NTNU
Norwegian University of
Science and Technology

Experiments towards digital exam with auto-grading in C++ programming courses

Johannes Omberg Lier
Thea Christine Mathisen

Master of Science in Computer Science
Submission date: June 2016
Supervisor: Lasse Natvig, IDI
Co-supervisor: Guttorm Sindre, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Statement

Climbing Mont Blanc (CMB) is a system for evaluation of programs executed on modern multicores. CMB evaluates both performance and energy efficiency, but in this project we will focus on how CMB can be combined with other means such as e.g. multiple choice questions in a future digital exam with auto-grading in a C++ programming course, or similar courses.

The long term idea for this and succeeding projects is to use multiple choice questions, or similar, that are possible to grade automatically, and a set of problems on the CMB system to make up a complete possible exam in courses like TDT4102 – Procedural and Object-Oriented Programming at the Norwegian University of Science and Technology (NTNU). The main goal of this project is to develop and execute a voluntary midterm test to be used in TDT4102 in the spring semester 2016, in order to find strengths and weaknesses of this kind of digital examination approach. Three of the more relevant aspects are student learning, grading fairness, and cost of grading exams. This master thesis project builds on the project work by the same two students finished in December 2015. That project was a feasibility study for this master thesis project.

The following elements are included in the project:

a) Implement, organize, and lead an experimental midterm test in TDT4102 in collaboration with its course staff and responsible teacher.

b) Implement a solution for extracting the complete results from the tests (both multiple choice and CMB) to data in an Excel spreadsheet or a similar format making automatic grading possible. Set up or program the receiving spreadsheet to facilitate the auto-grading.

c) Identifying alternative question types that are suited for an autograding system for C++ and other programming languages. Existing multiple choice questions for C++ courses should be looked for. “License” rules for use by others must be considered.

d) Discuss which of these question types will best fit into the C++ course at NTNU, and how they complement what can be covered by the CMB system.

e) Develop or find some multiple choice questions, or similar, that can be used in the midterm tests.

f) Develop or find, and upload some C++ problems to the CMB system to be used in the tests in the C++ course.

g) Outline ideas for a realistic 5-year track making the vision “digital exam with auto-grading” to a reality at NTNU.

If time, the students should also:

i) Discuss how the tests run in the spring semester could be correlated with other more traditional exam-like tests with the purpose of assessing the goodness of our approach.

j) Identify how user interface improvements can be implemented to better reach the overall goal, and describe these as prioritized proposals to the CMB-project.

Abstract

Climbing Mont Blanc (CMB) is an online judge system especially suited for evaluating energy efficient programming solutions currently in development by a team of professors and master students at the Norwegian University of Science and Technology (NTNU). This project seeks to further gain knowledge and experience in the field of automatic assessment of programming problems, by identifying various question types suitable for automatic and reliable assessment. We have found that most of the question types commonly found on learning management systems, such as itslearning and Blackboard, are suitable in an exam situation. As for CMB, we have found that *complete the code* and *fix the code* problem types can be considered suitable in most cases because the problem creator has much more control over the program flow. Problems that require complete implementations and topic specific programming problems are also suitable but in many cases measures needs to be done to maintain validity of the evaluation.

To reach a conclusion on the feasibility of digital examination at NTNU, we have conducted a midterm experiment with the students in the TDT4102 course, which has been described in detail. Elements such as the execution, result extraction and anonymization of the respondents are covered. The latter was done using a representative from the CMB team who had no formal connection to the experiment in itself.

All results from the midterm test were statistically analyzed by testing a set of null hypotheses that were defined early in the project. This analysis confirmed what we originally believed; the use of familiar digital tools such as the CMB system is beneficial when solving programming problems, compared to writing code by hand.

Finally, we conclude that the midterm experiment was successful, due to the results from both the midterm and questionnaire indicated that the system has potential. We also suggest a plan for the project's continuation in the years ahead.

Sammendrag

Climbing Mont Blanc (CMB) er et online judge system, spesielt egnet for å evaluere energi-effektiv programvare, som for tiden utvikles av et team med professorer og studenter på Norges teknisk-naturvitenskapelige universitet (NTNU). Dette masterprosjektet har som mål å opparbeide erfaring og innsikt om digital og automatisk evaluering av programmeringsoppgaver; for å gjøre dette skal vi identifisere et spekter av ulike spørsmålstyper og programmeringsproblemer som egner seg for automatisk evaluering. Vi har observert at de fleste spørsmålstypene man finner i e-læringssystemer, som itslearning og Blackboard, passer godt inn i en eksamenskontekst. På CMB ser vi at de mest egnede programmeringsoppgavetypene er typiske *fullfør kodesnutten*- og *fiks kodesnutten*-problemer, der oppgaveforfatteren har større kontroll hva gjelder programflyt og kodestruktur. Når det gjelder oppgavetyper som krever komplett implementasjon fra bunnen av, samt emne-spesifikke programmeringsproblemer, må en i flere tilfeller gjøre tilpasninger for å opprettholde validiteten til evalueringen.

For å konkludere hvorvidt en digital eksamensform er mulig å gjennomføre på NTNU har vi planlagt og organisert et eksperiment som bestod av en midtsemeserprøve for studenter i faget TDT4102. Vi har beskrevet aspekter ved eksperimentet, inkludert planlegging, gjennomføring og anonymisering av resultatene i ettertid. I sistnevnte prosess fikk vi god hjelp av en representant fra CMB-teamet uten noen formell tilknytning til vårt prosjekt.

Alle resultatene fra midtsemesterprøven ble analysert vha. nullhypotesetesting av hypoteser vi definerte tidlig i prosjektet. Denne analysen bekreftet vår teori om at det å bruke et digitalt programmeringsverktøy er fordelaktig når man skal løse programmeringsoppgaver, særlig sammenliknet med det å kode med penn og papir.

Til slutt konkluderer vi med at eksperimentet vårt var en suksess, da resultatene fra både midtsemesterprøven og spørreundersøkelsen indikerte at systemet har et stort potensial. Vi har også foreslått en plan for prosjektets fremtid.

Acknowledgements

First of all, we are exceptionally grateful to our supervisors, Lasse Natvig and Guttorm Sindre, who have provided us with invaluable guidance and support throughout the past year. Furthermore, a big thanks to the course staff of TDT4102 in the spring semester of 2016 for helping us out with our experiment. Finally, we would like to thank Sindre Magnussen, who has been an important supporter when we needed help with anything CMB-related.

Contents

List of Figures

List of Tables

List of Listings

List of Acronyms

BYOD	Bring Your Own Device.
CMB	Climbing Mont Blanc.
CPU	Central Processing Unit.
EDP	Energy Delay Product.
FEIDE	Felles Elektronisk IDEntitet.
FS	Felles Studentsystem.
ICPC	ACM International Collegiate Programming Contest.
IDI	the Department of Computer and Information Science.
LMS	Learning Management System.
MC	Multiple Choice.
MR	Multiple Response.
NTNU	Norwegian University of Science and Technology.
SEB	Safe Exam Browser.
SSO	Single Sign-On.
TA	Teaching Assistant.

Chapter 1
Introduction

From the first "`Hello, world!`" when learning the first steps of programming, to creating a personal website or developing advanced industrial software, the process of producing, compiling, and running program code is ultimately a purely digital one. However, students who take introductory programming courses at Norwegian University of Science and Technology (NTNU) are recommended to practice coding by hand to prepare for the final course evaluation, which incidentally requires them to produce hand-written code. If the university were to introduce digital examination using the Climbing Mont Blanc (CMB) system, an online judge system developed at NTNU – or any other digital assessment system for that matter – their programming courses could evenly mirror industry standards, and the outdated concept of programming by hand could perhaps retire.

In this thesis, we aim to contribute to this important development of programming assessment by conducting an experiment involving real students and digital examination.

1.1 Motivation

The motivation for this thesis stems not only from the desire to digitalize the examination process of programming courses at NTNU, but also the cost and potential impreciseness related to grading these exams [Har05]. Table 1.1 illustrates how in the more sizeable courses at NTNU such as TDT4100 and TDT4102, a team of evaluators might spend more than 250 hours, equivalent to nearly two man-months, grading hand-written exam submissions, costing the university tens of thousands of NOK. (These numbers were derived during our project thesis in the fall semester of 2015.) An examination system that allows for (semi-)automatic grading would be significantly more efficient with regard to both time and cost than the current process, and would furthermore provide a thoroughly unbiased and completely reliable form of evaluation [SV15].

	TDT4100	TDT4102
Hours	296	257
Professor	53,354	46,324
Associate Professor	42,805	37,165
Assistant Professor	36,328	31,542
Research Fellow	38,089	33,071
Research Assistant	30,769	26,715

Table 1.1: Costs (in NOK) of evaluating programming exams.

Implementing new university standards is a process that requires time and extensive research. Thus, this project will be the first step of a longer lasting experiment; this initial work seeks to provide a solid foundation on which further research on enforcing the use of digital exams at NTNU can be achieved, as well as producing an example of how this type of exam can be structured.

1.2 Assignment Interpretation

This project is a part of the commencement phase of a longer undertaking, aiming to explore the feasibility of a digital examination approach in programming courses at NTNU. As a way of reaching a conclusion to this matter, the problem statement describes a set of mandatory and optional elements to include our research. For easier reference later in this thesis, the elements are enumerated and listed below. The optional elements are marked with an asterisk (*****).

Research

R1 Identify alternative question types that are suited for an autograding system for C++ and other programming languages.

R2 Discuss which of the question types identified in **R1** are best suited to the TDT4102 course at NTNU.

R3 Outline ideas for a realistic 5-year track making the vision "digital exam with autograding" a reality at NTNU.

R4* Identify how user interface improvements can be implemented to better reach the overall goal, and describe these as prioritized proposals to the CMB project.

Experiment

E1 Organize an experimental midterm test in the TDT4102 course at NTNU.

E2 Develop and assemble a set of questions to be used in the midterm test.

E3 Administer a workshop to educate potential experiment participants about the CMB system.

E4 Perform a thorough system test of CMB.

E5* Discuss how the experiment results could be correlated with other more traditional exam-like tests with the purpose of assessing the goodness of our approach.

Implementation

I1 Implement a solution for extracting the complete results from the tests to data in an Excel spreadshet.

1.3 Hypotheses

After the experiment related to subtask **E1**, we will analyze the results to reject or accept each of the following null hypotheses:

$H0_1$: Having a computer when answering multiple choice questions is not an advantage with regard to performance.

$H0_2$: Having a computer when answering programming problems is an advantage with regard to performance.

$H0_3$: The use of CMB is equally efficient as the current method of solving an exam.

$H0_4$: The use of CMB is equally motivating to use as the current method of solving programming problems.

In $H0_4$, "motivation" refers to the level of satisfaction the students experience while using the CMB system.

1.4 Project Timeline

As the subtasks defined in Section 1.2 have a natural chronological order, we have constructed a rough timeline of our project, which is presented in Table 1.5. This timeline also includes other important events during the semester in which our research is conducted, in order to illustrate how our tasks correlate with the university

calendar.

Week	Event	Task
2	Lecture commencement	
8	CMB workshop	**E3**
10	Pilot run and system testing	**E4**
11	Voluntary midterm experiment	**E1**
12	Easter holidays	
17	Lecture conclusion	
23	TDT4102 final examination	

Table 1.5: Project timeline.

During the weeks leading up to the experiment, our main focus will be on researching appropriate question types (**R1** and **R2**) and developing the midterm assignment (**E2**). Before the experiment it is also important that task **I1** is completed; if not, we will have no simple way of extracting the midterm results, which will greatly delay our work and probably affect the quality of our analysis. After week 11, we will analyze the results in order to better suggest the directions for the project in the years ahead (**R3**).

1.5 Contributions

Overall, our project contributes to the general progress towards implementing digital programming exams at NTNU. More specifically, we have listed a set of four explicit contributions in Table 1.6, which also locates the various sections of the thesis where each task is dealt with.

	Contribution	Task	Location
1	A research study on various question types suitable for automatic grading.	**R1** and **R2**	Chapter 3
2	An original example of a midterm assignment for the TDT4102 course at NTNU.	**E2**	Section 4.3

3	Valuable insight with regard to the use of CMB or other online judge system as a tool for digital examination, compared to the current approach.	**E3**, **E4** and **E5***	Section 4.2.3 and Chapter 6
4	Guidelines and suggestions for future experiments based on gained insight and our own conducted midterm experiment.	**R3** and **E1**	Chapters 4 and 7
5	Suggestions for potential user interface improvements on CMB.	**R4***	Section 7.4
6	A Python-application for converting JSON-data from CMB to an Excel spreadsheet.	**I1**	Section 4.4.2 and Appendix A

Table 1.6: Project contributions.

1.6 Outline of Thesis

The remainder of our thesis is organized as follows:

Chapter 2: Background contains background material on the most important aspects of the TDT4102 course at NTNU, online judges and the CMB system, initiatives for digital examination at NTNU, and finally statistics concepts related to our research approach.

Chapter 3: Question Types delves into our research on various question types that may be suitable for digital examination with automatic grading. This chapter is divided into two parts, where the first part deals with the question types related to the general theory that one may implement in a typical learning management system and how they are commonly evaluated. The second part discusses approaches to testing programming proficiency, and how these approaches may have to be customized to fit CMB or online judge-type systems in general.

Chapter 4: Voluntary Midterm Experiment deals with all aspects of our experiment, from the planning phase to the actual execution. In this chapter, we also present the midterm assignment used in the experiment, as well as the manner in which we ensured anonymous participation, and how we extracted results from the applications we used.

Chapter 5: Results presents two types of results related to the midterm experiment; first of all, we look at the participant results measuring their proficiency in C++ . Second, we discuss a set of null hypotheses which are then statistically analyzed

based on implicit aspects of the experiment such as average scores and results from a questionnaire.

Chapter 6: Discussion provides discussions on the reliability and validity of CMB as an assessment tool, as well as our thoughts on the experiment results, positive and negative experiences from the project in hindsight.

Chapter 7: Future Work suggests a direction for the project moving forward. In this chapter, we propose a new experiment, as well as presenting a prospective plan for the next five years. Finally, we discuss potential new requirements for the CMB system in order to make it more suitable as an examination tool in an introductory programming course.

Chapter 8: Conclusion summarizes our tasks and contributions, providing a final evaluation of how these tasks have been executed.

Chapter 2
Background

In this chapter, we present background material relevant to our project. First, in Section 2.1 we thoroughly discuss aspects of the TDT4102 – Procedural and Object-Oriented Programming course at NTNU, from which we have used students as participants for our main experiment. We present our findings from analyzing trends regarding question types in written exams from the past eight years. Next, we describe the current process in effect at NTNU for implementing the use of digital tools for assessment in Section 2.2. Section 2.3 explains the concept of online judge systems. Section 2.4 introduces the Climbing Mont Blanc system, which is the digital tool we use in our experiment on digital assessment; in this section we describe how to create, manage, and solve problems on the system. Finally, we present statistical approaches for hypothesis testing in Section 2.5 which will be relevant in Chapter 5.

2.1 TDT4102

TDT4102 – Procedural and Object-Oriented Programming is an introductory programming course taught annually at NTNU, aiming to instruct students in basic concepts of the C++ programming language, as well as the most important aspects of object-oriented programming and how it differs from procedural programming. A registered student count of approximately 800 and a course staff of 56 makes TDT4102 one of the largest courses in its department.

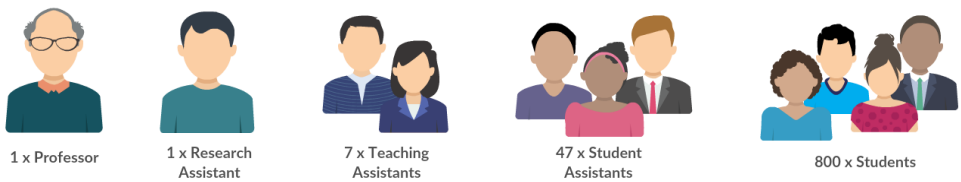


Figure 2.1: TDT4102 course size.

2.1.1 Learning Objectives

Listed below are the formal learning objectives for the course, obtained from the official TDT4102 course site [oSNa]. Each learning objective is identified with a short key for easy reference later in the paper.

Knowledge

K1 Has broad and practical oriented knowledge of the C++ programming language. Knows the syntax and rules for variable declarations and data types, type conversions, control structures, functions and operators, overloading, classes, inheritance, templates, exceptions.

K2 Has knowledge about automatic and dynamic variables and the use of pointers.

K3 Has knowledge about recursion, simple algorithms and data structures.

K4 Has knowledge of procedural- and object-oriented modularization of code and how code can be organized in multiple files, compilation and linking.

K5 Has knowledge of the standard libraries, in/out data, commonly used library functions and template-classes.

K6 Has knowledge of modern development tools, techniques for debugging and simple testing of code.

K7 Has knowledge of diagrams for object-oriented programs.

Skills

S1 Can develop a program from description of the problem to a working solution without errors.

S2 Knows commonly used coding techniques and patterns and is able to work efficiently and iteratively in the construction of a solution.

S3 Can create procedural programs where the code is reasonably modularized in functions, and object-oriented where the code is organized in classes. Is able to choose a solution that fits the problem.

S4 Is able to write code that is readable, reusable and simple to maintain.

S5 Is able to read code and understand how the code behaves in runtime.

Competence

C1 Can analyze a problem and construct a solution efficiently.

