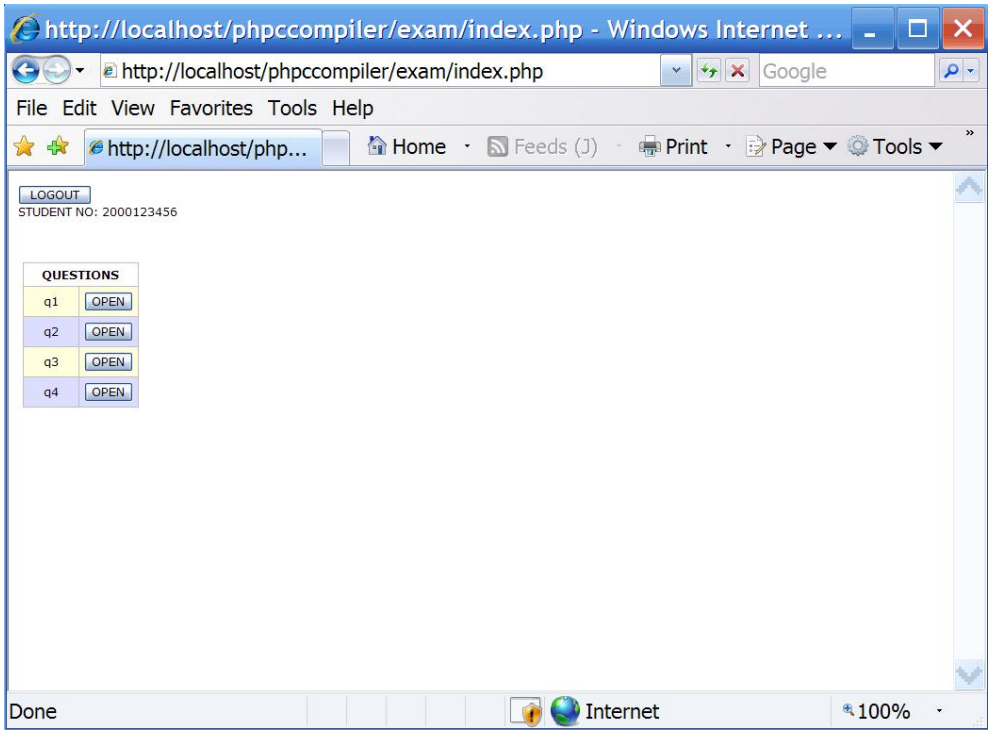


Figure 7.33. Screenshot of exam page

a header and trailer parts containing C code that the student cannot modify. This enables the instructor to determine the syntax and data structures that the student can use. For example, the instructor can give the main function at trailer part calling some functions, and can ask the student to write those functions. Figure 7.34 shows a screenshot of an exam question interface for a student. Here within main, a function with name calcRectangle is called and the student is supposed to write that function between header and trailer code (portion with blue rectangle where the cursor blinks).

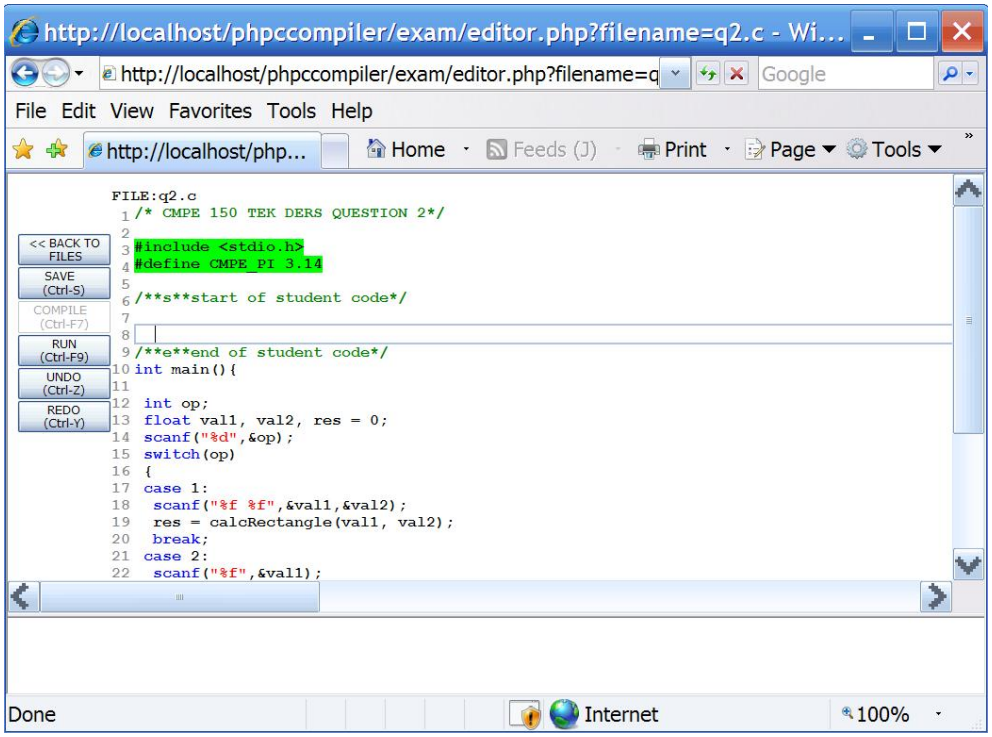


Figure 7.34. Screenshot of an exam question

Reading of exams is not done via our system. It is done with another computer program.

8. FEEDBACK

8.1. Student Recommendations

Several features were changed and some bugs were fixed upon comments received from students through the feedback option. This section lists those changes and fixes.

8.1.1. Comment Colors

As a part of syntax highlighting, comments in C code are colored with a distinguishing color. Previously comment color was light green. One student stated via recommendation form that light green over white background is not readable, a darker color would be better. As a result, we changed comment color to light green to darker green. Note that currently staff or admin users do not have the right to do those alterations. They must be done via altering the source code.

8.1.2. Disappearing \n bug

Due to a character escape bug, once \n's were disappearing. Thanks to in class feedback and reported bugs, we immediately fixed the bug.

8.1.3. File Rename

Creating and deleting file property was available since the beginning. However, students requested rename functionality, thus we added it.

8.1.4. File Sorting

Sorting source files under home section was by creation date. Students wanted sorting via name. We added a javascript plugin which enables sorting based on name, date and size.

8.1.5. Bracket and parenthesis coloring

Parenthesis errors are in top compile errors list almost always. Some students requested bracket coloring. We added coloring pair brackets and parenthesis which hierarchically colors up to seven nested parenthesis with the order of rainbow colors. More inner parenthesis are colored with the last color (purple). In figure 8.1 you can see bracket and parenthesis coloring in action.

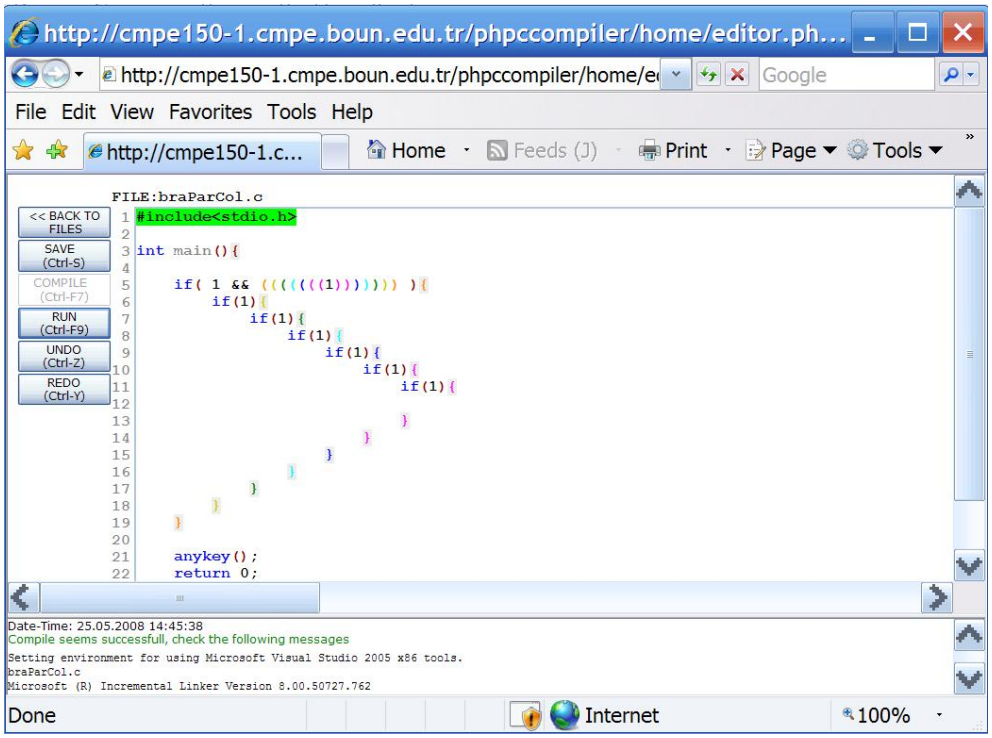


Figure 8.1. Screenshot of bracket and parenthesis coloring in action

8.2. Top Compile Errors

In addition to student feedbacks, top compile errors helps us to shape the course content and in-class warnings given by instructors a lot. Additionally it enables student to see most common errors and avoid them. However, whatever we try, bracket and parenthesis errors are always in top errors list.

9. EVALUATION

9.1. Comparison With Alternatives

An alternative to a web based compiler is a stand alone compiler. For our course, most common alternative to our system is Microsoft Visual Studio for C++. In table 9.1 we prepared a comparison between online compiler and Microsoft Visual Studio for C++ for their teaching and exam circumstances.

9.2. Computer Based Exam vs Paper Exam

Traditionally, programming exams are made on paper and evaluated by teaching staff. Our system enables realization of computer based exams. Computer based exams have advantages and disadvantages over paper exam. In table 9.2 we prepared a comparison.

9.3. Possible Ways to Analyze Usage Data

Although we did not implement reporting functionality yet, in fall 2007 semester, we checked midterm 1 grades vs. number of compiles to check whether there is a correlation or not. Figure 9.1 shows the scatter plot. Correlation of this data is 0.22 which is low. However, it is seen that lower left triangle of the plot is empty. This states 4 things:

1. There are students who worked with online compiler and got high grades.
2. There are students who did not work with online compiler and got low grades.
3. There are students who did not work with online compiler and got high grades.
4. **There are very few student who worked with online compiler and got low grade.**

Table 9.1. Microsoft Visual Studio for C++ vs Online Compiler

Visual Studio	**Online Compiler**
Can work without internet connection	-
-	Can be used from any computer, no necessity for a personal computer
Interface is designed for professional needs	Interface is specifically designed for programming course
Being used in industry	-
Has debugging support	-
-	Has course content
-	Low maintanence cost, especially for lab use
Intolerable delays in distribution and collection of scripts. Exams used to finish 1.5 hours later than the expected time. This was also creating excessive workload on the staff during exams for supervising	More tolerable delays mainly due to exam-password distribution which is done by staff