C2 Can communicate and discuss code solutions and explain how a program behaves.

C3 Is able to find tools and aids, find and use documentation of the programming language and standard libraries.

2.1.2 Participant Statistics

Gaining proficiency in C++ through the TDT4102 course is a mandatory part of certain study programs at NTNU, including Energy and Environmental Engineering (MTENERG), Cybernetics and Robotics (MTTK), Applied Physics and Mathematics (MTFYMA), and Electronics Systems Design and Innovation (MTELSYS). For other study programs, including Industrial Economics and Technology Management (MTIØT), Nanotechnology (MTNANO), Natural Science with Teacher Education (MLREAL), Chemical Engineering and Biotechnology (MTKJ), Physics (BFY), and Informatics (BIT), TDT4102 is an elective course option. In addition to this, other students at NTNU, especially those studying Computer Science (MTDT), have been known to take the course for supplementary competence.

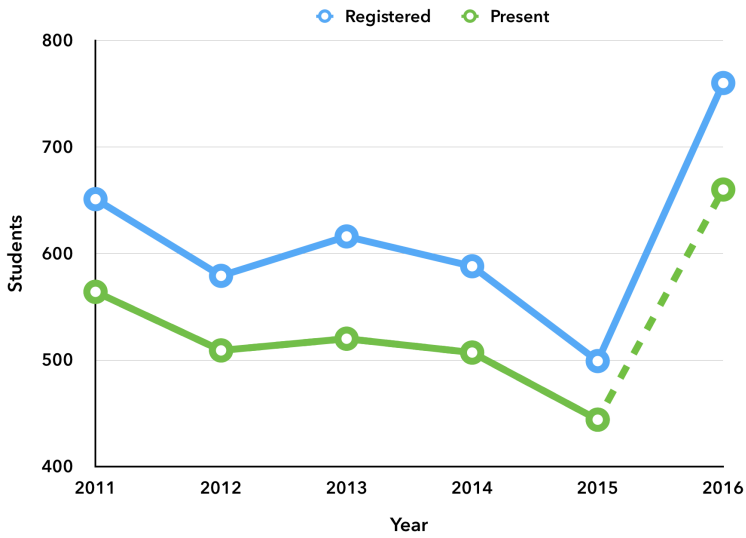


Figure 2.2: Examination statistics 2011–2015, including estimations for 2016.

From reports generated on NTNU's Grade statistics tool [oSNb], reinforced by the above information, one can infer that TDT4102 is one of the top five courses at the Department of Computer and Information Science (IDI) with regard to student enrollment. Figure 2.2 presents the number of students who were registered and present for the final examination during the past five years, as well as how many

students are registered for evaluation in 2016. The number of students present for the final examination in 2016 is an approximation calculated from percentages of preceding years.

Recent deviations in the course structure of the MTTK study program led to a situation where both first- and second-year students were enrolled in the TDT4102 course in 2016, explaining the significant increase in candidate numbers from the previous years, as well as the small drop from 2014 to 2015.

2.1.3 Exercises

Alongside the lectures, registered students are expected to complete an exercise project consisting of twelve separately evaluated programming exercises, of which eight must gain approval in order to take the final examination. Table 2.4 contains an overview of the exercise project as implemented in 2016.

No.	Content	Due (Week)
1	Transitioning from Matlab/Python to C++.	3
2	Procedural programming in C++; standard input/output; general intro to functions.	4
3	C++ file structure; using standard library functions; pseudo-random numbers; introduction to pointers.	5
4	Call-by-value vs. call-by-reference; arrays; creating a simple program.	6
5	Enums, structs, and classes.	7
6	Operator overloading.	8
7	Dynamic allocation of memory.	9
8	Streams and files.	10
9	SFML; creating a game with a graphic user interface.	13
10	Inheritance; polymorphism; virtual functions.	14
11	Standard Template Library; iterators; custom templates.	15
12	Exception handling; smart pointers.	16

Table 2.4: Exercise overview, TDT4102 2016.

2.1.4 Final Exam

A four-hour written examination concludes the students' enrollment in the course and fully determines their final grade. In accordance with support material code C [oSNe], students are permitted to bring a standard basic calculator as well as the course textbook, chosen by the lecturer each semester. As mentioned earlier, the final exam requires the students to produce hand-written code, as well as answer theoretical questions related to procedural and object-oriented programming with C++.

An analysis of the ordinary and re-sit examinations from the past eight years exhibits certain trends related to the questions types included in the problem set, listed and exemplified below.

Code Understanding

Most of the examinations commence with a few simple code understanding questions, in compliance with learning objectives **S5** and **C2**. This question type typically requires the student to analyze a block of code and either explain the functionality or determine the output, as exhibited in Figure 2.3.

Q What values are printed?

```
1 int a[] = {1, 2, 3, 4, 5};
2 int *b = a;
3 int *c = &a[2];
4 a[0] = 0;
5 cout << a[0] << endl;
6 cout << *b << endl;
7 cout << *(++c) << endl;
```

Figure 2.3: Exam 2011, Task 1b).

A derived form of this question was detected, where the student must analyze a block of code and explain why it does not perform as intended. The planned functionality is either described (as exemplified in Figure 2.4) or must be derived from the code sample itself (Figure 2.5).

Simple Problem Solving

In order to test the student's ability to solve simple problems with programming and satisfy learning objectives **S1**, **S4**, and **C1**, the exam often includes a handful of smaller problem descriptions from which the student must produce a coded solution, typically in one function. An example of this is presented in Figure 2.6. For this

Q The function `ArrayAlloc` should create an `int` array with a given size `n` and initialize all the elements in the array to a given value `v`. The following code is a first attempt at implementing `ArrayAlloc`:

```
1  int* ArrayAlloc(int n, int v) {
2      int ret[n];
3      for (int i = 0; i < n; i++)
4          ret[i] = v;
5      return ret;
6  }
```

Explain in one sentence (max 20 words) what is wrong with this implementation.

Figure 2.4: Exam 2013, Task 1b).

Q This code does not behave as planned. Find the error and explain what the correct code is.

```
1  int x = 5;
2  int y = 6;
3  if (x =! y) {
4    cout << x <<  "is different from" << y << endl;
5  } else {
6    cout << x << "is not different from" << y << endl;
7  }
```

[Enclosed hint] The source code compiles and runs, but prints out "0 is not different from 6". When you check the compiling output, you find a "Warning" stating "Using the result of an assignment as condition without parenthesis". This is a warning produced by XCode and it indirectly indicates what the error is.

Figure 2.5: Exam 2014, Task 1e).

question type, the teacher may test any of the learning objectives categorized as **Knowledge**, depending on the problem to be solved.

Q Write a function `bool isLeapYear(int year)` that returns `true` if `year` is a leap year and `false` if not. The function must implement the rules for leap years.
(Excluded: Formal definition of a leap year quoted from Wikipedia.)

Figure 2.6: Exam 2011, Task 2a), abbreviated.

Constructing a Program

Almost every published exam has a task where the student is expected to combine several aspects of the curriculum in the implementation of a complete program, typically in the form of a game. Problems of this type often account for the biggest percentage of the overall grade and thus require the longest time to solve. This task is highly appropriate to include because, depending on the functional requirements, the implementation process may cover most of the learning objectives of the course, especially those involving object-oriented programming and coding technique. In the final exam of 2015, the students were asked to implement elements from the popular video game Snake [Wik], involving graphics programming, the C++ Standard Template Library (STL), classes and inheritance, as well as acquainting oneself with provided code. Figure 2.7 quotes the original problem introduction, shortened to signify the extent of the task, which consisted of nine subtasks.

> **Q** In this assignment you will implement parts of a game that commonly is called **Snake**. In this game, the user controls with the arrow keys a snake that moves around on the screen and eats pieces of food that pop up at random places. Whenever a piece is eaten the length of the snake increases. The game is over if the snake hits the borders of the window or if the snake collides with itself. [...] The challenge in this assignment is to understand a somewhat complex problem (given the time limits) and to design and write code that is essential for the program. [...] Examples and explanations can be found in the appendix. Start by figuring out how `main()` works.

Figure 2.7: Exam 2015, abbreviated introduction to Task 2.

2.2 Digital Assessment at NTNU

Introducing digital exams could greatly improve upon processes that are currently in place at Norwegian university institutions [SV15]. In August 2015, Berit Kjeldstad from the rectorate at NTNU wrote a blog post presenting the *Digital Exam* project at NTNU, aiming for a full digitalization of exams at NTNU within 2022 [Ber15]. As of 2015, NTNU is the university in Norway with the highest number of annually conducted exams with about 132 000 exams. Kjeldstad states the two primary reasons for this digitalization. Firstly, is for the students to be able to take their exams digitally, whether it is a regular exam or other forms of evaluation. Secondly, the digitalization is just as necessary for the course staff (teachers, sensors, administration, etc.) as it will improve workflow. The system that has been chosen for this project is Inspera Assessment (discussed in Section 2.2.1), which is specifically aimed at making the final examination process in courses at NTNU more efficient and digitalized. In

the fall semester of 2015, a handful of courses from various institutes at the university were selected for pilot testing of the system.

Organization of courses at NTNU are done through a Learning Management System (LMS); itslearning, which is further discussed in Section 2.2.2, is the current system of choice, but will be replaced by Blackboard in 2017 [Norc, Sol16]. Many courses have a mandatory or voluntary mid-term evaluation during the semester, often in the form of a multiple choice quiz completed on itslearning, which has an extensive framework for this type of test. For this reason, we will utilize itslearning for our midterm experiment; fortunately, Blackboard has an equally extensive test framework with the same capabilities as itslearning. Thus, our findings may be applicable to the new system if our research is continued in the future.

2.2.1 Inspera Assessment

Inspera Assessment is the flagship product of Inspera AS, and is currently being introduced as a tool for digital examination at NTNU [oSNd]. The system allows the students to take exams on their own devices; a concept denoted as Bring Your Own Device (BYOD). This is made possible by requiring the participants to use a browser called Safe Exam Browser (SEB). Research has shown that SEB is not a 100% secure application [Sø15], but its aim is to transform any computer into a secure workstation that regulates what utilities the exam participants can access during an examination [Saf]. Inspera Assessment also includes features such as a plagiarism checker from Ephorus [Eph], hosting by Amazon Web Services [Ama], as well as student user management through Felles Elektronisk IDEntitet (FEIDE) [Fei] and Felles Studentsystem (FS) [Fel], which are the services used by students and employees at NTNU for access to itslearning, Innsida (NTNU's intranet), and Studentweb (administration site for course registration, exam dates, grades, etc.).

Inspera Assessment is a solid system for theoretical examinations where the participants are required to answer questions and problems with a longer answer text. However, Inspera Assessment is sub-optimal when it comes to programming examinations, as it contains no or minimal support for programming such as compiling and running code, testing, etc. It does, however, support syntax highlighting for selected programming languages and displaying line numbers.

2.2.2 itslearning

itslearning is a cloud-based LMS utilized by several of the bigger learning establishments in Norway [its], including NTNU for the time being. Developed at Bergen University College (HIB), itslearning offers an extensive education-oriented forum for students and professors. Each taught course has a dedicated subpage accessible by their respective participants, assigned various roles such as *Professor*, *Teaching*

Assistant (TA), or *Student*. At NTNU, itslearning is mostly used as a platform for distributing information and organizing course exercises during a semester. However, some professors will additionally make use of the LMS as an evaluation tool by creating mandatory tests for the students using itslearning's *Test* tool.

A user with higher access rights (typically a *Professor* or a *TA*) can create a test as part of a course subpage. As test creator, one has the option to set test variables such as time limits, individual question scores, penalty scores for wrong answers, and assessment type. Of the available assessment types, the most interesting is the *grade* option, whereby a student will receive a letter grade based on their overall score. In addition to this, the test creator may define a non-static score range for each grade level; if these ranges are modified after or during the test, each student's grade is automatically recalculated. This particular functionality mirrors the grading process for regular exams at NTNU. Each question may also be weighted, such that answering certain questions correctly yields a higher score.

After a completed test, itslearning allows the extraction of result data to an Excel spreadsheet if a user with *Professor* access rights has tuned a few specific parameters in the course subpage's settings. itslearning also offers a statistics overview directly on the test page, illustrating the grade distribution as a graph, as well as accuracy statistics on individual questions. This information is unfortunately not included in the downloadable Excel spreadsheet; this file only contains a list of students and their total test scores.

Test Question Types

When creating a test, the professor may select between ten different question types, categorized as either *general* or *interactive*. Questions of the *general* classification uses familiar methods like multiple choice and open answer to test the knowledge of the participant. There are seven *general* question types to choose from:

- **Either/or**
  Users are presented with a question and exactly two alternative answers where exactly one of them is correct. Typical use cases include True/False statements.
- **Multiple choice**
  Users are presented with a question and a specified number of alternative answers where exactly one of them is correct.
- **Multiple response**
  Similar to the *multiple choice* question type, but more than one alternative answer may be correct. Users can typically distinguish a *multiple response* question from a *multiple choice* question by looking out for checkboxes instead of radiobuttons, indicating multiple selection.

- **Short answer**
  Users are presented with a question and an input field where they should submit a written answer in 90 characters or less. For automatic evaluation, the test creator can provide a list of keywords or key phrases that should be included in a correct answer.

- **Open answer**
  Similar to the *short answer* question type, but the written answer has no character limitation. Also known as *essay question*, *open answer* questions are best suited for manual assessment. However, itslearning offers automatic evaluation similar to *short answer*, where the test creator can provide keywords and keyphrases as a basis for evaluation.

- **Select from a list**
  Users are required to fill in the missing words or expressions in a given text by selecting from a dropdown list of alternatives.

- **Fill in the blank**
  Similar to the *select from a list* question type, except the users, are not provided with a list of suggestions for each missing word, but rather a blank text field.

Interactive questions are largely based on drag-and-drop. There are three question types under this category:

- **Match**
  Users are presented with two lists of equal size and are required to find relations between the items in the lists. To achieve this, the user clicks and drags an item from the second list into empty areas that are connected to each item in the first list.

- **Order**
  Users are asked to arrange items from a list in a certain order. Similar to the *match* question type, the user achieves the ordering by clicking and dragging the elements into the correct place.

- **Hotspot – Click the picture**
  Users are presented with a question and an image, and must provide an answer by clicking the correct area (the "hotspot") in the provided image. The test creator defines the hotspot by using a selection marker similar to the ones found in photo editing applications.

2.3 Online Judge Systems

From the notion that programming is a purely digital task, the concept of online judge systems has emerged to offer users an easy and fun way of learning and practicing coding [CKLO03]. An online judge system has a database containing an extensive collection of problems that can be solved with programming. Users upload their coded solutions, which are then compiled and evaluated by the system and either accepted or rejected as valid submissions based on a set of tests. Besides the acceptance status, users may receive more substantial feedback such as runtime and potentially occurred program errors. Organized programming contests such as the ACM International Collegiate Programming Contest (ICPC) [ICP, PW08] or, more locally, IDI Open [Norb], are a recurring use case for these systems.

Typical problems found in an online judge system are tasks where the user must engineer a program or algorithm that should return the correct output based on some given input. In order to test the correctness of a submission, the system utilizes a set of input and output pairs. Submissions are run with the input set, and the program's output is then compared to the correct output. Generally, the page containing the problem description will also include a sample input and output pair, so that the users have something to test their code with locally before submitting their solution. The test set that is used on the program after submitting is often quite large and will handle edge cases that may be more challenging to identify.

Kurnia et.al. [KLC01] discuss online judges as automatic grading systems for programming assignments. These are often embedded in programming courses; users have the opportunity to enroll in online classes and solve problems suited to the class they are taking. Examples of this are Kattis [EKN+11], which was developed and used actively in programming courses at the Royal Institute of Technology (KTH) in Sweden, and CMB, which is currently being introduced at NTNU in courses like TDT4102.

2.4 Climbing Mont Blanc

Climbing Mont Blanc (CMB) [Nat, NFS+15] is an online judge system especially suited for evaluating energy efficient programming solutions. Running on the state-of-the-art multicore processor Exynos Octa [Ltd14] used in modern smartphone technology from Samsung, the CMB system evaluates submissions in terms of execution time, energy consumption, and energy efficiency measured in Energy Delay Product (EDP).

CMB was initiated as the central part of the master thesis project of Torbjørn Follan and Simen Støa in 2015 [FS15], and is currently being developed by Sindre

Magnussen as a part of his master thesis. The idea that inspired these projects belongs to Professor Lasse Natvig at IDI at NTNU, who has taken the role as a supervisor for the developers throughout the process. Two separate versions of the system have been available to us on during the project: `climb-dev`, which is currently an open beta hosted by the development server, and `climb`, which is the latest production version of CMB.

CMB is inspired by the Mont-Blanc project [Taf13], which was initiated in October 2011. Behind this project lies the anticipated use of exascale computer system and the notion that their energy consumption will significantly restrict these systems [SDM10]; an exascale computer system is capable of performing at least one exaflop (10^{18}) operations per second. The Mont-Blanc project's primary goal is to develop a new computer architecture which will set the standard for future high-performance computing systems.

As far as we know, CMB is unique within its field because it considers the measured energy efficiency during evaluation of submitted code solutions. During the evaluation process, the system treats the submitted program as a black box, feeding it input and obtaining output without any concept of code structure, syntax, or program flow. Similar to other online judges, the output is then compared to the contents of a text file corresponding to the expected results, yielding an acceptance verdict which is returned to the user along with the measured energy consumption.