Table 9.2. Paper Exam vs Computer Exam

Paper Exam	**Computer Exam**
Traditional and proven method	New method
Enables asking of "what does this program outputs" type questions	-
Read by staff. Reading 300 programming exam takes one assistant's 6 weeks with 5-6 hours per day	Read by a computer program with little interaction with supervisor. Takes one assistant's 1-1.5 workdays
Giving partial credit to incomplete programs is possible	Giving partial credit to only sample input-output cases is possible
-	Students have compiler access. They have the ability to edit code with highlight support and view syntax errors. They also have the chance to run and test the program

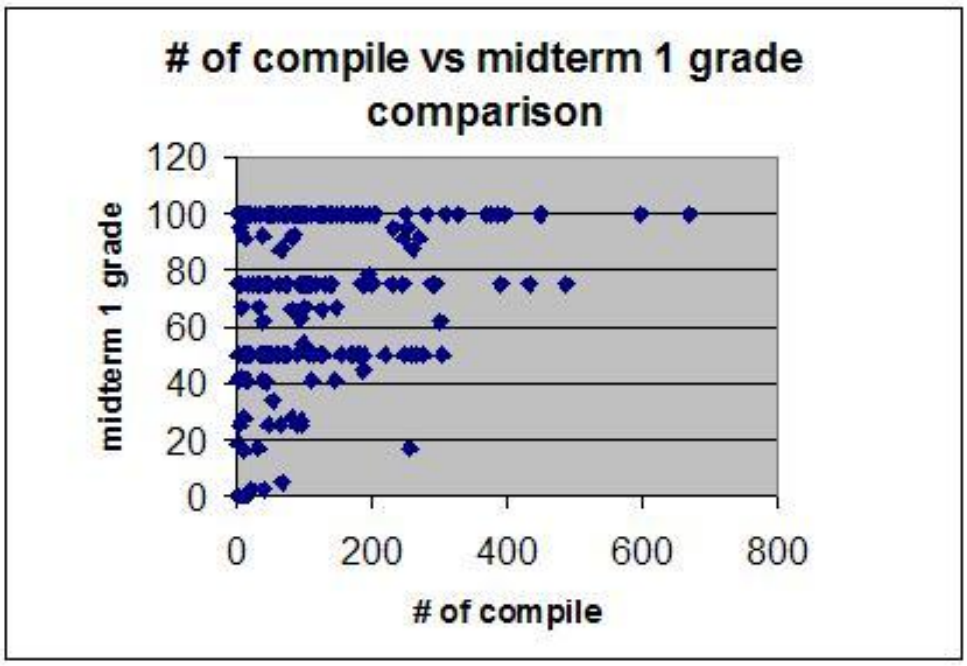


Figure 9.1. Number of compiles vs. Midterm 1 grade

Table 9.3. Results of *compiler use* question of survey. For full table, please refer to appendix D.1

Explanations	**MVS 6.0**	**MVS 7.0**	**MVS 8.0 pro**	**MVS 8.0 exp**	**Online Compiler**	**Borland**	**Other**	
Mean-1 :	0.166	0.125	0	1.083	2.791	0.042	0.416	**4.625**
% use :	3.6	2.7	0	23.4	60.3	0.9	9.0	⇑ **Sum (Mean-1)s**

According to first, second and fourth statements, studying with online compiler seems helping. For the third statement, we guess two possibilities:

1. Those are students who work with an installed compiler such as Microsoft Visual Studio for C++.
2. Some students can get good grades without practicing programming.

Based on our educational experience, second statement is unlikely if the student does not have a technical background in programming but we do not have a proof for that.

9.4. Survey Results

We realized a survey on 24 students asking their compiler preference and their comments about features of online compiler. The number 24 is a small number for precise measurement but that survey is realized just for having some insight.

In table 9.3, there is the results of *compiler use* question. Grading is 1:none, 2:rarely, 3:mediocre, 4:often, 5:only this. While evaluating these grades, we mapped 1-5 to 0-4 since 1 means no use. After, we calculated the mean, sum of means and calculated percentage via $mean/sumOfMeans * 100$. According to that information, online compiler use seems to be around 60%.

Table 9.4. Results of *compiler features* question of survey. For complete table, please refer to appendix D.2

Explanations	**Syntax Highligt**	**Indentation**	**Autocomplete**	**{ closing**	**(closing**	**" closing**	**' closing**	**Error Highlight**	**Onl. Crs. Notes**
(Mean-3)/2 :	*0.69*	*0.26*	*0.23*	*0.40*	*0.21*	*0.50*	*0.48*	*0.57*	*0.70*
% like :	*69*	*26*	*23*	*40*	*21*	*50*	*48*	*57*	*70*

We also asked students to rate the features of online compiler. Results are published on table 9.4. Grading is 1:terrible-remove, 2:bad, 3:mediocre, 4:good, 5:very good. While evaluating these results, we mapped 1-5 to -2-2 since 1 and 2 are negative comments and 3 is 0. After that, we took mean, divided by 2 and multiplied by 100 in order to map to percentage. According to these results, there seems to be no negative feature. However, indentation, auto-complete and parenthesis closing ratings are lower than other features' ratings. We fixed parenthesis closing for nested situations for a better use. We also improved auto-complete to be less annoying by adding quit support via outside mouse click in addition to esc key. For indentation, we use four spaces instead of tab character. Although this is a desired feature for most of the editors, this is not pedagogical since students have tendency to write in the middle of 4 spaces whereas tab character is unsplittable. Sadly, rich content editing of Internet Explorer currently does not support tab characters but we are searching for better solutions.

9.5. Portability

Currently, the system runs on Windows based servers with PHP-Apache-MySQL installation and using Microsoft Visual Studio 2005 C++ express edition as a back-end compiler. System is mostly developed based on needs with incremental steps rather than constructing a complete design and then implementing it. For now, customization of the system is done mostly via editing source codes instead of management forms and interfaces. Due to performance considerations, the system is tightly connected to

technologies which is implemented and altering those technologies will require big effort. For example, MySQL stored procedures are used for database interaction and since the stored procedure syntax differs among databases, it will require to re-write all of them if the database is changed. Additionally windows shell commands are called within PHP code for compiling source files thus the system is currently Windows dependent.

However, since the object oriented design is utilized, "once and only once" principle is followed and PHP language and MySQL database contains many features previously implemented for web development, code footprint is relatively small for a project of this scale (it would probably be doubled or tripled with JSP or ASP). Thus alterations of source will not require much effort.

10. FUTURE WORK

10.1. Teaching Staff Based Improvements

10.1.1. Exam Reader

Currently exams are read by an external software. We are planning to implement a built in exam reader which reads and grades exams instantaneously.

10.1.2. Editable Course Notes

Errors in course notes are currently being fixed by remotely connecting to server machine and altering source files. Although an interface is implemented, it is impractical. We are planning to integrate a wiki engine with our online compiler engine thus utilizing wiki editing interface for editing course notes.

10.1.3. Improved Feedback Reporting

Currently, compile errors, page visits etc. are being logged. However, the only implemented interface for feedback from those is top compile errors of the week.

We are planning to implement a reporting interface for data with graph plots, continuous graphical feedback on system use. Analysis of the use can be presented in graphical format to provide insights about how the course is going which is would be very helpful for Staff users. Additionally, feedback to students on how they are progressing would be motivating.

Additionally, we want to add feedback features enabling student modeling in order to keep track of student levels and assist them with various kinds of support depending on how they are progressing.

10.1.4. Improved Admin Interface

Currently, the only additional capability of admin user is creating and deleting user accounts. Although editable course notes feature will be controlled by admin user, modification system configuration parameters such as enabling-disabling exam part still requires manually editing source files.

10.1.5. Example Question and Quiz Creation Interface

Currently, there exists written questions on the web page for studying. We also put archive questions on the web. However, in order to enable addition of examples by staff or even by students, there should be an interface for creating questions which are shown to students with compiler interface for solving. There also should be an interface for reading and grading those questions.

There also can be quizzes creatable by instructors with multiple choice or true-false questions for measuring students knowledge beyond coding.

There also can be an interface for test exams, listing questions, putting time limit and grading questions.

10.2. Unimplemented Requests From Students

10.2.1. Turkish Character Support

Unline linux distributions, Windows operating system's native file naming is not utf8 [67] based. Therefore, file names including turkish characters (ç,Ç,ğ,Ğ,ı,İ,ö,Ö,ş,Ş,ü,Ü) are problematic. This is currently left as an issue planning to be solved later.

10.2.2. Link Error Bug

Sometimes, unexpectedly compiler gives a link error. When the link error is given, it sometimes become a failed compilation, sometimes successful. Although we think that this is a problem due to a race condition within file system, we could not diagnose it yet. We are planning to solve this issue.

10.2.3. Cross Browser Support

Online compiler requires Windows Internet Explorer. There are many students using other browsers especially Firefox [68]. Although windows dependency is going to continue and Internet Explorer is standard with Windows (see section 5.1.2), still cross browser compatibility is a desired accessibility feature for most of the systems.

10.2.4. Code Folding

Many editors support code folding and unfolding via showing a plus (+) sign at the beginning of code blocks or functions etc. This is a very helpful feature especially while writing long programs. We are considering to implement this feature but it is currently very difficult and thus not planned soon.

10.2.5. Offline Online Compiler

Some students requested an offline version for online compiler in order to study with online compiler instead of Visual Studio when not connected to the Internet. We highly appreciate this request since it shows that even students having their own computers desire to use online compiler but currently this is very difficult to implement since online compiler depends on many other software such as Apache web server, PHP engine, MySQL database and creating a full install package is far beyond our available labor.

10.2.6. Mass Deletion of Source Files

Although there exists delete support for files, students request mass deletion feature. It is doable via HTML checkboxes similar to a mass mail deletion interface which can be found on many mail clients such as Yahoo or Gmail. We are planning to implement this but we are also concerned about not increasing complexity of the home interface.

10.3. Our Plans

10.3.1. Improved User Interface and User Experience

Current user interface is kept as plain as possible in order to focus on simplicity and performance. However, it can be improved while keeping or enhancing simplicity and performance.

Additionally, currently course web page and online compiler are two separate entities. Online compiler interface can be enhanced to include the course web page thus enabling integration of system components to include syllabus, exams, etc. within one point location to visit.

Furthermore, addition multimedia content such as voice or video content as course materials can enhance user experience.

Last but not least, default exe output written in C pops up with a DOS prompt containing white text on black background. In contrast, today's common interface is paper-like interface in which dark text on light background is used. DOS prompt can be modified or altered to ensure a better visualization.

10.3.2. Social Networks

In order to enable healthy communication paths between students for the purpose of increased learning, we are planning to add social network mechanisms into our system. Today, the Internet highly benefits from online community aspects thus sharing between students will enable the richness and quality of programming education. However, caution must be exercised with respect to to policies of cheating.

10.3.3. Code Tidy

Many students have the tendency to write spaghetti codes lacking proper indentation. We are planning to add a button, which tidies the C code in an appropriate industry standard fashion. This both will help students while coding and also will stand as an example for good indentation style.