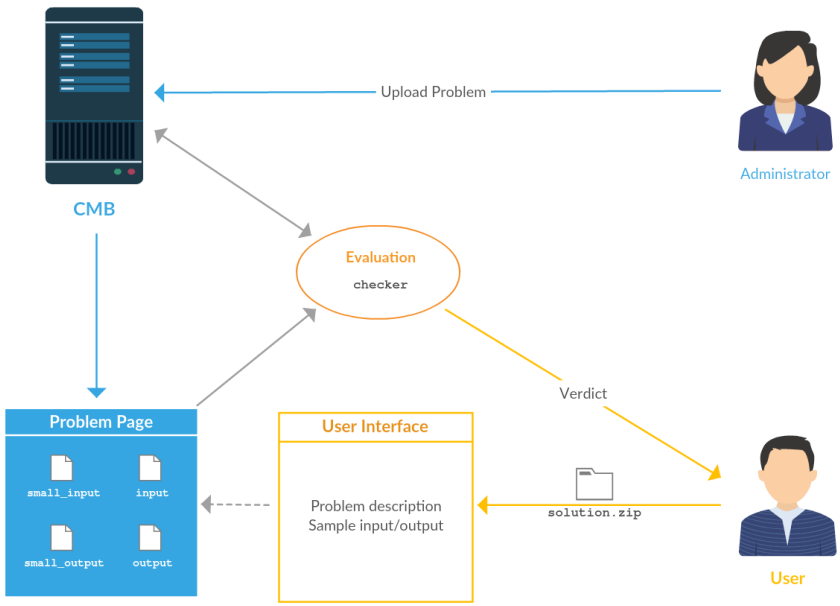


Figure 2.8: Interaction model for the CMB system.

Figure 2.8 illustrates the system interaction for an administrator (blue) and a typical user (yellow), further discussed in the following subsections.

2.4.1 Adding Problems to CMB

Adding problems to CMB is essentially a process that consists of adding files to the system. This dataset comprises four text files (.txt), and a script for testing the submitted answers. The four text files are two pairs of input and the corresponding output. One pair is `small-input` and `small-answer` which makes up the *small correctness test*, testing whether the submitted code compiles and runs without error. The other pair is `input` and `output` which is the *big correctness test* that covers all edge cases.

The script used for evaluating the submissions must be named `checker.cpp`, and is advised to be implemented individually for each problem in CMB. This way, each submission can be tested and evaluated in a tailored fashion. For instance, some exercises might yield partial credit or credit for close-to-correct answers, such as floating point numbers where the error margin can be minuscule. For problems like these where there may be multiple correct solutions, an administrator has the option to add a "goodness" evaluation parameter, measuring the degree of coverage of a problem solution; for example in the Vertex Cover problem, where the goodness value will typically be based on the percentage of total vertices covered. Feedback and error messages for individual cases can also be implemented in the `checker` if desired.

2.4.2 Solving Problems on CMB

As mentioned, CMB uses an input/output approach when evaluating submitted code. Therefore, it is crucial that the uploaded solution handles input and output; the resulting program must be able to read input from a stream of undefined size, and write to output in the appropriate format, often exemplified on the problem description page. Generally, this task is straightforward and can be implemented as presented in Listing 2.1, which is derived from the system documentation [CMB] and is especially suited for exercises that perform an operation for each line of input. In more complex problems, one may need to preprocess each line of input (e.g., typecasting, or deriving distinct components). Output can be written in several manners; after testing the system we found any combination of `cout`, `printf`, `cerr` and `clog` to be successful.

Certain guidelines specified by the system documentation must be taken into consideration when uploading solutions to CMB. As mentioned, the most important requirement is that the code should be able to handle input and output. Other significant requirements are related to the format of submissions, such as file ex-

Listing 2.1 Example of input/output handling.

```
#include <iostream>
#include <string>
using namespace std;

string solveProblem(string num);

int main(void) {
  cin.sync_with_stdio(false);
  string input;
  while (cin >> input)
  {
    cout << solveProblem(input) << endl;
  }
  return 0;
}
```

tension restrictions and naming conventions. All code files (`cpp`, `c`, `h`, or `hpp` files) associated with a submission must be placed into a single directory, which must then be compressed into a `zip` file. This `zip` is then unpacked and its contents are preprocessed and compiled, a procedure which is performed by a sequence of bash scripts. Because of the lack of filename sanitation, subdirectories and filenames cannot begin with a hyphen (`-`), nor include instances of the slash (`/`) or whitespace character. Contradicting these conventions can potentially lead to unwanted behavior during compilation, which, in the worst case, may lead to system failure.

2.5 Hypothesis Testing

In Section 1.3, we presented a set of null hypotheses $H0_x$, which are each formed in such a way that they are rejected if there is a significant difference in the samples involved in each respective hypothesis [BS87]. $H1$, called the alternate hypothesis, is considered true if $H0$ is rejected. However, $H0$ can not be considered true if it is not rejected and $H1$ cannot be considered false. Later, in Chapter 5, we will use the terms *one-tail-* and *two-tail* tests. A one-tail test seeks to determine whether or not the mean of the test set A (μ_A) is greater or less than the mean of test set B (μ_B). A two-tail, on the other hand, seeks to determine whether or not μ_A is unequal to μ_B.

$$H1 = \begin{cases} \mu_B > \mu_A \text{ or } \mu_B < \mu_A & \text{if one-tail} \\ \mu_B \neq \mu_A & \text{if two-tail} \end{cases} \tag{2.1}$$

This process comprises some statistical calculations. If the investigated hypothesis, $H0$, involves two sets of data, one must start off with an F-test [ML06] to compare the variances. The main test is the T-test [Ros14], which will be used to determine whether or not $H0$ should be rejected. The results of the F-test is needed for the T-test to produce appropriate results.

To reject the hypothesis, the T-test must show that there was, in fact, a significant difference between the two sets of results, as opposed to a coincidental difference. This is done by comparing the p-value from the T-test with a commonly defined value, $\alpha = 0.05$. If the T-test states that $H0$ cannot be rejected, no further testing is needed. On the other hand, if the T-test states that $H0$ can be rejected, more testing should be done in order to ensure and strengthen validity. Since the T-test is sensitive for normality, one can conduct a Shapiro-Wilk test [SW65] to determine whether the samples are normally distributed. If that is the case, one can assume the conclusion from the T-test to be correct. If not, a Mann-Whitney test, which is a non-parametric test, is needed to increase the validity of the conclusion [Ros14].

The tools we will be using for the analysis are *Excel 2013* [Micb] with the *Analysis ToolPak* add-on [Mica] for the F-test and T-test and *Real Statistics Resource Pack* [Staa] for the Shapiro-Wilk test, *Mann-Whitney U Test Calculator* [Stab] for the Non-Parametric test, the *Effect Size Calculator* [Uni] for Cohen's d and Hedge's g, and finally the *Sample Size/Power Calculator* [Bio] for calculating the number of participants needed in each group to achieve a strong effect size.

2.5.1 Conclusion Validity

After the final conclusion about the hypothesis is drawn, the effect size is calculated to determine a measure of confidence of the conclusion; the lower the effect size, the less confident one can be about the conclusion. Measures commonly used are *Cohen's d* [Coh92] and *Hedge's g* [HO14], where the latter is used because it takes the population sizes into account. The effect size can further be used to calculate the desired size of each sample in order to make the conclusion sufficiently powerful. This calculation is useful for future references.

Small	0.2
Medium	0.5
Large	0.8
Very large	1.3

Table 2.5: Common effect size indices for Cohen's d.

Common effect size indices for both Cohen's d and Hedge's g are shown in Table 2.5 [SF12]. With a relatively large effect size, one can be certain to avoid erroneous

conclusions as shown in Table 2.6.

	$H0$ **is true**	$H1$ **is true**
Do not reject $H0$	Correct decision	Type II error
Reject $H0$	Type I error	Correct decision

Table 2.6: Outcome of hypothesis testing.

Chapter 3
Question Types

This chapter contains an in-depth study of various question types, categorized into two sections: Section 3.1 discusses question types that are not directed toward pure programming tasks, but rather suited for theoretical questions typically answered using itslearning or other possible learning management systems that support automatic evaluation. Section 3.2 covers more general questions types that explicitly involve programming, and that are suitable for CMB. For each of the presented question types, we will provide an example related to the C++ programming language as well as a discussion on whether the presented question types are well-suited for automatic grading on the respective systems.

3.1 LMS Question Types

Part 1 of our proposed examination form consists of a set of questions that do not directly require programming to be answered. Instead, they cover the more theorical aspects of C++ and thus satisfying the TDT4102 learning objectives (presented in Section 2.1.1) **K1** through **K7**, **S5**, and **C2**.

The question types in this section are all based on the question types offered by itslearning, first presented in Section 2.2.2.

3.1.1 Multiple Choice (MC)

A *Multiple Choice (MC)* question, exemplified in Figure 3.1, consists of a textual question (the *stem*), and a list of answers, where one is correct (the *key*), and the rest are wrong (the *distractors*); respondents are asked to distinguish the key from the distractors.

An important disadvantage of using MC questions is that upon not knowing the correct answer, a respondent still has a $\frac{1}{i}$ chance of guessing the key, where i denotes the total number of alternatives. In the case of *either/or* and *multiple response*

Q What is the difference between a `struct` and a `class` in terms of default access modifier?

- ◯ All `struct` members are public while `class` members are private.
- ◯ All `struct` members are protected while `class` members are public.
- ◯ All `struct` members are public while `class` members are protected.
- ◯ All `struct` members are private while `class` members are public.

Figure 3.1: Example of multiple choice question.

questions (discussed below in Sections 3.1.2 and 3.1.3) the probability is even higher. This issue is commonly addressed by penalizing wrong answers with a negative score to counterbalance the points accumulated from guessing [Bur04, Hol24].

Guidelines

According to Hansen and Dexter [HD97], «a high quality MC question should present a task that is clearly understood and be constructed so that it can be answered correctly by those who have achieved the intended learning outcome, but so that its content or structure does not reveal the correct answer to the uninformed student». Several research teams have produced sets of guidelines for ensuring quality in MC items [Aik87, HD89, HDR02]. We list the most commonly occurring guideline entries as follows:

- **Avoid negative stems.** Using terms like "not" in the stems increases the complexity of the question. If it is absolutely necessary to use the negative wording, emphasize negative terms so that they are less likely to be overlooked.
- **Balanced key positioning.** Position the key so that it appears the same number of times in each possible position.
- **Punctuation conventions.** Use capitalization and punctuation in a consistent manner. Alternatives should be capitalized when they represent answers to a question; alternatives should not be capitalized if they represent completion to a statement. Unnecessary punctuation when used with numbers may be confused with decimal points.
- **Grammatical consistency.** The stem and alternatives should all be grammatically consistent; a lack of consistency may reveal the key or enable the respondents to eliminate certain distractors.

- **Correctness of key.** Make sure that the key is clearly the only correct or single best option.
- **Avoid absolute terms.** Statements that include words like "always" or "never" are usually false, and increases the probability of an uninformed respondent guessing the correct answer.
- **Distractor plausibility.** The overall quality of an MC question is often decided by the quality of the distractors; make sure that all of the distractors seem plausible for respondents who cannot distinguish the key from knowledge.
- **Length consistency.** Make sure that all of the alternatives are of approximately the same length.
- **Avoid verbal clues.** Using more textbook language in one of the alternatives may disclose the key. Two alternatives that are worded differently but have essentially the same meaning eliminates them both as distractors, as there can only be one correct answer.

3.1.2 Either/Or

A direct sub-category of the MC question type is the *either/or* question type, where there is exactly one key and exactly one distractor. True/false questions, like the one presented in Figure 3.2, are the most common occurrence of this question type, but one can also specify other variants as exemplified in Figure 3.3.

Q In C++, all variables must be declared before they may be used.

◯ True ◯ False

Figure 3.2: Example of true/false question.

Q Which copy-approach does the default copy-constructor use?

◯ Deep copy ◯ Shallow copy

Figure 3.3: General example of either/or question.

3.1.3 Multiple Response (MR)

The *Multiple Response (MR)* question type is another sub-category of MC, where there may be more than one key in the list of answers, but not necessarily. Thus,

an MR question is more complex to answer than an MC question, as the number of keys is unknown to the respondent. A typical use of an MR question is listing a set of statements and asking the respondents to identify the ones that are true. In a programming course, one may provide a variation of this, exemplified in Figure 3.4.

Scoring

When scoring an MR question, one must choose between a *polytomous* or *dichotomous* approach. Polytomous scoring implies the use of partial credit, which can be done in a number of ways based on various weighting methods [DLT+95]. Dichotomous scoring, on the other hand, is based on an all-or-nothing concept requiring all selected responses to be correct for a question.

Q (5 points) According to the class hierarchy given by the following code, which of the declarations are valid?

```
1 class A {};
2 class B: public A {};
3 class C: public B {};
4 class D: public A {};
```

☒ ***A a = C();*** ☒ `D d = B();`

☐ `C c = A();` ☐ ***B b = C();***

Figure 3.4: Example of multiple response question, with answer.

Figure 3.4 exemplifies a respondent who has selected one correct and one wrong alternative (the correct answers are marked in bold text) to an MR question with a possible score of 5 points. A dichotomous scoring approach would yield 0 points because the submitted answer is not 100% correct, whereas a polytomous approach would yield a score between 0 and 5 depending on the scoring algorithm.

itslearning's test framework uses a dichotomous scoring model for MC, and a polytomous model for MR; each response alternative can yield either 0 or $\frac{q_{max}}{n}$ points, where q_{max} denotes the maximum possible score for question q, and n is the number of response alternatives for question q. An option yields points when its checked-state corresponds to its correctness value; hence, an incorrect option will yield points if it is *not* checked, according to itslearning's evaluation model. Using this scoring model, the answer submitted in Figure 3.4 would amount to 2.5 points. Selecting all of the alternatives yields 0 points unless they are all correct, but this would serve against the purpose of a fair test. Selecting none of the alternatives also yields 0 points.

3.1.4 Open Answer

An *open answer* question is ideal to use in a code understanding situation, or when testing the ability to reflect upon an issue or about a statement. Out of all the question types discussed in this chapter, the open answer question type is the one that most greatly corresponds with the format of the questions used in the current examination form. Below are two examples of how this question type could be used in our proposed digital examination form, as a code understanding question, modified from its original appearance in the final exam of TDT4102 in 2013 (Figure 3.5), and as a reflection question (Figure 3.6).

Q The class `Tree` below is used to create a binary tree. The class definition declares a constructor and two member functions. The implementation of one of the member function is given below the class definition. What does the member function `insert(string name)` do?

```
1  class Tree {
2    Tree *_left;
3    Tree *_right;
4    string _name;
5  public:
6    Tree(string name) : _left(NULL),
7                        _right(NULL), _name(name) {}
8    void insert(string name);
9    void insert(const Tree &tree);
10   friend ostream& operator <<(ostream&, const Tree&);
11 };
12
13 void Tree::insert (string name) {
14   Tree **destination = &_right;
15   if (name < _name) {
16     destination = &_left;
17   }
18   if (destination == NULL)
19     (*destination) = new Tree(name);
20   else
21     (*destination)->insert(name);
22 }
```

Figure 3.5: Example of open answer question with code understanding.

Q Should the stream-operator (<<) be implemented as a member? Why, or why not?

Figure 3.6: Example of open answer question with reflection.

3.1.5 Short Answer

Short answer questions are well-suited for assessing a respondent's ability to provide short and concise descriptions of possibly extensive concepts. Being able to do this reflects the level of knowledge within a specific subject; requiring a short answer eliminates the possibility of "covering up" the fact that one does not know the exact answer by submitting a wordy and obscure response. As opposed to *open answer*, the respondents are restricted by a character limitation with regard to their submitted answer. On itslearning, this limitation is set to 90 characters; depending on the question, limiting the input length to 90 characters may be too demanding, but in these situations, a blurry character limitation that may still classify as a short answer question can be simulated using the *open answer* question type. Figure 3.7 exemplifies a short answer question where 90 characters are more than enough, whereas Figure 3.8 presents a question where 90 characters may be too restrictive.

Q What is the output of the following program?

```
#include <iostream>
#include <string.h>
using namespace std;

int main() {
  char str[] = "Hello\0World";
  cout << strlen(str) << " " << sizeof(str);
}
```

Figure 3.7: Example of short answer question.

Automatic evaluation

Because of the reflective nature of a *short* or *open answer* question, being able to provide automatic evaluation is a significant challenge. itslearning has attempted to face this challenge by allowing the teacher to prepare a list of key words that are expected to be included in a correct answer. However, grammatical aspects such as typographical errors or synonyms must be considered by the teacher, as the system can only handle upper- and lowercase transformations automatically. Thus, to reflect a realistic manual evaluation, the teacher must think of every possible variation of

Q The following function should create an `int` array with a given size `n` and initialize all the elements in the array to a given value `v`. Explain in one sentece what is wrong with the implementation.

```
1  int* foo(int n, int v) {
2    int ret[n];
3    for (int i = 0; i < n; i++) {
4      ret[i] = v;
5    }
6  return ret;
7  }
```

Figure 3.8: Example of simulated short answer question.

the keywords, and even then, there is no guarantee that the respondent has used the keywords in the correct context.