10.3.4. Debugging Support

We are planning to add a simple interface for step by step execution and variable monitoring. This will both help the students while coding and it will help the instructors while lecturing for showing the algorithmic steps.

10.3.5. Automatic Evaluation Of Code

Similar to how exams are graded, small assignments or even general code can be evaluated. Certain properties of the code, like comments, and standards can also be evaluated. This could improve quality issues.

10.3.6. Stress Test

After fixing the session-exec hanging bug (see section 6.5), the system seems to work without server load. However, realizing a stress test would enable to realize the current limitations of the system.

10.3.7. Usability Test

While designing the interface of the system, we inspired from existing web systems. However, a comprehensive usability test would reveal the deficiencies in the interface thus would enable us to improve the user-friendliness of the system.

10.3.8. Online Compiler For The World

We are planning to create a public version of online compiler for every people to sign up and use. This both will advertise our department all will supply us lots of precious data for software and data mining research. We are currently waiting to have a spare server for this.

11. CONCLUSIONS

In this study, we developed a novel web based programming instruction environment. Our motivation were real necessities of a given "Introduction to programming course". Our system is now deployed and successfully in use.

We observed positive effects on students and staff improving the quality of the given course. Additionally stated requirements are mostly satisfied and improvements are continual.

Although we developed our system based on our needs and currently using this system for our course, this system have the potential to be used under any appropriate circumstances.

APPENDIX A: Stored Procedure Sources

A.1. bug_report_insert:

```
CREATE DEFINER=‘root‘@‘localhost‘
PROCEDURE ‘bug_report_insert‘(
        param_student_id text,
        param_subject text,
        param_explanation text,
        param_how_to_reproduce text
)
BEGIN
        insert into
                ‘cmpe150onlinecompiler‘.‘bug_report‘(
                        student_id,
                        date_time,
                        subject_,
                        explanation,
                        how_to_reproduce
                )
        values(
                param_student_id,
                now(),
                param_subject,
                param_explanation,
                param_how_to_reproduce
        );
```

A.2. error_explanation_toplist:

```
CREATE DEFINER=‘root‘@‘localhost‘
```

```
PROCEDURE ‘error_explanation_toplist‘()
BEGIN
  select
          error_log.error_explanation,
          count(error_log.error_explanation) as frequency,
          error_hint.hint
  from
          ‘cmpe150onlinecompiler‘.‘error_log‘ left join
          ‘cmpe150onlinecompiler‘.‘error_hint‘ on
          ‘cmpe150onlinecompiler‘.‘error_log‘.error_explanation =
            ‘cmpe150onlinecompiler‘.‘error_hint‘.error_explanation
  where
          date_time > now()- interval 7 day
  group by
          error_log.error_explanation
  order by
          frequency desc
  limit 10;
END$$
```

A.3. error_explanation_toplist_by_student_no:

```
CREATE DEFINER=‘root‘@‘localhost‘
PROCEDURE ‘error_explanation_toplist_by_student_no‘(
        param_student_id text
)
BEGIN
  select
          error_log.error_explanation,
          count(error_log.error_explanation) as frequency,
          error_hint.hint
```

```
  from
          `cmpe150onlinecompiler`.`error_log` left join
          `cmpe150onlinecompiler`.`error_hint` on
          `cmpe150onlinecompiler`.`error_log`.error_explanation =
           `cmpe150onlinecompiler`.`error_hint`.error_explanation
  where
          date_time > now()- interval 7 day and
          student_id = param_student_id
  group by
          error_log.error_explanation
  order by
          frequency desc
  limit 10;
END$$
```

A.4. error_hint_insert:

```
CREATE DEFINER=`root`@`localhost`
PROCEDURE `error_hint_insert`(
        param_error_explanation text,
        param_hint text
)
BEGIN
        insert into
                error_hint(
                        error_explanation,
                        hint)
        values(
                trim(param_error_explanation),
                trim(param_hint)
        );
```

```
END$$
```

A.5. error_log_insert:

```
CREATE DEFINER=‘root‘@‘localhost‘
PROCEDURE ‘error_log_insert‘(
       param_student_id text,
       param_part_id bigint,
       param_error_code bigint,
       param_error_file_name text,
       param_error_line_number bigint,
       param_error_explanation text

)
BEGIN
  insert into
         ‘cmpe150onlinecompiler‘.‘error_log‘(
                student_id,
                part_id,
                date_time,
                error_code,
                error_file_name,
                error_line_number,
                error_explanation)
  values(
         param_student_id,
         param_part_id,
         now(),
         param_error_code,
         param_error_file_name,
         param_error_line_number,
```

```
         trim(param_error_explanation)
  );
END$$
```

A.6. file_action_log_insert:

```
CREATE DEFINER=‘root‘@‘localhost‘
PROCEDURE ‘file_action_log_insert‘(
        param_student_id text,
        param_part_name text,
        param_action_name text,
        param_file_name text,
        param_remote_addr text
)
BEGIN
        declare var_part_id bigint;
        declare var_action_id bigint;
        #set var_part_id=3;
        select
          part_id into var_part_id
        from
          part
        where
          name=param_part_name;
        select
          action_id into var_action_id
        from
          action
        where
          name=param_action_name;
```

```
    insert into
         `cmpe150onlinecompiler`.`file_action_log`(
                student_id,
                part_id,
                action_id,
                date_time,
                file_name,
                remote_addr)
  values(
         param_student_id,
         var_part_id,
         var_action_id,
         now(),
         param_file_name,
         param_remote_addr
  );
END$$
```