3.1.6 Fill in the Blank

In a *fill in the blank* question, the respondent should fill in the missing words or phrases from a given text. Figure 3.9 shows an example where the text is a code segment, requiring the respondent to understand the given code and be able to complete it the intended way.

Q Fill in the missing pieces of the following code segment. The purpose of the program is to swap the values of the two pointers `first` and `second`.

```
1  void swapPointers(int* first, int* second) {
2    int firstValue = ???;
3    int secondValue = ???;
4    ??? = secondValue;
5    ??? = firstValue;
6  }
7
8  int a = 2;
9  int b = 4;
10
11 swap(???, ???);
```

Figure 3.9: Example of fill in the blank question.

Synonymity

A potential pitfall with the *fill in the blank* question type on itslearning is that the automatic evaluator checks perfect equality between the submitted string and the correct string. There is no option to provide a list of synonyms or alternative ways of expression; for instance, in the above example (Figure 3.9), the correct answer for the first blank (`???`) is `*first`, but an alternative and completely valid way of dereferencing a pointer in C++ is `*(first)` (with parentheses) or `* first` (with whitespace).

Blackboard has addressed this issue by allowing a list of correct answers to be included by the test creator, as well as introducing three options for evaluation of each input field: *exact match*, *contains*, and *pattern match*. *Exact match* evaluates each input similar to itslearning, that is, the input string should be exactly the same as the correct answer provided by the test creator. *Contains* allows for minor variations to the provided correct answer, such as the plural form. That is, the correct answer must be a substring of the respondent's submitted answer. Finally, *pattern match* evaluation checks the submitted answers using regular expressions [Fri02] so as to allow for more variability of answers.

3.1.7 Select From a List

Q Fill in the missing pieces of the following code segment. The purpose of the program is to swap the values of the two pointers `first` and `second`.

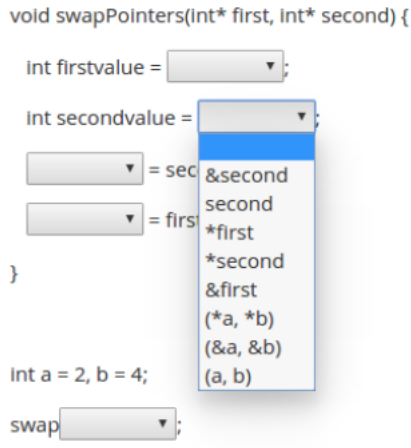


Figure 3.10: Example of select from a list question.

The *select from a list* question type is similar to the aforementioned *fill in the blank*, except the respondent is not required to directly type in the missing words or phrases, but rather select the correct option from a provided list of alternatives. In a way, each missing word or phrase becomes an MC question. Figure 3.10 presents the same question as in Figure 3.9, but with a dropdown menu with options instead of blanks. The dropdown menu is the same for all of the blanks, even if the options have already been selected elsewhere in the text, and the test creator can add additional distractors that do not belong in any of the blanks.

3.1.8 Match

Match questions involve finding the relation between pairs of elements from two separate lists. Figure 3.11 illustrates a very simple example of such a question. Our main concern is that questions of this type might be too trivial to include in an exam context.

Q Match the value with the correct type.

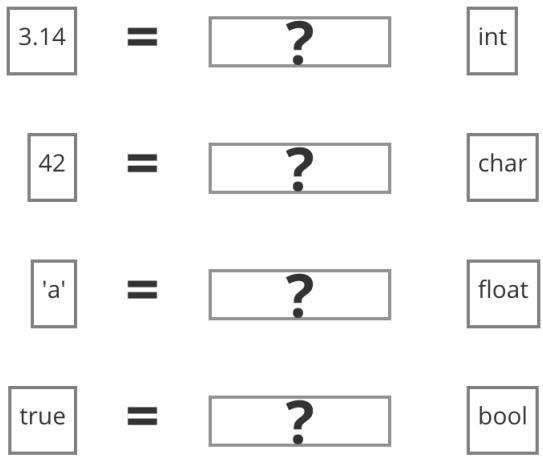


Figure 3.11: Example of match question.

The boxes on the right side should be drag-and-dropped into the correct areas marked with "**?**".

3.1.9 Order

Order questions involve sorting a list of elements with regard to some condition described in the problem text. An example of an order question is presented in

Figure 3.12, in which the boxes should be dragged-and-dropped into the appropriate order in the empty placeholders.

Q Sort the operators by order of evaluation. An operator to the left will be evaluated *before* the operator to the right.

Figure 3.12: Example of order question.

Scoring approach

A critical drawback with the *order* question type when using itslearning is that the automatic grading feature only considers whether the elements are in the exact correct index in the placeholders, and does not take into consideration the relative order of the elements. That is, if one element is misplaced such that the succeeding elements are shifted one place to the right, no points are rewarded for getting the relative order of the remaining elements correct. This is illustrated in Figures 3.13 and 3.14; the first figure shows the correct answer to the order question in Figure 3.12, giving the respondent a maximum score of 10 points. The second figure, however, will only yield a score of 2 points because only the first element is placed in its exact correct place.

Figure 3.13: Correct answer to order question example.

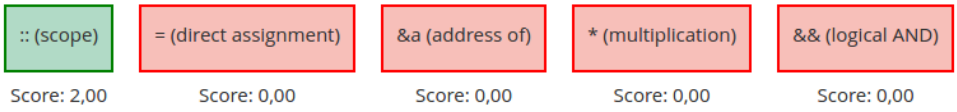


Figure 3.14: Incorrect answer to order question example.

A third situation is exemplified in Figure 3.15, where the elements are placed backward. None of the elements are positioned relatively correct to each other; however, one of the elements is in the exact right place yielding 2 points, which makes little sense with regard to the question.

Figure 3.15: Reversed answer to order question example.

An alternative way of scoring this type of question is thus to consider each unique pair of elements and give a partial score based solely on their relative order. For the submitted answer in Figure 3.14, the scoring for each element would proceed as follows:

- **:: (scope)** is relatively correct to all of the other elements, yielding a maximum partial score of 2 (q_{max}) points.
- **= (direct assignment)** is positioned relatively correct to the first element, but not the rest. Thus it gains a partial score of 0.5 ($q_{max} \times \frac{1}{n-1}$) points.
- The three remaining elements are positioned relatively correct to five out of six of the other elements, thus yielding a partial score of 1.5 ($q_{max} \times \frac{3}{n-1}$) points each.
- The total score is the sum of all of the partial element scores: $2+0.5+3\times1.75 = 7.75$ out of a possible 10.

Here, $q_{max} = \frac{s_{max}}{n}$, where s_{max} is the maximum score for the question, and n is the number of elements to be ordered (in this case, $s_{max} = 10$ and $n = 5$).

Using the same scoring method on Figure 3.15 would yield a total score of 0 points as none of the elements are positioned correctly relative to each other.

3.1.10 Hotspot – Click the Picture

Hotspot questions requires the respondent to click on a certain area of a provided image to locate something described in the problem text. For example, Figure 3.16 shows the picture of a motherboard, and the task is to locate the Central Processing Unit (CPU).

A potential pitfall with the *hotspot* question type is the potential lack of boundary precision of each component in the picture. A test creator may not necessarily include the explicit boundaries, and a respondent might intend to press the correct component, but miss the borders. Alternatively, and more relevant for the course TDT4102, one could use this question type to require the students to locate a certain

Q Locate the CPU on the motherboard.

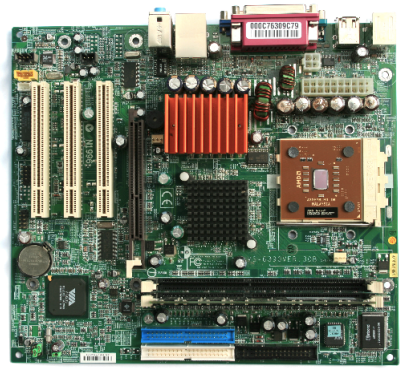


Figure 3.16: Example of hotspot question.

code line or block, function or similar. This would resemble both the MC and *select from a list* question types, except the alternatives are not explicitly shown.

3.1.11 Suitability

Although all of the question types are to a certain degree suitable with regard to testing the programming knowledge of a respondent, a central criterion for suitability in this context is the possibility of automatic evaluation in a fair manner that should reflect the way one would evaluate manually. Table 3.1 presents an overview of the LMS appropriate question types and a conclusion on whether or not they are suitable to use in an auto-grading context. The conclusions marked with an asterisk (*) come with certain reservations, discussed below.

MC, *MR*, *either/or*, and *select from a list* are all straightforward to evaluate automatically, as they all have a well-defined assessment model that will be equivalent for any candidate system, with no potential pitfalls. *Short answer* is suitable as long as the question is kept concise and unambiguous; questions like "What does this function do?" is near impossible to predict correct answers to, and would require an evaluator to inspect the answers manually, as they would have to in *open answer* questions.

Fill in the blank and *order* would without a doubt be suitable if their evaluation models were improved. With that said, it is important to note that we have only had the opportunity to test these question types on itslearning. For both of the question types, we have discussed alternative evaluation approaches that we believe would surpass the assessment models used by itslearning with regard to mirroring a manual

Question Type	Suitable
Multiple choice	Yes
Either/or	Yes
Multiple response	Yes
Open answer	No
Short answer	Yes*
Fill in the blank	Yes*
Select from a list	Yes
Match	Yes*
Order	Yes*
Hotspot	Yes*

Table 3.1: Evaluation of suitability for LMS questions.

evaluation by a human grader. Luckily, the reluctance we have experienced might not be a great setback for future work, as Blackboard will soon replace itslearning as the acting LMS at NTNU.

Finally, when it comes to *match* and *hotspot* questions, they both have a well-defined assessment model suitable for automatic grading. However, our concerns are strictly related to the simplicity of the questions that fit under these question types. That is, the question types are well-suited for automatic evaluation, but the questions will not necessarily be complex enough to belong in a final exam.

3.2 CMB Question Types

We have already identified three programming problem types when discussing trends from previous exams for the TDT4102 course (Section 2.1.4), and we will thus not repeat them here. They will, however, be considered during the suitability analysis in Section 3.2.4.

3.2.1 Complete the Code

Complete the code problems are appropriate when the problem text requires an implementation to be structured in a specific way, for example using a certain method or data structure. Figure 3.17 shows an example of this problem type (assume the problem text also includes a detailed description of how the sorting algorithm works). The respondent is required to implement the *bubble sort* algorithm [Ast03] by adding the missing code where it is indicated. There are a number of

ways to implement this, but the example shows how the problem creator can make sure the students solves the problem specifically with the use of nested `for`-loops. It is also necessary for the student to understand the given code, which is in direct compliance with learning objective **S5**.

Q Complete the following code that is supposed to sort a `vector` of integers using the bubble sort method.

```
1 void bubblesort(vector<int> * list) {
2   for (int i = 0; i < list -> size(); ++i){
3     for (int j = 0; j < list -> size() - 1; ++j){
4       // YOUR CODE HERE
5     }
6   }
7 }
```

Figure 3.17: Example of a *complete the code* problem.

Code integrity

On CMB there is no way of making sure the respondent has not altered the provided code, which directly opposes the intentions of the problem type. A solution to this issue that does not involve changes to the CMB system is to make sure that the effort of completing the provided code is less than the effort needed to solve the problem from scratch. Thus, the problem to be solved must be sufficiently complex with regard to finding a solution without a nudge in the right direction.

3.2.2 Fix the Code

Fix the code problems are seemingly similar to *complete the code* at first glance. However, the main focus lies more in the fine details of programming. Respondents need to study the given code, which is disguised as a complete program, and locate the errors (syntactic or logical). With this problem type, one can test the students in the more specific parts in a program, such as potential pitfalls or commonly overlooked elements. Figure 3.18 shows an example where the respondent is required to fix an attempted implementation of a function that should return `1` if the input is a prime number and `0` otherwise.

Our example contains four errors, where two of them are logical errors which result in wrongful behavior and the last two cause runtime errors. The first error is found on line 3; the description clearly states that negative integers (including 0) are defined as primes. The second error can be found on line 5. Since the numbers 1 and 2 are both primes the return value should be `1` and not `0`. The third (and perhaps the

most obvious) error is on line 8, where there should be a `for` identifier instead of `while`. The last is on line 15, where `i` should not initially be 0 because it implies zero-division. Preferably, the initial value should be 3.

Q Below is an attempt at implementing a function for finding prime numbers. The function takes an integer as input and should return `1` if it is a prime number and `0` otherwise.

```
1  int is_prime(int n) {
2    //non-positive numbers are not defined as prime numbers
3    if(n < 0) return 0;
4    // 1 and 2 are defined as primes
5    if(n <= 2) return 0;
6    // No even numbers are primes
7    if(n % 2 == 0) return 0;
8    while (int i = 0; i < sqrt(n) + 1; ++i){
9      if(n % i == 0) return 0;
10   }
11   return 1;
12 }
13
14 int main() {
15   for (int i = 0; i <= 100; ++i){
16     if(is_prime(i)) cout << i << endl;
17   }
18 }
```

Figure 3.18: Example of a *fix the code* problem.

To implement this problem type on CMB, one would give the respondents the faulty code (which would not be accepted by the system), and they would have to fix it in such a way that the system accepts the solution.

3.2.3 Topic Specific Programming Problems

In this section, a selection of important topics in programming are described, and their suitability with regard to online judge system is discussed.

Principles of object-oriented programming

Object-oriented programming is a widely used programming paradigm which is based on the concept of *objects*. Many universities offer courses completely dedicated to the subject, most likely due to the fact that there is a substantial amount of programming languages that support it and rely heavily on it. It is also a vast paradigm containing other frequently used mechanisms such as *inheritance*. A typical object-oriented assignment is exemplified in Figure 3.19.

> **Q** In this task you will implement a class, `WDiary`, which is intended to function as a workout diary. The user must be able to create, save and delete `Entry` instances in the diary. Each `Entry` must contain a `date` and a comment related to the workout. Each `WDiary` has an instance of `Person` called `owner`, containing a name, current weight and height, and methods for accessing this information and changing the weight. Additionally, each entry can store other users who joined the workout if any. Since this is not an actual publicly available application, we can assume all existing users are accessible and can be added without their permission.

Figure 3.19: Example of an *object-oriented* problem.

The task of customizing an object-oriented programming assignment suitable for online judge systems is not straightforward. Since one cannot detect the object-orientated structure of a code submission based on the input or output, the online judge must be able to examine the code itself to evaluate the use of object-orientation, i.e., with the use of pattern recognition.

Operator overloading

A key feature in C++is *operator overloading*, which makes it possible for common operators such as arithmetic operators and the stream operator (`<<`) to work on custom object types. A typical assignment involving operator provides the respondent with a class definition and asks for the implementation of one or more operators.

Another example is presented in Figure 3.20, where one should re-define the addition and subtraction operators for the `string` type.

> **Q** Everything that can be represented using a `string` contains a non-negative number of characters. Overload the operators for the type `string` so that the result when adding a string to another should be the number of total characters. Equivalently, subtracting two strings should result in the difference in the number of characters in the two strings.

Figure 3.20: Example of an *operator overloading* problem.

Similar to the issue with object-oriented programming concepts, operator overloading is difficult to validate on an online judge such as CMB as there is no guarantee that the respondent has structured the code in the correct manner. However, a solution could again be to provide the students with skeleton code that is so complex and possibly obfuscated that more effort is required to try and find a way around the skeleton code to reach a working solution.

File I/O

Reading from and writing to a file is important in many applications. It is used for permanently storing information, accessing permanently stored information or reading information from an external source. File I/O is an important part of the learning objectives of the course TDT4102.

> **Q** In this task you will implement a permanently stored Top 10 high score list. The list should be stored in an external `.txt` file which your code will read from and write to. The program will take a score as input and correctly insert it in the list if it belongs in the top ten. The list must never exceed ten lines.

Figure 3.21: Example of a *file I/O* problem.

We have not come across any online judge systems, including CMB, that support file operations. Thus, code inspection is required in order to properly evaluate this problem type.

Exception handling

Exception handling is a concept often used in programming languages to handle unexpected behavior without crashing the program. In the TDT4102 course, exception handling assignments are often related to input sanitation, as exemplified in Figure 3.22.

Q Some important information about a car is its owner, mileage and model year. Given the class *Car*, implement its constructor which should properly set said information with the following restrictions:

- Every registered car has an owner (i.e., the owner field cannot be `null`).
- The mileage must be non-negative and be of type `int`.
- One can assume the model year is between than year 1899 and current year.

Implement appropriate exceptions for each of these cases.

Figure 3.22: Example of a problem using *exception handling*.

Programs uploaded to online judges are commonly quite simple and do not have complex function call stacks, which is one of the reasons to include exception handling. On a typical exam, exception handling simply involves writing output to signify what went wrong in the code. As online judge systems are only based on input and output and do not know anything about structural aspects of the code, again we have the issue where it is impossible to validate whether the respondent has handled the problem using exceptions at all.

3.2.4 Suitability

Table 3.2 show a summary of the programming problem types and whether or not they are suitable for online judge systems. The conclusions marked with an asterisk (*) come with certain reservations, discussed below. An important criterion for suitability of the programming problem types is that the solution should not have a strictly defined structure in order to be accepted during manual evaluation. Alternatively, as we have discussed earlier, it should be possible to create a code skeleton so complex that the student does not bother trying to solve the problem from scratch.