A.7. page_visit_log_insert:

```
CREATE DEFINER=`root`@`localhost`
PROCEDURE `page_visit_log_insert`(
        param_student_id text,
        param_url text,
        param_remote_addr text)
BEGIN
        insert into
         `cmpe150onlinecompiler`.`page_visit_log`(
                student_id,
                date_time,
                url,
```

```
                remote_addr)
  values(
         param_student_id,
         now(),
         param_url,
         param_remote_addr
  );
END$$
```

A.8. part_select:

```
CREATE DEFINER=‘root‘@‘localhost‘
PROCEDURE ‘part_select‘(part_id_base int,name_example text)
BEGIN
  select
    *
  from
    ‘cmpe150onlinecompiler‘.part
  where
    part_id>part_id_base or
    name=name_example;
END$$
```

A.9. recommendation_insert:

```
CREATE DEFINER=‘root‘@‘localhost‘
PROCEDURE ‘recommendation_insert‘(
 param_student_id text,
        param_subject text,
        param_explanation text
)
```

```
BEGIN
        insert into
          `cmpe150onlinecompiler`.`recommendation`(
            student_id,
            date_time,
            subject_,
            explanation
            )
        values(
          param_student_id,
          now(),
          param_subject,
          param_explanation
        );

END$$
```

APPENDIX B: Javascript Sources

B.1. Automatic Bracket and Quote Closing

```
var strNearCursor = field.cursorManager.getStringNearCursor(-2,1);
var char2BeforeCursor = strNearCursor.charAt(0);
var charBeforeCursor = strNearCursor.charAt(1);
var charAfterCursor = strNearCursor.charAt(2);

if(char2BeforeCursor == '\\') return; // omit escapes

// delete user close (),",') if necessary (justAutoClosed):
if(
justAutoClosed &&
(charBeforeCursor == '"' && charAfterCursor == '"' ||
 charBeforeCursor == "'" && charAfterCursor == "'" ||
 charBeforeCursor == ")" && charAfterCursor == ")" ||
 charBeforeCursor == "]" && charAfterCursor == "]")
){
field.cursorManager.deleteStringNearCursor(0,1);
justAutoClosed=false;
}
else{ // AUTOCLOSE:
justAutoClosed=true;
switch(charBeforeCursor){
case '(':
field.cursorManager.insertAfterCursor(')');
break;
case '[':
field.cursorManager.insertAfterCursor(']');
break;
```

```
case '"':
case "'":
field.cursorManager.insertAfterCursor(charBeforeCursor);
break;
default:
justAutoClosed=false;
break;
}
}
```

APPENDIX C: PHP Sources

C.1. PHP Compile Error Parsing

```
$error_line_parts = explode(":",$error_line,3);
$file_and_line_number = $error_line_parts[0]; // need further parsing
$error_code_portion = $error_line_parts[1]; // need further parsing
$error_explanation = $error_line_parts[2];

$error_code = preg_replace('/.*?(\d+).*/','$1', $error_code_portion);
$error_file_name = preg_replace('/(\w+\.\w+).*/','$1',$file_and_line_number);
$error_line_number = preg_replace('/\w+\.\w+\((\d+)\).*/','$1',$file_and_line_num

if($error_line_number != 0 && preg_match('/\d+/',$error_line_number)){
$error_line_numbers[] = $error_line_number;
$error_explanations[] = $error_explanation;
}
```

APPENDIX D: Survey Results

D.1. Compiler Use

For full table of compile use survey, please refer to table D.1

D.2. CompilerFeatures

For full table of compile features survey, please refer to table D.2

Table D.1. Results of *compiler use* question of survey

Explanations	**MVS 6.0**	**MVS 7.0**	**MVS 8.0 pro**	**MVS 8.0 exp**	**Online Compiler**	**Borland**	**Other**	
	1	1	1	3	4	1	1	
Grading	1	1	1	3	4	1	1	
1:none	1	1	1	3	4	1	1	
2:rarely	1	1	1	2	3	1	1	
3:mediocre	1	1	1	3	4	1	1	
4:often	1	1	1	3	3	1	1	
5:only this	1	1	1	2	4	1	1	
	1	1	1	2	4	1	1	
	1	1	1	1	5	1	1	
	1	1	1	3	4	1	4	**pelles-C**
	1	1	1	2	4	1	1	
	1	1	1	2	4	1	1	
	1	1	1	2	5	1	1	
	1	1	1	1	1	1	5	**mvs 5.0**
	1	1	1	2	4	1	1	
	3	1	1	4	3	2	1	
	1	1	1	2	5	1	1	
	1	1	1	2	1	1	1	
	1	1	1	2	4	1	1	
	1	1	1	1	5	1	1	
	1	1	1	1	5	1	1	
	1	1	1	2	5	1	1	
	3	1	1	1	4	1	4	**mvs 2008 beta**
	1	4	1	1	2	1	1	
Mean-1 :	0.166	0.125	0	1.083	2.791	0.042	0.416	**4.625**
% use :	3.6	2.7	0	23.4	60.3	0.9	9.0	⇑ **Sum (Mean-1)s**