Simple problem solving and *fix the code* are all considered suitable for an online judge system because the structure of the code should not impact the verdict; the central requirement is to solve a problem. Also, it is the problem designer's task to make sure the system can evaluate the problem. When constructing a *simple problem solving* assignment, one must be sure to describe clearly how the input should be processed and the output formatted.

Complete the code, *operator overloading* problems and *object-oriented programming* problems all face the issue of not being able to validate the structure of the submitted

Question Type	**Suitable**
Code understanding	No
Simple problem solving	Yes
Constructing a program	Yes
Complete the code	Yes*
Fix the code	Yes
Object-oriented programming	Yes*
Operator overloading	Yes*
File I/O	No
Exception handling	No*

Table 3.2: Evaluation of suitability for CMB questions.

code. However, they are possible to design in such a way that an online judge system can evaluate them. A method used to make this work for *operator overloading* and *object-oriented programming* problems is described in Section 4.3.

Code understanding are deemed unsuitable for online judge systems because they do not involve programming explicitly, but rather the ability to follow the logical flow of a block of code in order to determine what the output is, or what is supposed to happen. This problem type is more suited in the LMS part of the digital examination, specifically as a *short answer* question as discussed in Section 3.1.5.

Exception handling problems are not considered suitable for online judge systems because one cannot know if the respondent outputs the exceptions correct, i.e., uses `try-catch` blocks and `throw`. `cout` can easily replicate the output from an exception. However, one can, with some creativity, try to ensure the correct use of exception by e.g. providing skeleton code. Hence, we marked the conclusion with an asterisk.

Finally, *File I/O* is not suitable for online judge systems because they require reading from and writing to files, which is seemingly from the experience we have gained from researching online judge systems not currently supported.

Chapter 4
Midterm Experiment

In this chapter, we will discuss all aspects of the midterm experiment conducted in March 2016. First, Section 4.1 states the description of the experiment as it was defined at the beginning of this research project. Section 4.2 delves into the preliminary phase of the project, including partial results from testing the CMB system prior to the experiment. Section 4.3 deals with the assignment used in the experiment. Finally, we discuss the actual experiment execution in Section 4.4, including a description of how we kept the results from the test anonymous.

4.1 Experiment Description

A set of students from the TDT4102 course will participate in the experiment, where they will take a midterm test prepared by us. Seeing as the aim of the experiment is to compare two different exam approaches, the participants will be split into two groups: **Group A** will complete the test in the traditional manner using pen and paper, and **Group B** will complete the test digitally using CMB and itslearning. Ideally, the two groups will be as similar as possible with regard to student program distribution. Splitting the participants into two groups requires a higher number of total participants to ensure that each group consists of enough representatives to make the results feasible; therefore, we have set the ideal number of participants to 100.

In total, two consecutive 45-minute sessions will be at our disposal; sacrificing the 15-minute break, we have 1 hour and 45 minutes during which we must introduce the project, walk the participants through the guidelines of using the CMB system, execute the test itself, manage delivery of responses, and serve food. Based on these circumstances, we set the duration of the test to be 1 hour and 20 minutes.

Unlike the final examination, no aids will be permitted during the test, to ensure that the outcome of the test is not affected by uncontrolled available assistance. Obviously, the group taking the test digitally will have complete access to the Internet; hopefully,

these participants will respect the nature of the experiment and not exploit this circumstance. As the results from the midterm will not count in any way towards their final grade in the course, the participants have nothing to gain by cheating.

As a way of attracting more students, we will order pizza for all participants when the test is done. After completing the test, the participants will also be asked to complete an anonymous questionnaire (attached in Appendix C) which will later be used by the research team in order to derive additional information about the experiment.

4.2 Planning and Preliminary Work

4.2.1 CMB Workshop

Aiming to familiarize TDT4102 students with CMB and online judge systems, a workshop took place in **Week 8**, replacing a regular lecture and consisting of a demonstration and interactive work. Said workshop was led by the research team, as well as a representative from the CMB development team. Also present were two representatives from the course staff, who were both familiar with the system from a user's standpoint but with no additional significance to the CMB project.

In preparation for the workshop, we created and uploaded a handful of exercises of varying difficulty to CMB. Most of these exercises were close to trivial, seeing as the majority of the students were assumed unfamiliar with the programming approach often required by an online judge system.

During the presentation of the system, the students saw a demonstration on how to solve a simple problem; the Prime Numbers problem was used as an example, where one should determine whether or not a number is a prime, and output `1` (true) or `0` (false) based on the conclusion. After the demonstration, the students were left to work for themselves and solve the previously mentioned set of prepared problems.

Originally, we wanted to use the `climb` server for the workshop, seeing as this is the server we would use in the experiment later on. However, we ended up having to use the `climb-dev` server, as `climb` was down for reasons discovered later; this is discussed in Section 4.2.3. These issues probably affected the overall interest for the midterm experiment, which we intended to promote in the workshop.

4.2.2 Participant Registration

By **Week 9**, a registration form for the experiment was created using Google Forms [Goo] and distributed to the students of the TDT4102 course. In the form we asked for the following information (all fields were required):

Full name Full name of each volunteer was necessary to be able to map each participant to their respective username on CMB.

Study program and study year This information was crucial because we attempted to achieve a balanced representation of each class grade and each study program in the two groups used in the experiment, seeing as the educational background of each participant may affect performance. Representatives will also have varying prerequisites from having taken a diverse number of university-level courses.

E-mail address Prior to the experiment, we needed the participants e-mail addresses to enable distribution of information. After the process of splitting the participants into two groups (described in Section 4.2.4), we sent out informative e-mails to the respective groups.

Familiarity with online judges Knowing whether or not the participant has experience with using CMB or a different online judge system was relevant because we wanted to assign these students to the digital group in the experiment. We assumed this would save time on the experiment day, as well as increase the overall success rate of submissions to CMB.

Consession to the use of results Because the plan was to store the results from the experiment indefinitely, each participant had to be informed of this, as well as grant us the permission to do so.

4.2.3 Pilot Run and Testing

In **Week 10**, we administered a pilot run for the midterm in collaboration with the TDT4102 course staff, where the aim was to check whether the test questions were understandable, as well as confirming the timeframe in accordance with the workload. Two representatives from the course staff received the test and completed it digitally with itslearning and CMB. Upon feedback from the pilot participants, we made a few revisions to the test, mainly in the provided source code, where several `#include` statements had been neglected. Both candidates completed the test in less than the allotted time of 1 hour and 20 minutes, endorsing our estimation of the time required of less proficient C++ programmers.

During the same week, a thorough test of the CMB system was completed to ensure

reliability and stability throughout the experiment, so as to avoid the situation we found ourselves in during the CMB workshop discussed in Section 4.2.1. These tests uncovered three important undocumented facts that could lead to problems when submitting solutions on CMB.

Hidden files

When compressing directories on a Mac running OS X, the resulting `zip` includes two hidden files not supported by CMB; `__MACOSX` and `.DS_Store`. Similarly, `zip` files created in Windows may include a hidden `desktop.ini` file, though this is not common. These files are automatically created by the operating system, and must be removed manually. On a Mac, where the issue is consistent over all compressions, the bash commands below can be executed in the Terminal application in order to exclude the unwanted files in a compression.

```
$ rm -rf path_to_solution/.DS_Store
$ zip -rX filename.zip path_to_solution
```

Line endings

Different operating systems have different standards for line endings, a common inconvenience amongst programmers [Jef]. Unix-based operating systems such as Mac OS X and Linux use the `LF` (Unicode `U+000A`) standard, whereas non-Unix operating system, including Windows, commonly operate with `CR LF` (Unicode `U+000D U+000A`). Administrators using Windows machines must make sure that input and output files use the Unix standard for line endings before uploading a problem to CMB, which runs on a Unix server.

No timeout on small correctness test

For a problem entity in the CMB system, the administrator-provided files `small_input` and `small_output` make up the *small correctness test* which is, as described in Section 2.4.1, used to identify potential runtime errors and to validate the problem solving abilities of the submitted solution. A common and highly relevant issue is the infinite loop, which is technically not identified as a runtime error. Therefore, it is important that systems handling user-submitted code are able to recognize and manage infinite loops, where the management aspect often implicates some sort of timeout functionality.

In our case, the issue with line endings described earlier caused infinite loops in certain submissions. Because CMB at the time did *not* handle this circumstance, the system's capacity to run submissions was exhausted, and the server seemingly went down.

4.2.4 Constructing the Groups

When constructing the two groups for the experiment, we considered each participant's *student profile*, defined as a unique combination of the *student program* and *class grade* features; the most common student profile amongst the registered participants was (MTTK, 1). A full list of the represented student profiles after initial registration is listed in Table 4.3.

To ensure equal representation of each student profile in Groups A and B, we went through the student profiles one by one. Unless one of the groups had claimed half of the participants in the current student profile (in which case the remaining participants were placed in the opposite group), the following process was repeated for each student: If a student had declared familiarity with online judge systems, they were automatically placed in Group B; otherwise, their assigned group was decided by coin toss.

Student Profile	**Count**
(BFY, 2)	1
(BIT, 3)	1
(BMAT, 1)	1
(BMAT, 2)	2
(MLREAL, 2)	2
(MTDT, 1)	2
(MTDT, 3)	1
(MTELSYS, 1)	11
(MTENERG, 1)	1
(MTENERG, 2)	6
(MTENERG, 3)	1
(MTFYMA, 1)	1
(MTFYMA, 2)	8
(MTKJ, 4)	1
(MTTK, 1)	32
(MTTK, 2)	18

Table 4.3: Student profile representation.

4.3 Midterm Assignment

Presented in this section is the midterm test we developed and used for the experiment in Week 11. As the students have not been through the entire curriculum, we had to restrict the questions to only deal with subjects up to and including exercise 8 (Table 2.4 contains an overview of the exercises and their subjects). There are two parts to the midterm assignment; Part 1 consists of MC, MR, and short answer questions, and should be completed on itslearning for participants in Group B. Part 2 includes three programming questions that should be evaluated on CMB for Group B. Group A will complete all of the questions with traditional pen and paper. The respondents have 1 hour and 20 minutes to complete the test.

All of the questions presented in this section are summarized in Appendix B, which contains the actual document that was delivered to the participants (in Norwegian, as TDT4102 does not require English understanding).

When creating a test on itslearning, the test creator must configure a set of test properties. Table 4.4 shows the configurable properties and how they were set during the test.

Assessment	Scoring
Scoring method	Without penalty
Question navigation	Free navigation
Number of allowed attempts	1
Max time allowed per attempt	Unlimited (minutes)
Show answer to participant	When the teacher decides
Use feedback	No feedback

Table 4.4: Configurations for midterm test on itslearning.

The reason for setting the *Scoring method* property to *Scoring* instead of *Grade*, which seems like the obvious option, is explained in Section 4.4.2.

4.3.1 Multiple Choice Questions

Either/or questions have exactly two alternative answers, whereas a user can differentiate between an MR from an MC question from the checkboxes, which replace the use of radio buttons when there is only *one* key.

Q1a (5 points) Which of the expressions are `true` after running the following code?

```
int x = 5;
int y = 42;
int* ptr1 = &x;
int* ptr2 = ptr1;
*ptr2 = y;
```

☐ `x == 5`
☐ `x == 42`
☐ `ptr2 == &y`
☐ `*ptr1 == y`

Q1b (5 points) According to the class hierarchy given by the following code, which of the declarations are valid?

```
class A {};
class B: public A {};
class C: public B {};
class D: public A {};
```

☐ `A a = C();`
☐ `C c = A();`
☐ `D d = B();`
☐ `B b = C();`

Q1c (5 points) One of the approaches to inspect whether two C-string variables are equal is to use the `==` operator.

◯ True
◯ False

Q1d (5 points) How can you declare a function to return an `int` array of size `n`?

◯ It is not possible.
◯ `int[] createArray(n)`
◯ `int* createArray(int n)`
◯ They are both valid.

Q1e (5 points) In which cases is the copy constructor of `MyClass` called? You can assume that `myInstance` is an instance of the `MyClass` type.

- ☐ `MyClass newInstance = MyClass(myInstance);`
- ☐ `void myFunction(MyClass c);`
- ☐ `void myFunction(MyClass &c);`
- ☐ `MyClass newInstance;`

4.3.2 Short Answer Questions

In order to make the test compatible with itslearning's automatic grading tool using a list of pre-defined keywords, we had to select short answer questions with a clear, non-negotiable answer.

Q2 (5 points) What is the output of the following program?

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5   int x = 7;
6   int y = 5;
7   int c = y++ + --x;
8   cout << c;
9  }
```

Q3 (5 points) Which `char` is used to terminate a C-string?

4.3.3 MyInteger

The first programming problem is called *MyInteger* (**Q4**). This problem was mainly intended to test proficiency with operator overloading in C++ (**K1** in 2.1.1). Also, learning objectives **K4**, **S1**, **S2**, **S5** and **C1** are partly or completely covered. Some of the header file was given to the students to assist in understanding the intended structure of the code. Group B received a `zip`-file including the unfinished header-file, as well as a complete implementation of the `main`-method that handles the input and output on CMB.

Group B was additionally asked to implement the remaining arithmetic operators `-`, `-=`, `*`, `*=`, `/`, and `/=`. This would seem like a considerable increase of workload, but seeing as the respondents could essentially copy and paste their implementations

Q4 (25 points) In this task you will implement a class `MyInteger`. This class should have a single member variable, `value`, of type `int`, and should also implement the following:

- two addition operators, `+` and `+=`
- the unary operator `-` (used to represent negative numbers, e.g., -10)
- the insertion operator `<<`

Create a constructor for `MyInteger` that accepts an integer value, and implement the required operators. You can use the provided header-file (below) as a starting point; note, however, that you must decide for yourself how the declarations of the addition operators should be defined.

```
1   // MyInteger.h
2   #include <iostream>
3   using namespace std;
4
5   class MyInteger {
6    int value;
7
8   public:
9    MyInteger(int val);
10
11   int getValue() const;
12
13   MyInteger operator-() const;
14
15   friend ostream& operator <<(ostream& cout,
16                               const MyInteger& myInt);
17  };
```

of `+` and `+=`, only having to replace a single character, we did not regard this as a significant disadvantage. This extra requirement was added because the parser we used to validate the submissions on CMB made use of all arithmetic integer operators; had we provided code for the remaining operators in the provided skeleton code, Group B would have the solution and thus an unfair advantage.

Evaluation

As the assignment essentially was to replicate the functionality of the `int` datatype, some measures needed to be taken to make sure the respondents used the custom

`MyInteger` class and not the built-in `int` type. Hence, we created a parser to translate mathematical expressions represented as strings to an expression where the `MyInteger` class represented each of the operands. This parser was included as a separate file (`Parser.cpp`) in the attached skeleton code.

In an attempt to ensure that the respondents could not tamper with either of the attachments and thus discover the program flow, we performed simple code obfuscation [CTL97] where function and variable names were "encrypted".

4.3.4 Split

As opposed to **Q4**, the `split` problem requires a complete implementation from scratch. Hence, **K6**, **S1**, **S2** and **C1** is the most relevant learning objectives because they emphasize the process of creating a solution. As described in the problem text (**Q5a**) the intended purpose of the program is to divide an input text at every instance of a certain character. For Group B, a `zip`-file was provided, containing an implementation of the `main`-method for handling input and output.

> **Q5a** (15 points) Implement a function `split` that should split up a text string into substrings based on a given character. The function should accept a `string` (the text string to be split) and a `char` (the character deciding where to split) argument, and should return a `vector<string>` consisting of the resulting substrings. You can use the attached documentation on `vector` for hints of which methods to utilize in your solution.
>
> Example: A call to the function `split("Programming_is_fun", '_')` should return `vector` consisting of the elements `["Programming", "is", "fun"]`.
>
> *Hint:* `string` *has a member function* `substr(startPosition, length)`.

Evaluation

The *split* problem was the easiest one to adapt to CMB. Except the basic code for getting the inputs from system arguments, nothing was done to adapt the task to the system. The only thing to be aware of is to avoid using the same test input set in `small-input.txt` as in `input.txt`. If the two test sets are equal, the respondents will see the expected result as an error message in case of a logical error.

This potential problem is not considered critical because the respondent would have to guess that the big input set is equal to the smaller input set; nothing about the problem description or other problem aspects should indicate this to the respondents.

4.3.5 Roommates

The final programming problem created for the midterm test was intended to be the most time-consuming and challenging problem. The problem provides a skeleton code for both groups and is divided into three parts. For the first part, shown as part one in **Q5b**, the students must implement the `Person` class and additional methods for accessing fields which, suggested by the header file, should at least include `getName()` and `getRoomies()`. The second part requires the completion of the method `Person::addRoomie()`. For the third and last part the test takers must implement a function that returns a `Person` instance given the name of the person as a string.

In the skeleton code, "roommates" was abbreviated to "roomies" to spare Group A for having to write long function declarations. The task generally seeks to cover learning objectives **K1** through **K6**, **S1**, **S2**, **S5** and **C1**.

Evaluation

While designing the *roommate* problem we experimented with creating a parser that would run text as functions. For example, if a line in an input file is `"increment()"` the program should run the function called `increment()` if it exists. However, this programming feature, called *reflection* [DM95], is unfortunately not supported in C++.

If C++ supported reflection, we could control the entire program flow from the input set. The input could be a set of method- or function calls each with its own prompt (e.g. "John added Paul as a roommate" in `john.addRoomie("Paul")`).

Instead, we provided an extensive skeleton code in the problem description, which would discourage the respondents to solve the problem in another fashion. There would simply not be enough time to solve the entire task in a different way than what was intended.

Q5b (25 points) **Roommates**

1. Implement a class `Person`. An instance of this class should have a name and a list of roommates. Implement appropriate methods to access these characteristics. You can use the following header-file to get you started.

```
// Person.h
class Person{
  string name;
  vector<Person*> roomies;

public:
  Person(string name);
  void addRoomie(Person* roomie);
  vector<Person*> getRoomies() const;
  string getName() const;
};
```

2. Implement the method `Person::addRoommate`. This method should accept an instance of `Person` as an argument and add it as a roommate to the list of roommates. **Remember that if *Person A* lives with *Person B* and *Person B* lives with *Person C*, then *Person A* also lives with *Person C*.** To get you started, we have provided a `for` loop that prevents a potential infinite loop.

```
// Person.cpp
/* Add roommate */
void Person::addRoomie(Person* roomie) {
  /* LEAVE THIS CODE HERE to avoid infinite loop */
  for (int i = 0; i < roomies.size(); i++) {
    if (roomies[i] == roomie || roomie == this){
      return;
    }
  }
  // YOUR IMPLEMENTATION HERE
}
```

3. Implement the function `findPersonByName` that accepts a name as a `string` and a `vector` containing instances of `Person` as arguments, and returns a pointer to the correct `Person` object from the vector.

4.4 Voluntary Midterm Experiment

The actual midterm experiment was executed in the middle of **Week 11**; a thorough description of the experiment can be found in Section 4.1. On the experiment day, we used the following checklist to administer the various tasks that were necessary to complete:

- Triple-check working solutions on CMB
- Add all participants in Group B to private group on CMB
- Manage permissions on itslearning such that only participants in Group B may take the test
- Make all problems *visible* on CMB
- Prepare lists of participants for each group
- Prepare list for anonymization
- Print out at least enough copies of the test for the participants in Group A
- Confirm pizza order

At its peak, the participant count was 89. However, we received a large number of e-mails from students who could not make it to the experiment after all, and so we ended up with 69 registered and finally 65 attending participants. Because we did not expect so many cancellations, we divided the registered candidates into group A and B several days before the actual experiment. Incidentally, the majority of withdrawals came from participants in group B, and thus the final group distribution was somewhat unbalanced, with 36 students in Group A and 29 in Group B.

4.4.1 Experiment Execution

After separating the participants into two groups, we sent out two e-mails (one to each group), telling them which group they had been put in and the implications that followed. In the same e-mail, we asked the participants in Group B to create a user on CMB and submit their username to us, so that we could make the process of adding them to the private group on the experiment day more efficient, and so that they would not have to waste any time during the test.

When the participants arrived on the experiment day, we had printed out lists for both groups so that no one would be confused about which group they belonged to. We spent a few minutes in the beginning of the session thanking all of the participants for their contribution and explaining some rules of engagement with

regard to uploading solutions to CMB, related to the issues that were uncovered in the system tests (described in Section 4.2.3).

During the test, there were a few questions related to the test itself, as well as a few participants from Group B, who had trouble uploading to CMB. A general problem was that they did not follow the rules for which files to upload, as well as the naming conventions of the directories and files; when analyzing the uploads later, we discovered a few `.xcodeproj` and `.vcproj` files (entire Xcode and Visual Studio project files). Near the end of the test, we asked the students who had not managed to upload a solution successfully to submit their files to a representative from the course staff, who was there to help with delivery and anonymization (discussed later in Section 4.4.3).

When the test reached its end, the participants in Group A delivered their answers to two representatives from the TDT4102 course staff who registered their presence and gave them an identification number to be used in the anonymization process. Finally, we served all the participants pizza and distributed the questionnaire per e-mail, based on the list of present students.

4.4.2 Result Extraction

Preferably, the results from the midterm test should be collected from itslearning and CMB separately, and be combined in an Excel-spreadsheet to mirror the current process of grading exams for professors at NTNU. The following is a description of how this can be achieved for each of the systems we used in out experiment.

itslearning

Luckily, itslearning has a feature for extracting test results and other course page information as an Excel-spreadsheet (alternatively, data can be downloaded as a CSV file), with some tuning of the course page's settings. A user with sufficient access rights (for example *Teacher*) must visit the `Settings` page and enable the `Assessment record` feature under `Course properties and features`. After an ended test, the same user can download the assessment record file from the `Status and follow-up` page.

An important note is that in the Excel spreadsheet, the result representation for each test corresponds to the assessment option chosen when creating the respective test. I.e., if the *Grade* assessment is chosen for a test, the spreadsheet will only contain a letter in the range A through F, and not the actual accumulated score per student. Thus, it is beneficial to select the *Score* assessment option when the score ranges for each grade could potentially be changed after analyzing the results (as we had to do, discussed in Section 5.1).

CMB

At the beginning of this project, there was no available method for exporting results from CMB involving Excel or any similar spreadsheet applications. However, finding the desired information was not impossible; any user (admin or regular user) could log into CMB online to see result data in two alternative ways.

The first alternative was to visit a specific problem on the CMB site. Each problem page publicly lists the users who have managed to solve the respective problem (implying that CMB has managed to compile and run the submitted code, and accepted the proposed output) with username and details such as code runtime and energy efficiency. In an assessment context, one can conclude whether the students who are listed have solved the problem in a satisfactory manner based on, for example, the code runtime, or simply have "blind faith" in the system and rewarding the student a full score because the solution was accepted. However, there was no way of retrieving the information of users who have tried to solve the problem but failed. This was a big issue because depending on which approach one wishes to take with regard to automatic assessment, one may wish to differentiate between respondents who have attempted to solve a problem and those who have not provided a solution at all.

A second and superior alternative were to access the admin interface of CMB to retrieve *all* submissions to a specific problem, regardless of whether or not the submitted solution was accepted by the system. These submissions can be downloaded separately and then inspected manually from the `submission` page. This option enables the professor to reward respondents with answers that are close-to-correct; students who have tried but not completely succeeded could be given partial credit if they show knowledge and understanding despite not having finished a complete working solution. However, this option still does not meet our requirement to automated result extraction to an Excel spreadsheet; the professor must still create a dedicated spreadsheet and fill in each cell manually while inspecting submissions.

All of the information we sought was available, all that was missing was a way to extract it into an interim format such that it could easily be converted into a spreadsheet in Excel. Communicating with the CMB development team yielded a specialized function for group creators on CMB, which parses all submissions for all problems in a group problem set and returning it in a downloadable JSON file [JSO]. Based on this data, we developed a Python application we named `json2excel`, which is simply customized to convert the JSON data from CMB to an Excel-file.

`json2excel` outputs an Excel-file where each row, representing a user, contains four fields per problem in the group problem set. These four fields contain runtime, energy efficiency, EDP, and an empty field for the professor or sensor to fill in a score for the

problem. In the spreadsheet, the reader can identify users who have attempted to upload a solution without success, as all of the corresponding problem fields contain a `0`, in contrast to those who have not attempted to solve the problem at all, for which the corresponding fields are empty. This way, the evaluator can decide for himself whether or not to inspect specific users' code for a potential partial score. Documentation for setup and use of `json2excel` can be found in Appendix A.

4.4.3 Ensuring Anonymity

Prior to the experiment, the participants were informed that all results would be stored in an anonymous and confidential manner, to which they gave their consent during registration. To ensure anonymity of the results, a representative from the TDT4102 course staff with no other association to the experiment mapped each participant to a unique identification number. The final mapping was kept unavailable to outside parties, including the research team who performed the evaluation of each test submission.

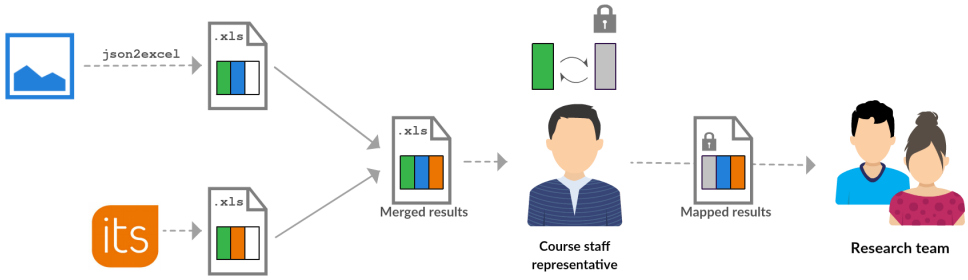


Figure 4.1: Anonymizing midterm results.

Figure 4.1 illustrates the anonymization process. For the CMB results of the experiment, one of the representatives from the course staff ran our `json2excel` script on extracted JSON data from CMB. The resulting spreadsheet was merged with the spreadsheet downloaded from itslearning, and the names in the identifying column of the spreadsheet were replaced with the correct ID number from the previously mentioned mapping before the spreadsheet was forwarded to the research team for the evaluation and grading process.

The anonymization is necessary because if we choose to store sensitive personal data, we are obligated to notify the Norwegian Centre for Research Data (NSD) at least 30 days prior to the experiment [Nora]. In combination with the fact that storing personal data was not a necessity for the experiment we found it more convenient to perform the anonymization.

4.4.4 Questionnaire

After the experiment, we distributed a questionnaire so that the participants could supply their opinions and feedback with regard to the use of digital tools in an examination setting. The questionnaire was later analyzed to answer some of the hypotheses presented in Section 1.3. A Norwegian copy of the questionnaire can be found in Appendix C, and the insights gained from the submitted answers are presented in Section 5.2.

Chapter 5
Results

In this chapter, we present the results from our conducted midterm experiment presented in Chapter 4. First, in Section 5.1, we describe our scoring strategies for the two parts of the midterm test, as well as how the students performed, providing a comparison of the average results of the two groups. Then, we present and discuss the answers from the questionnaire we distributed after the experiment in Section 5.2.

Three sets of hypotheses are presented throughout this chapter, one for each of the two parts of the midterm test (multiple choice/short answer and programming part) and one for the questionnaire. By analyzing the results, we are seeking to determine whether the use of the CMB system is more suitable than the current way of evaluating code. To do this, we make use of the null hypotheses introduced in Section 1.3.

5.1 Midterm Results

For the voluntary midterm test, the maximum achievable score was 100 points; the first part including a set of MC, MR, and short answer questions worth 35 points, and a programming part worth 65 points. Overall, the students assigned to Group A got an average total score of 49% whereas the students who were assigned to Group B got an average total score of 53%. All solutions submitted by Group B not accepted by CMB were assessed manually with the same criteria as for Group A.

Optimally, all test results at NTNU should be close to a Gaussian (normal) distribution [Inf] with C as the average grade. The average score for the two groups combined was 51% which in accordance with the grade scale defined by NTNU is equivalent to the grade E [Nord].

In our attempt at normalizing the grade distribution, the point range of the lower section of the grades (i.e. F, E, D, and C) were adjusted as shown in Table 5.1. The adjusted ranges make the assessment more favorable to the majority of the

participants. Figure 5.1 shows the original grade distribution which is dominated by grades in the bottom half which could indicate that the test may have been too challenging. The normalization makes the score more equitable and comparable with other tests, as well as giving the students a more suitable impression of their performance [Win]. Also, since we are inexperienced test creators, some tasks or subtasks might have been assigned more or fewer points than they should have, based on their level of difficulty and significance. After the adjustment, the overall result is normalized and in our opinion reasonable. Figure 5.2 shows the result after the adjustment.

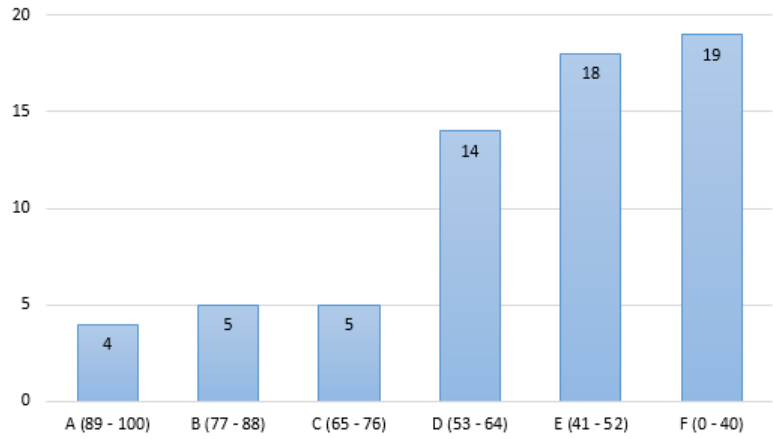


Figure 5.1: Grade distribution from the midterm test.

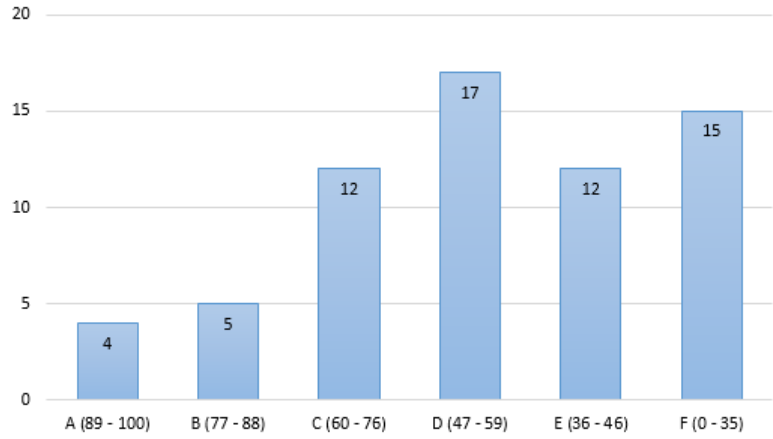


Figure 5.2: Grade distribution after an attempt at achieving a Gaussian distribution by adjusting the point ranges.

Grade	Adjusted Range	Default Range
A	89 – 100	89 – 100
B	77 – 88	77 – 88
C	**60 – 76**	**65 – 76**
D	**47 – 59**	**53 – 64**
E	**36 – 46**	**41 – 52**
F	**0 – 35**	**0 – 40**

Table 5.1: Point ranges.

5.1.1 Part 1 – Multiple Choice and Short Answer

The first part of the midterm test is the easiest part to assess, as all of the utilized question types have a clearly defined evaluation model, ensuring unbiased assessment. Because there is no partial credit to gain, this section of the test is evaluated the same way manually as it is assessed digitally (in this case by itslearning). The maximum achievable score in this part was 35 points (5 points for each sub-question).

The students' performance will help us accept or reject the following null hypothesis:

$H0_1$: Having a computer when answering multiple choice questions is not an advantage with regard to performance.

Scoring the Multiple Choice and Multiple Response Questions

In our midterm test, the applied scoring approach was decided by itslearning, where the only configurable option was to use negative marks for incorrect answers (which we chose not to do). As described in Section 3.1.3, itslearning uses a dichotomous scoring model for MC (no partial scoring), and a polytomous model for MR. Table 5.2 shows an overview of the total score for a MR question based on how many of the alternatives were correctly or incorrectly checked; here, **correct** means that a key was selected or that a distractor was *not* selected, and **incorrect** means that a distractor was selected or that a key was *not* selected. The maximum achievable score for each question was 5 points, and all MR questions had exactly four answer alternatives.

At NTNU, the usual standard in MC-like exams is to include negative marking for wrong answers. As explained previously, this is to balance out potential points gained by guessing where the respondent did not know the correct answer. However, we chose not to penalize wrongful answers with negative marks because the end result would be more complex to analyze; the aim of the experiment was to test the CMB

# Correct	# Incorrect	Score
0	4	0
1	3	1.25
2	2	2.5
3	1	3.75
4	0	5

Table 5.2: Multiple response scoring with four answer alternatives.

system as a tool for programming examinations, and not the students' proficiency in C++.

Scoring the Short Answer Questions

For the short answer questions, we utilized itslearning's automatic scoring tool by storing keywords corresponding to the accepted answers. As mentioned, we had to select questions with clear, non-negotiable answers, in order to avoid the risk of forgetting crucial keywords in the test framework.

Q2 was a code understanding question where the student was expected to submit the correct output; the question has exactly one correct answer and could yield 0 or 5 points.

For **Q3**, the correct answer is `'\0'`, but the question could prompt optional answers that would also be correct. Thus, `\0`, `null-char` or `null-character` also yielded a full score of 5 points. Some students answered `'0'` or `'/0'`, but we decided to not award partial score to these attempts, as they are ultimately wrong. We did not account for typographical errors, as this would require manual inspection of each digitally submitted answer.

Analysis of Results

Starting with the multiple choice part we see a widespread in scores. The average score from the multiple choice questions of the students that solved the problems using pen and paper were 48% (16.84 out of a possible 35). We expected the results of the multiple choice part to be close to equal for the two test forms seeing as it is just a matter of choosing the correct alternative; the medium in which the students used to solve the test should not affect performance. However, the average scores of the students that solved the multiple choice problems digitally was 63% (22.05 out of a possible 35). Figure 5.3 illustrates the average score for the two groups for the first part of the midterm test.

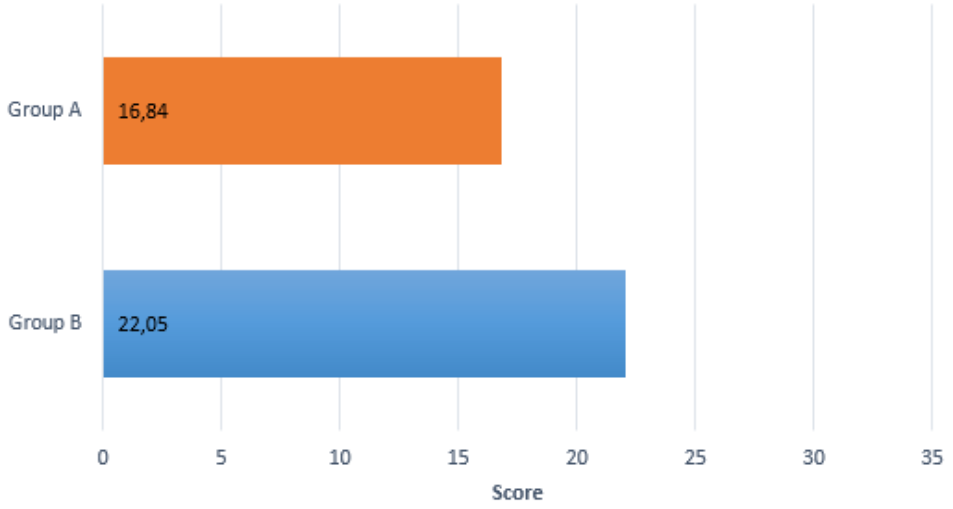


Figure 5.3: Average score achieved on the multiple choice part.

Hypothesis Conclusion

The process of testing hypotheses mentioned in Section 2.5 will be used when testing all hypotheses, starting with $H0_1$. This hypothesis involves the two sets of results from Group A and Group B from the first part of the midterm test, of which the average scores are shown in Figure 5.3. The results from the F-test suggests using the T-test assuming equal variances. We initially expected the results to be close to equal; we expect the mean of the results in Group A (μ_A) to be equal to the mean of the results in Group B (μ_B). The T-test yields a relatively small two-tail p-value, $0.001 = p < \alpha = 0.05$, which means the difference between the two results are significant. The two-tailed p-value is used since the alternate hypothesis, the means will be unequal, can be that one mean is either lower or higher than the other.

Conducting the Shapiro-Wilk test shows that the samples can not be considered normally distributed, which means the Mann-Whitney test should be conducted. Finally, the p-value from the Mann-Whitney test is calculated, which supports the results from the T-test. Based on these findings, it seems that $H0_1$ can be rejected. However, further calculations are needed to be more certain and to have a measure of how strong the conclusion of a rejection will be.

Lastly, the effect size is calculated. The two samples have an effect size of Cohen's $d = 0.80$ and Hedge's $g = 0.79$. Based on Table 2.5 we can see that the values are considered large which indicates a strong confidence in our conclusion. Since the effect size is high, we can assume the groups are already sufficiently large (the size of