Table D.2. Results of *compiler features* question of survey

Explanations	**Syntax Highligt**	**Indentation**	**Autocomplete**	**{ closing**	**(closing**	**" closing**	**' closing**	**Error Highlight**	**Onl. Crs. Notes**
	5	4	4	4	4	5	5	5	5
Grading	4	4	5	4	3	4	4	2	2
1:terrible	4	4	4	2	4	4	4	4	3
2:bad	4	3	4	4	4	4	4	2	3
3:mediocre	4	4	4	4	3	4	4	4	5
4:good	4	4	1	5	5	5	5	5	4
5:very good	4	3	3	3	3	3	4	4	4
	5	5	5	5	5	5	5	5	5
	5	3	4	4	4	4	3	5	4
	4	4	5	4	2	3	4	4	5
	5	4	3	4	4	4	4		
	5	4	3	5	2	5	5	4	5
	4	4	5	4	5	5	5	4	5
	4	2	3	1	1	1	1	3	5
	5	3	4	5	2	4	4	5	5
	4	2	3	5	5	5	4	3	4
	4	3	3	2	4	5	4	5	5
	4	4	2	5	5	5	5	5	5
	4	4	4	5	3	5	5	5	5
	5	5	5	5	5	5	5	5	4
	5	3	1	3	3	3	3	2	4
	4		5	4	4	4	4	5	4
	4	4	2	3	1	3	3	4	5
	5	1	1	1	1	1	1	5	5
(Mean-3)/2 :	*0.69*	*0.26*	*0.23*	*0.40*	*0.21*	*0.50*	*0.48*	*0.57*	*0.70*
% like :	*69*	*26*	*23*	*40*	*21*	*50*	*48*	*57*	*70*

REFERENCES

1. DOĞU, A. H., "THE LOCAL ANALAYSIS OF THE LEVEL OF USING BASIC KNOWLEDGE TECHNOLOGIES OF THE STUDENTS ATTEND TO THE UNIVERSITY", *AKADEMİK BİLİŞİM 2008*, Akademik Bilişim 2008, 2008.

2. Microsoft, "Microsoft Visual Studio 2005 C++ Express Edition", `http://msdn.microsoft.com/en-us/visualc/default.aspx`, 05 2008.

3. Dinç, D., "Cmpe150", `http://www.cmpe.boun.edu.tr/courses/cmpe150/spring2008/`, 04 2008.

4. Microsoft, "Microsoft Windows", `http://www.microsoft.com/windows/`, 04 2008.

5. Microsoft, "Internet Explorer: Home Page", `http://www.microsoft.com/windows/products/winfamily/ie/default.mspx`, 04 2008.

6. Wikipedia-Community, "World Wide Web - Wikipedia, the free encyclopedia", `http://en.wikipedia.org/wiki/World\_Wide\_Web/`, 04 2008.

7. W3C, "About The World Wide Web", `http://www.w3.org/WWW/`, 03 2008.

8. Wikipedia-Community, "Web application - Wikipedia, the free encyclopedia", `http://en.wikipedia.org/wiki/Web\_application`, 04 2008.

9. Wikipedia-Community, "ILOVEYOU - Wikipedia, the free encyclopedia", `http://en.wikipedia.org/wiki/ILOVEYOU`, 06 2008.

10. Yahoo!, "Yahoo! Mail: The best web-based email!", `http://mail.yahoo.com`.

11. Microsoft, "Microsoft Corporation", `http://www.hotmail.com`.

12. Google, “Gmail: Email from Google”, http://www.gmail.com.

13. BÜ-BİM, “Boğaziçi University Webmail”, https://webmail.boun.edu.tr/, 04 2008.

14. TRT, “TRT/TELEVİZYON”, http://www.trt.com.tr/wwwtrt/tv.aspx, 04 2008.

15. TRT, “TRT/RADYO”, http://www.trt.com.tr/wwwtrt/radyo.aspx, 04 2008.

16. Youtube, “Youtube - Broadcast Yourself”, http://www.youtube.com/, 06 2008.

17. Metacafe, “Best Videos and Funny Movies”, http://www.metacafe.com/, 06 2008.

18. Joffe, B., “Canvascape - 3D walker”, http://www.abrahamjoffe.com.au/ben/canvascape/, 06 2008.

19. Uselesspickles, “Real Time 3D in Javascript”, http://www.uselesspickles.com/triangles/demo.html, 06 2008.

20. Salesforce, “CRM Software On Demand, Online CRM Solutions - salesforce.com”, http://www.salesforce.com/, 04 2008.

21. University, S. I., “http://roboti.cs.siue.edu/eyebot/68compiler/”, http://roboti.cs.siue.edu/eyebot/68compiler/, 05 2008.

22. Tschalär, R., “JXXX Compiler Service”, http://www.innovation.ch/java/java_compile.html, 05 2008.

23. Delorie, D., “DJGPP Public Access Cross-Compiler”, http://www.delorie.com/djgpp/compile/, 05 2008.

24. Data, R., “HTML Tutorial”, http://www.w3schools.com/html/default.asp, 05 2008.

25. Data, R., "Javascript Tutorial", `http://www.w3schools.com/js/default.asp`, 05 2008.

26. Haverbeke, M., "In-browser code editing", `http://marijn.haverbeke.nl/codemirror/`.

27. Haverbeke, M., "Eloquent JavaScript – interactive tutorial", `http://eloquentjavascript.net/`, 05 2008.

28. Kelleher, C. and R. Pausch, "Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers", *ACM Comput. Surv.*, Vol. 37, No. 2, pp. 83–137, 2005.

29. Wirth, N., "The programming language Pascal", pp. 121–148, 2002.

30. Irons, E. T., "A syntax directed compiler for ALGOL 60", *Commun. ACM*, Vol. 4, No. 1, pp. 51–55, 1961.

31. Cooper, S., W. Dann, and R. Pausch, "Alice: a 3-D tool for introductory programming concepts", *CCSC '00: Proceedings of the fifth annual CCSC northeastern conference on The journal of computing in small colleges*, pp. 107–116, Consortium for Computing Sciences in Colleges, , USA, 2000.