Hypothesis	$H0_1$	
	Group A	Group B
Mean	47.56%	63.18%
Variance	3.99%	3.62%
F	1.104	
F Critical	1.841	
Variance	Equal	
T-test p-value	0.002	
Normal distribution	No	
Mann-Whitney	0.002	
Reject $H0$?	Yes	
Cohen's d	0.801	
Hedge's g	0.789	

Table 5.3: Mean, variance, F-test, T-test, Mann-Whitney and effect size for $H0_1$.

Group A was 36 and Group B was 29). The number of participants required for a strong effect size is calculated to be 29 for Group A and 24 for Group B.

In conclusion, we can quite confidently reject the hypothesis, $H0_1$, which states that there is no benefit in using a computer to answer multiple choice and short answer questions.

5.1.2 Programming Problems

The programming part is evaluated much differently manually compared to how it is evaluated by CMB. As discussed in Section 6.3, the manner in which one chooses to evaluate the submitted code greatly impacts the end results.

We will analyze the students' performance on the programming part to accept or reject the following null hypothesis:

$H0_2$: Having a computer when answering programming problems is an advantage with regard to performance.

Scoring the Programming Problems

Seeing as we used a team of two people to evaluate the tests, it was necessary with a strict guideline on how to score the programming problems to make the assessment criteria as consistent as possible. To achieve this, we found it most efficient to break

down the three programming tasks into fundamental components, and assign a partial score for each achieved component.

Component	**Points**
Correct implementation of `MyInteger` class constructor	2.5
Correct implementation of `getValue`	2.5
Correct declaration and implementation of the `+` operator	2.5
Correct use of call- and return by reference in `+` operator implementation	2.5
Correct declaration and implementation of the `+=` operator	2.5
Correct use of call- and return-by-reference in `+=` operator implementation	2.5
Correct implementation of the unary `-` operator	5
Correct implementation of the `<<` operator	5

Table 5.4: Evaluation guideline for **Q4** – `MyInteger`.

Tables 5.4 to 5.6 show how we broke down the subproblems in **Q4** and **Q5** to evaluate each component. Because CMB can reject submissions based on minor faults in the code, and **Q5** has three major problems that need to be solved correctly (`split`, `addRoomie` and `findPersonByName`), we chose to upload `split` as a separate problem to lower the acceptance threshold. Submissions from Group B candidates that were approved by CMB automatically earned a full score, based on the implication that the problem must be solved correctly in order to be accepted by the system.

Component	**Points**
Iterate through the input-string using correct `for`-loop syntax	2.5
Correct insertion to `vector`	2.5
Omit the `split` character during insertion	5
Remember to insert the last word of the input-string	5

Table 5.5: Evaluation guideline for **Q5a** – `split`.

Component	Points
Correct implementation of `Person` class constructor	2.5
Correct implementation of `getName` and `getRoomies`	2.5
Add input `roomie` to your own list of roommates in `addRoomie`	2.5
Add input `roomie` to your current roommates' lists in `addRoomie`	5
Make the recursive call `roomie->addRoomie(this)` in order to add yourself and your roommates to input `roomie`'s and its roommates' lists	2.5
Correct logic in implementation of `findPersonByName`	7.5
Correct use of pointer syntax in implementation of `findPersonByName`	2.5

Table 5.6: Evaluation guideline for **Q5b** – `Roomies`.

Analysis of Results

When it comes to the programming problems the two groups performed almost identically. The students in Group B achieved an average score of 48% (31.2 out of a possible 65 points) on the programming part exclusively, while the students in Group A got an average of 49% (31.85 out of a possible 65 points). It is worth mentioning that the observed similar performance is despite the fact that more of Group B's answers were given zero points than for Group A. As many as 32% of the answers from the digital test were given zero points. For the handwritten answers, only 10% of the answers were given zero points.

Some of the zero-point answers came from students who did not submit their code. There were cases where students submitted a Visual Studio project file which did not include any source files. Some files we received were corrupted. Also, we suspect that some students forgot to send their code by e-mail if they were unable to upload their code to CMB. In these cases, we could not give partial credit.

If the programs submitted by any of the participants were accepted by CMB, they were immediately given a full score for that problem. Because CMB does not reward partial credit, the sensors have to examine the code that did not yield any score. If the examination results were completely automatic with no partial credit rewarded by manual inspection, the result achieved by the Group B would be significantly worse; the average score would then be 19% compared to the original average of 48%.

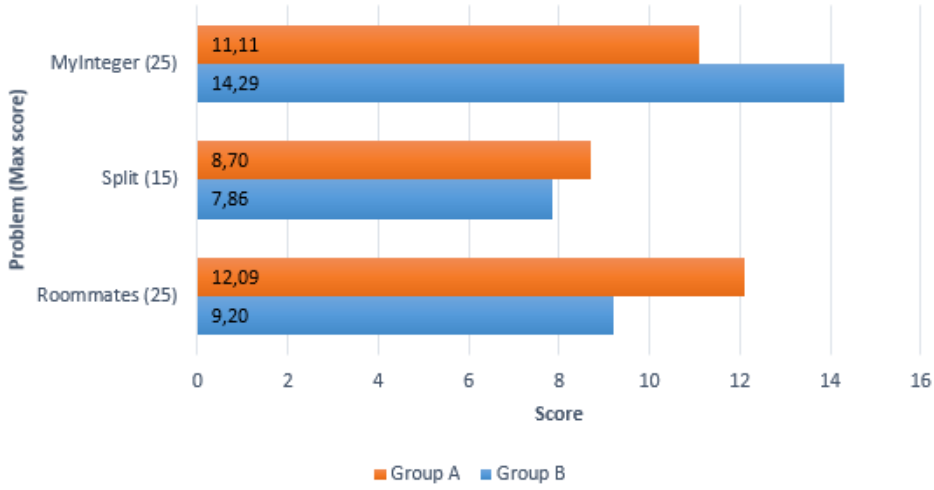


Figure 5.4: Average score given to the programming problems

Figure 5.4 shows the average score distribution between the programming problems.

Hypothesis Conclusion

Hypothesis	$H0_1$	
	Group A	Group B
Mean	49.38%	47.88%
Variance	8.06%	5.46%
F	1.475	
F Critical	1.799	
Variance	Equal	
T-test p-value	0.816	
Normal distribution	-	
Mann-Whitney	-	
Reject $H0$?	No	
Cohen's d	-	
Hedge's g	-	

Table 5.7: Mean, variance, F-test, T-test, Mann-Whitney and effect size for $H0_2$.

As with the previous testing, the variances are compared using the F-test to determine what T-test is appropriate. From the F-test, we conclude that we should continue

using the T-test assuming equal variances to compare the two data sets. Again, we look at the two-tail p-value. The results show that the p-value is relatively high compared to the α-value, $0.82 = p > \alpha = 0.05$. Based on these findings we cannot reject the hypothesis, $H0_2$, stating that the two groups will perform equally. Since there is no strong indication that $H0_2$ is false there is no point in doing further calculations. This result is further discussed in Section 6.3.

For comparison, all the answers that received zero points were removed in both groups and analyzed again. The alteration was done because we suspected that Group B's performance was greatly reduced due to confusion with the process of submitting. The new data sets were analyzed the same way as the others. The new means was 69.55 % for Group B and 53.55 % for Group A. The T-test now shows that there is, in fact, a significant difference between the two mean with the Mann-Whitney test supporting it. The effect size can be considered strong with a Cohen's d of 0.83 and a Hedge's g of 0.82. The strong effect size indicates that the number of respondents, which is in this case adjusted to 35 in Group A and 26 in Group B, is more than enough. Still, we calculate the minimum number of participants needed for a strong effect size to be 28 in Group A and 21 in Group B. In other words, based on this analyze we can confidently reject the hypothesis, $H0_2$, stating that there is no advantage in using a computer to solve programming problems.

Table 5.8 shows the numbers from the different tests conducted. The mean and variance are represented as percentages because of varying total score among the respondents after some of their answers were ignored.

5.2 Questionnaire

After the participants finished the midterm test, they were asked to complete a questionnaire concerning their view on the use of CMB in general, how CMB affected their performance if they used it and how the process of solving the test went, regardless of method. The questionnaire is found in Appendix C.

Our goal with the questionnaire is to accept or reject the following null hypotheses:

- $H0_3$: The use of CMB is equally efficient as the current method of solving an exam.
- $H0_4$: The use of CMB is equally motivating to use as the current method of solving programming problems.

Figures 5.5 to 5.10 show charts for some of the questions given in the questionnaire where a score of 1 represents the most disagreeing response, 3 is a neutral response,

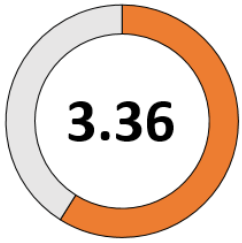
Hypothesis	$H0_1$	
	Group A	Group B
Mean	53.55%	69.55%
Variance	19.26%	19.39%
F	1.002	
F Critical	1.897	
Variance	Equal	
T-test p-value	0.002	
Normal distribution	No	
Mann-Whitney	0.003	
Reject $H0$?	Yes	
Cohen's *d*	0.828	
Hedge's *g*	0.817	

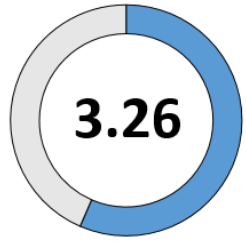
Table 5.8: Mean, variance, F-test, T-test, Mann-Whitney and effect size for $H0_2$ after removing answers that received zero points.

and 5 represents the most agreeing response. The charts show the average response given from the participants divided into the two groups.

Group A Group B

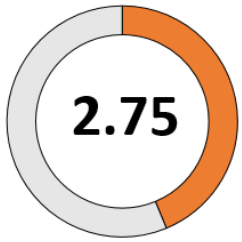
Figure 5.5: To what degree were you able to demonstrate your knowledge in C++ with regard to your assigned form of evaluation?

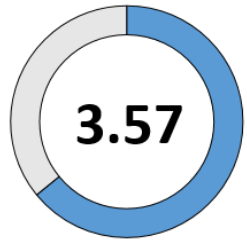




Group A Group B

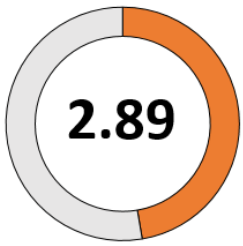
Figure 5.6: To what degree were you able to demonstrate common methods/techniques in C++ with regard to your assigned form of evaluation?

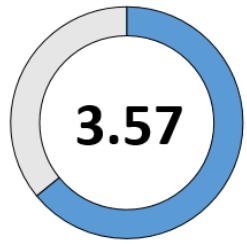




Group A Group B

Figure 5.7: To what degree would you say that your assigned form of evaluation is an efficient way of solving programming problems?





Group A Group B

Figure 5.8: To what degree would you say that your assigned form of evaluation is an motivating way of solving programming problems?

Group A Group B

Figure 5.9: To what degree would you say that your assigned form of evaluation is a suitable way of solving programming problems?

Group A Group B

Figure 5.10: To what degree would you say that your assigned form of evaluation is a suitable way of being evaluated in programming problems?

Hypothesis Conclusions

Analyzing the questionnaire we attempt to accept or reject the two last hypotheses. First, we want to determine whether the use of CMB is more efficient than the current way of taking an exam. To do so, we will see if $H0_3$, which states that the two methods are equally efficient, can be rejected. The hypothesis involves the answers to the question shown in Figure 5.7.

The F-test suggests conducting the T-test assuming equal variance. Since it is already known that the average response indicates that $H0_3$ can be rejected, the one-tail results will be studied. I.e., if there is a significant difference we can assume that using the CMB system is more effective.

The T-test outputs a one-tail p-value, $p = 0.001 < 0.05 = \alpha$, which indicates that the hypothesis can be rejected, but further testing is desirable to strengthen the conclusion.

Hypothesis	$H0_3$		$H0_4$	
	Group A	Group B	Group A	Group B
Mean	2.750	3.565	2.893	3.565
Variance	0.787	0.893	1.358	0.893
F	1.135		1.520	
F Critical	1.950		2.004	
Compared Variance	Equal		Equal	
T-test P-value	0.001		0.015	
Normal distribution	No		No	
Mann-Whitney	0.0038		0.024	
Reject $H0$?	Yes		Yes	
Cohen's d	0.895		0.641	
Hedge's g	0.884		0.624	

Table 5.9: Mean, variance, F-test, T-test, Mann-Whitney and effect size for $H0_3$ and $H0_4$.

Same as before, the normality of the answers must be tested to determine the necessity if a Mann-Whitney test. The Shapiro-Wilk test states that the answers are not normally distributed, ergo the non-parametric Mann-Whitney test is needed. With a p-value, $p = 0.004 < 0.05 = \alpha$, the results supports the conclusion to reject from the T-test.

Next the strength of the conclusion that $H0_3$ will be rejected will be calculated by determining the effect size. Both Cohen's d and Hedge's g is relatively high and can be considered a large effect size with Cohen's $d = 0.89$ and Hedge's $g = 0.88$. The number of participants needed in the two groups was calculated to be 23 in Group A and 19 in Group B. The actual number of students who answered the questionnaire were 28 from Group A and 23 from Group B.

Based on these findings we can conclude with confidence that the use of the CMB system is more efficient than the current way of taking an exam.

The final hypothesis tested in this chapter, $H0_4$, involves the two sets of answers to the question shown in Figure 5.8.

Just like for the previous hypothesis, the T-test is conducted assuming equal variances based on the result from the F-test. The T-test suggests to reject $H0_4$ based on the relatively low p-value, $p = 0.015 < \alpha = 0.05$. Since Shapiro-Wilk's test states that the answers are not normalized, Mann-Whitney's test is conducted and supports the

conclusion from the T-test.

With a Cohen's $d = 0.64$ and Hedge's $g = 0.62$ the effect size can only be considered to be medium. Hence, calculating the size of the two groups needed to achieve a sufficiently powerful conclusion can be useful. Based on those calculations, the size of Group A would have to be 46 people and 38 for Group B, which means that 15 more participants for both groups were needed.

In conclusion, it is likely that the use of the CMB system is more motivating than using pen and paper, although it is not as certain as the previous conclusions.

Chapter 6
Discussion

In this chapter, we reflect on the reliability and validity aspects of CMB as an evaluation system in Sections 6.1 and 6.2. Section 6.3 contains a discussion of the midterm results, including our thoughts on why the results turned out the way they did. Section 6.4 concerns the questionnaire that was distributed after the midterm experiment. Finally, we present thoughts on positive and negative experiences from the project in Sections 6.5 and 6.6.

6.1 Reliability of CMB's Assessment

As previously mentioned, one of the primary motivations behind the CMB system is to remove the uncertainty of a human grader to improve the reliability of assessing final examination on a university level. Reliability can be described as the accuracy of an assessment where irrelevant factors do not influence the evaluation. These factors, such as mood, time of day, or personal bias, does not exist within the CMB system. Even objective evaluations (e.g. multiple choice) may be wrongfully assessed by a human grader, discussed in Section 6.1.1.

Equivalent submissions to CMB will yield the same result. However, this assumption is not based on anything other than the concept of online judge systems, where one expects this to be the case. No tests have been formally conducted to validate this, but seeing as CMB is a small application running on a specific server, this must be true because all submissions are run on the same machine.

We will see in Section 6.2 that the reliability of CMB alone does not make it a perfect assessment system.

6.1.1 Threats to Reliability

In our experiment, we experienced an error that would not happen in a fully automatic system: Two of the identification numbers were swapped in the spreadsheet containing

the results, which led to an inconsistency between the submissions we received after the anonymization and the result extracted directly from itslearning and CMB. Fortunately, we got a sense that something had gone wrong when we went through every solution submitted to CMB that had not get accepted. One submission was registered with a suspiciously high score based on the content of the submission, which was investigated and consequently, the mix-up was discovered. With that said, the mix-up had happened during the anonymization process. Hence, this was a human error; had the anonymization process been fully automated the mixup would probably not have happened.

Today, programming exams are written by hand, and it is sometimes possible to identify a person based on handwriting. Although it may not be relevant in an examination situation, it might reveal some demographic factors about the respondent; depending on the course, the students can choose to answer in either Norwegian, Swedish, Danish or English [oSNc]. In addition to this, Norwegian has two written standards. All these different factors can contribute to identifying a student, which may affect the examiner. For example, if a class of 30 students contains one exchange student from outside Scandinavia, the teacher can be quite certain that the one person who answers the exam in English is the exchange student. Even if the evaluator is unable to relate a submission to a specific individual, the examiner may be able to determine the gender based on handwriting, and treat the submission based on this.

Because the system is "blind" of the submitted code and only takes the output into account, a human grader has to check the code manually if necessary and give credit based on it. Although all submissions are written generically on a computer in a universal programming language, all human grading needs to be removed to achieve the most reliable system possible.

6.2 Validity of CMB's Assessment

As described by Wynne Harlen [Har05], «validity refers to what is assessed and how well this corresponds to the behavior or construct that it is intended to test or assess». In other words, the final assessment should reflect the students' understanding of the subject as a whole, and not simply the final answer to each question. Validity ensures that the given answer to each task in a test is deducted from appropriate reasoning according to a particular curriculum. If a student gives the correct answer but backs it up with incorrect statements, the answer should be deemed invalid and be given no or partial credit.

The problem with CMB is the validity of its assessment. It can only verify that the resulting output of a student's code is correct or not, regardless of how the output was generated. Because of this, programming tasks that focus on e.g. specific

programming methods, architectures or techniques are significantly harder to design to work as desired on the CMB system. The problem designer needs to take measures to avoid that the students simply print out the correct output without making use of the elements of the problem on which they were supposed to be tested. Also, since there is no partial scoring, a submission that is close-to-correct and a submission that is nowhere near correct would both currently receive the same zero score.

6.2.1 Threats to Validity

Our biggest concern when creating the programming problems was to ensure that the respondents could not find another way of solving the problem without making use of the essential elements which were to be tested. For example, given a problem that requires the respondents to create a class, *Person*, and create several instances of it with names given as input. To check if the respondent has done this correctly, one could be tempted to iterate through the `Person` instances and print out the `name` field. In this (simplified) case, one could directly output the names coming from the input, without any further processing.

Another potential concern, based on the same issue that the submitted code is treated as a black box, is that the students could make use of external libraries to solve subproblems that were supposed to be solved using logic skills.