32. Felleisen, M., R. B. Findler, M. Flatt, and S. Krishnamurthi, *How to design programs: an introduction to programming and computing*, MIT Press, Cambridge, MA, USA, 2001.

33. Sussman, G. J. and J. Guy L. Steele, "An Interpreter for Extended Lambda Calculus", Technical report, Cambridge, MA, USA, 1975.

34. Findler, R. B., J. Clements, C. Flanagan, M. Flatt, S. Krishnamurthi, P. Steckler, and M. Felleisen, "DrScheme: a programming environment for Scheme", *J. Funct. Program.*, Vol. 12, No. 2, pp. 159–182, 2002.

35. Kernighan, B. W. and D. M. Ritchie, *The C programming language*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1978.

36. Aran, O., "Computer Engineering Department / Boğasiçi University", `http://www.cmpe.boun.edu.tr/courses/cmpe150/`, 05 2008.

37. Wikipedia-Community, "Syntax Highlighting - Wikipedia, the free encyclopedia", `http://en.wikipedia.org/wiki/Syntax\_highlighting`, 04 2008.

38. Wikipedia-Community, "Indent style - Wikipedia, the free encyclopedia", `http://en.wikipedia.org/wiki/Indent\_style`, 04 2008.

39. Wikipedia-Community, "Undo - Wikipedia, the free encyclopedia", `http://en.wikipedia.org/wiki/Undo`, 04 2008.

40. Wikipedia-Community, "Line number - Wikipedia, the free encyclopedia", `http://en.wikipedia.org/wiki/Line\_number`, 04 2008.

41. Wikipedia-Community, "Autocomplete - Wikipedia, the free encyclopedia", `http://en.wikipedia.org/wiki/Autocomplete`, 04 2008.

42. Wikipedia-Community, "CamelCase - Wikipedia, the free encyclopedia", `http://en.wikipedia.org/wiki/CamelCase`, 04 2008.

43. Microsoft, "Microsoft Visual Studio 2005 C++ Express Edition", `http://go.microsoft.com/fwlink/?linkid=57034`, 04 2008.

44. PHP-Group, "PHP: Hypertext Preprocessor", `http://www.php.net/`, 04 2008.

45. MySQL, "MySQL :: The world's most popular open source database", `http://www.mysql.com/`, 04 2008.

46. Apache-Group, "Welcome! - The Apache Software Foundation", `http://www.apache.org/`, 04 2008.

47. Bourdon, R., "Install PHP 5 Apache MySQL on Windows : WampServer", http://www.en.wampserver.com/, 04 2008.

48. Incorporated, A. S., "Adobe - Adobe Flash Player", http://www.adobe.com/products/flashplayer/, 04 2008.

49. Sun-Microsystems, "Adobe - Adobe Flash Player", http://java.sun.com/applets/, 04 2008.

50. Wikipedia-Community, "Multitier architecture - Wikipedia, the free encyclopedia", http://en.wikipedia.org/wiki/Three-tier_(computing), 04 2008.

51. Üsküdarlı, S., "Eng101", http://elele.cmpe.boun.edu.tr/wiki/bin/view/Eng101, 04 2008.

52. Wikipedia-Community, "Forms in HTML documents", http://www.w3.org/TR/html4/interact/forms.html\#h-17.7, 04 2008.

53. Wikipedia-Community, "HTML - Wikipedia, the free encyclopedia", http://en.wikipedia.org/wiki/HTML, 04 2008.

54. Wikipedia-Community, "Cascading Style Sheets - Wikipedia, the free encyclopedia", http://en.wikipedia.org/wiki/Cascading_Style_Sheets, 04 2008.

55. Wikipedia-Community, "Javascript - Wikipedia, the free encyclopedia", http://en.wikipedia.org/wiki/JavaScript, 04 2008.

56. Google, "Google Code", http://code.google.com/, 04 2008.

57. Gorbatchev, A., "syntaxhighlighter - Google Code", http://code.google.com/p/syntaxhighlighter/, 04 2008.

58. Wikipedia-Community, "Regular Expression - Wikipedia, the free encyclopedia", http://en.wikipedia.org/wiki/Regular_expression, 04 2008.

59. Muze, “Helene - A syntax highlighting text editor in javascript.”, http://helene.muze.nl/, 04 2008.

60. Alfred V. Aho, R. S. and J. D. Ullman, *Compilers: Principles Techniques, and Tools*, Addison-Wesley, 1986.

61. GCC-team, “GCC, the GNU Compiler Collection - GNU Project - Free Software Foundation (FSF)”, http://gcc.gnu.org/, 04 2008.

62. PHP-Community, “PHP Bugs: #22526: session_start/popen hang”, http://bugs.php.net/bug.php?id=22526, 04 2008.

63. PHP-Group, “PHP: date - Manual”, http://tr2.php.net/date, 04 2008.

64. PHP-Group, “PHP: $_COOKIE - Manual”, http://tr2.php.net/manual/en/reserved.variables.cookies.php, 04 2008.

65. Elert, G., “hypertextbook.com”, http://hypertextbook.com/, 05 2008.

66. Microsoft, “Excel Giriş Sayfası - Microsoft Office Online”, http://office.microsoft.com/tr-tr/excel/default.aspx, 05 2008.

67. utf 8.com, “UTF-8 and Unicode Standards”, http://www.utf-8.com/, 05 2008.

68. Mozilla, “Firefox web browser — Faster, more secure, & customizable”, http://www.mozilla.com/firefox/, 05 2008.