In the questionnaire we asked the students which form of examination they thought would be the most vulnerable with regard to cheating; 50 out of 51 respondents agreed that the CMB system is more susceptible to cheating. We were not very concerned about cheating during our experiment because the students would not personally gain anything by doing so. Instead, we chose to trust the participants to respect the validity of the experiment result. In the future, when the system used for digital programming examination, cheating by using the Internet will hopefully not even be a subject of discussion due to the use of security frameworks such as SEB. If anything, machine-written text on a screen in front of a student compared to handwritten text lying flat on a table might make it easier to read other students' answers during an exam.

6.2.2 Attempts at Improving Validity

While designing our midterm, we found ways to eliminate the possibility of defying the intended code structure of the more comprehensive problems. One of the attempts is found in the first programming task in the midterm test, where the students receive a skeleton code for parsing text into mathematical expressions. All variables and functions were given encrypted names to make it harder for the students to skip the parsing and simply print out the answers to the expressions. This approach does

not guarantee that the students do not understand the encrypted code, but it would take a considerable amount of effort to exploit it.

Another attempt at improving validity is to give the respondents enough help (e.g. skeleton code or complete functions) that most alternative implementations other that the intended would be a more challenging choice. This approach is practiced in the last exercise in the midterm test, where the students received a class definition, incomplete and empty functions and header files, and were instructed to complete the code in order to make it run the intended way.

A third alternative is to split the task in such a way that when the test taker reaches the point where the final program segment that completes the entire program needs to be implemented, any other implementation that would yield the same output would be unnecessary because most of the code is already finished. The problem maker leads the students to solve a bigger problem in a certain way. This method is similar to the example above, but in a way the participants are creating the skeleton code on their own.

6.3 Midterm Results

Before the hypothesis testing was conducted, the significant difference in the multiple choice results of the two groups (as shown in Figure 5.3) was considered to be coincidental, but we suspected that some participants may have cheated by checking for answers online or by testing the theory questions in an IDE. On the other hand, we find it strange if some would break the given rules since their performance does not influence their final grade in the course. We were suspicious because there are no apparent reasons why there should be such a significant difference. Since the hypothesis stating that using the CMB system is equally efficient as solving the midterm using pen and paper is false, one cannot conclude that the difference is coincidental. More participants in Group B might have had spare time reasoning on the questions before or after they completed the programming questions.

An explanation could also be our method of separating the participants, which was supposed to result in two equal sized groups with approximately equal skills. As mentioned in Section 4.4.1, the participants did not end up evenly distributed between the test forms, with a larger amount of participants in Group A (37 participants against 28 in Group B). Also, there was no way of knowing that the two groups were on average approximately equally experienced programmers; to ensure this we would need to significantly increase the number of participants in the experiment.

In the second part of the midterm test, the achieved average score of the two groups was almost identical. Before the execution of the midterm test, we were

confident that Group B would perform better than Group A because for three reasons: Firstly, due to it being the way of programming that closely resembled their most used programming environment. Second, the additional feedback and possibility of debugging code should lower the threshold of proceeding from a close-to-correct answer to a completely correct answer. Finally, the ease of editing the code would result in a significantly more efficient use of the respondents' time, which would again result in more time to solve the problems.

When reviewing the submissions for the *split* problem that got the wrong output, we found something rather peculiar. Even with error messages like "Result: 'United States of America' was: 'United States of' ", many of the participants had still not fixed the problem with the code (excluding the last word of the input string). We concluded that there can be two reasons for this; either they did not have the time to fix the issues, or they did not understand the feedback. If they managed to split a sentence at the correct place at the start of the phrase, adding the last remaining word should not be too challenging.

For the hypothesis $H0_2$, we did a test with altered data sets for comparison. The resulting numbers indicate that without the zero point answers, Group B did, in fact, perform significantly better that Group A. Hence, the test results supports the belief that the respondents in Group B did lose a significant amount of points due to problems with submitting their code. In this case, we believe that this conclusion is the closest to describe the performance of the respondents since it is only taking their code into account, not their attempt at submitting it.

6.4 Questionnaire

Group B achieved a similar average score as Group A when rewarded with partial credit, and significantly lower when not. However, based on the responses from the questionnaire, it seems that using CMB and itslearning for evaluation is the preferred option. On average, Group B responded more positively to the evaluation on how their assigned form of evaluation affected their performance. Also, there was a general disagreement to the statement that the assigned form of evaluation affected their performance negatively. For Group A, the opposite can be said about the two questions; their answers leaned more towards their assigned form of evaluation affecting their performance in a negative way.

The fact that both groups seemed positive to digital examination as a form of evaluation strengthens our belief that Group B would perform significantly better if they were more familiar with the system if they had more time to complete the test, and if their performance would actually affect their final grade in the course. One cannot assume that the students would find CMB favorable if they found that

it affected their performance in a negative way.

The use of CMB seems to be more efficient than solving by hand based on the results of the testing of the null hypothesis, $H0_3$. As mentioned earlier, the reason is believed to be the benefits that follow the use of a code editor. Another reason might be that the feedback given by the system helps the students figure out what they are doing wrong, so they avoid grinding on their code for longer periods of time.

Based on the testing of the null hypothesis, $H0_4$, it seems that the use of CMB is more motivating than the traditional evaluation form. The reasons can be a variety of things, and it is certainly a matter of opinion. However, some factors may be recurring amongst the respondents. There might be a great satisfaction in submitting a solution and instantly getting accepted if the solution is correct. Compared to the traditional way where you either receive verbal feedback or have to check online to see the written feedback from a TA (with varying quality), or receive only a grade several weeks after delivering an exam.

For some, the notion of competition, in the form of the high score list, might encourage further efforts in their submissions. However, in an exam context, the high score list should be hidden so as not to reveal information about other respondents.

One part of the questionnaire was only given to the participants placed in Group B, as they specifically concerned the CMB system. One of the questions asked in this part was whether they would prefer CMB over the current form of evaluation for the exercise project in the course, where the overall response was negative. Hopefully, the reason was that they already had grown accustomed to the current form, and the system still was a bit new and overwhelming. Another reason might be that by using CMB for the exercises would remove the interactions with the TAs, on which many of the students rely on with regard to understanding important concepts in C++.

However, when asking whether they would prefer the CMB system over the current form of evaluating exams they were on average more positive, which is a more important result with regard to this project.

6.5 Positive Experiences

Overall, the execution of the midterm experiment went well without any serious problems. Because of an error while updating the production system server (`climb`) prior to the experiment, we considered asking all the participants to create a new user on the development system (`climb-dev`) and solve the problems there. The new update was supposed to solve the issue with hidden files explained in Section 4.2.3. Instead, we told the participants to ask us if they experienced issues with their uploads, and we would come around and help them remove the hidden files. Thankfully, the

system had no problems handling the traffic of many users, as both we and the CMB development team had feared. In the case of a system crash, we would have to inform all participants to switch to `climb-dev`, or solve the test in any IDE, but instead of submitting to CMB they would have to submit it to the course staff representative by e-mail. In the latter case, we would have to submit each solution to CMB in order to get valid JSON data for our `json2excel` application.

Based on the information gathered from the questionnaire, as well as verbal feedback after the midterm experiment, it seems that many students have a positive attitude towards the use of CMB. This is important as the involvement of students is crucial to the project overall.

6.6 Negative Experiences

One thing we regret not doing before or during the test was to spend more time making sure the participants knew exactly how to deliver their code, and how to make sense of the system feedback. Many students did not manage to do this correctly, and we spent many hours uploading their code to CMB after we received the files from the course staff representative who assisted with anonymization. As previously mentioned, some of the participants even sent us submissions that did not consist of any source code at all. Our data would be more consistent if we spent more time distributing this information such that every submission would be valid and accountable. Also, we may not have been clear enough on how the students in Group B should use the CMB system during the examination. Some students did not know they could submit several times and use the system as a way of debugging their code.

After the division into the two groups, some participants canceled their attendance, especially the participants assigned to Group B. These cancellations resulted in an imbalance in the size of the two groups. To solve this issue, we should have waited with the division until perhaps the night before the day of the experiment. In this case, every respondent would have to bring their computer in case they got assigned to Group B. The separation on the day of the experiment would be manageable and efficient with some planning beforehand.

Finally, the biggest letdown from the experiment is that we forgot to distribute the questionnaire before the volunteers left. To make up for this, we sent out an e-mail to all of the registered participants containing a link to the questionnaire, which was created with Google Forms. However, we did not receive feedback from all of the participants; only 51 students submitted answers to the questionnaire, and we lost valuable data.

Chapter 7
Future Work

In this chapter, we present our insights on how the next years could look like for the CMB project, as an online judge and as a tool for evaluation of programming exams. All propositions are made based on results and findings discussed earlier in this thesis. First, Section 7.1 suggests two possible experiments that could be conducted in collaboration with the TDT4102 course. Section 7.2 presents a five-year plan for the project. Finally, we specifically identify possible functional and design requirements for the CMB system in Sections 7.3 and 7.4.

7.1 Future Experiments

7.1.1 Simulation of an Exam

An experiment similar to ours should make a few alterations to both the planning and the execution. Because NTNU is in the process of replacing itslearning with Blackboard as its utilized LMS, one should perhaps do another review of the available question types and the possibilities for automatic evaluation. Alternatively, one could implement a completely new system that supports the question types described in Chapter 3.

After developing a set of questions, a pilot test should be completed in order to confirm that the allocated time is sufficient to solve all of the problems. This is also a perfect opportunity to receive reviews of the content and quality of the included questions.

Registration for the experiment can be done in a similar manner as our experiment, using Google Forms distributed on the course page of itslearning. Because we did not achieve the expected number of candidates, it may be interesting to promote the experiments to students outside of the course (with a certain level of C++ knowledge). Also, the division of candidates should preferably be done similar to the process described in Section 4.2.4, but closer to the day of the experiment. One can use the

checklist provided in Section 4.4 for further guidance of what should be done on the day of the experiment.

It may be a good idea to team up with someone familiar with the course as well as CMB itself. During a test, many questions or situations may arise when students are unfamiliar with new systems, and several people are needed to respond to every request. If anonymization of the participants is necessary, one can recruit an external resource to be responsible for this.

Right before the initiation of the test, it is highly recommended to make an effort to ensure that the students know how to use the system, both the process of submitting and how to use the feedback given by the system for debugging. Preferably, the students should also have gained some experience beforehand, for example in one or more workshops prior to the experiment, or from the exercise project as suggested in Section 7.1.2.

If a questionnaire is to be distributed in order to gain insight subsequent to the experiment, one should make sure that it is presented directly after the end of the test, preferably before food is served, if there is food. For reference, our questionnaire is found in Appendix C; a relevant goal for the experiment could be to gain improved feedback, and analyze the factors that might lead to this improvement. Questions such as the process of submitting or the content of the feedback of the system are highly valuable during the analysis of the results. If many of the participants were confused with the process of submitting and therefore were unable to submit, that would greatly impact the result. If possible, one should also consider placing the experiment closer (but not too close) to the final exam, which would make it more similar to an actual exam situation seeing as the students would probably have gained more experience with the programming language, and the results would better reflect the participants' actual knowledge of the subject. If the students were more comfortable with programming in general, they would hopefully look at the system as the aid in which it is intended, and not an obstacle or further confusion.

7.1.2 Exercises

An interesting experiment would be to implement the CMB system into the exercise project which is completed throughout the semester, perhaps by offering the students the chance to complete an exercise using CMB alone. To increase the number of potential participants the teacher could allow the students to complete the CMB exercise in the place of a regular exercise.

Although the students were not particularly positive about submitting all their exercises to the system, many would probably enjoy the gamification factor. In a

potential experiment, one could tempt the students into participating by rewarding the top students on the high score list in some way.

This suggested experiment is based on the assumptions that the experiment takes place in one sitting, similar to the midterm experiment where all the participants are sitting in the same room solving the problems. One could also allow the students to solve the exercises at home like they normally would do. However, to complete this experiment, it is necessary that a handful of the TAs have sufficient knowledge about the system to be able to help the students. Also, in this case, one must be aware that the students have access to the Internet, and must design the problems with this in mind. Though the intention is to avoid human interference in the evaluation process, there is currently no way of detecting plagiarism on CMB that does not involve manual inspection of submitted code.

7.2 Five Year Plan

During the next five years, more experiments should be conducted with the guidance of the experience described in this thesis. Since NTNU is replacing itslearning with Blackboard as its LMS, the project team should familiarize with the new system. A natural next step is to conduct one or both of the experiments described in Section 7.1 in the near future. Our suggestion is to start with a simulation of an exam, for which this thesis can contribute with the most experience and information. After the execution and analysis of the result, one can utilize the information from both this thesis and the newly gathered experience from the conducted experiment in planning the exercise experiment. Testing the system with an exercise will yield new experiences and valuable insight which will further contribute to the overall project goal.

When the course staff is comfortable with both the system and conducting tests with it, the system can be further integrated with the TDT4102 course. At this point solving exercises using CMB might be a mandatory part of the course. It is reasonable to believe that many students will appreciate an increased focus on CMB in class when their performance on an exercise depends partially on their knowledge about the system.

As of now, the CMB system is not ready to completely replace all evaluation in the TDT4102 course, but in the future when some of the missing functional requirements listed in Section 7.3 hopefully have been implemented, and both the staff and students are comfortable with the system, a complete substitution can be a reality. From thereon, using the system as a part of the final evaluation will be more natural as it would be a familiar working environment for the students.

7.3 New Functional Requirements for CMB

After gaining experiment with the CMB system a number of missing features has come to mind. Some more important than others, and some considered more realistic based on assumed implementation cost.

Code Evaluation

As described in Section 6.2 the problem designer have to be creative to ensure that the students are solving the intended problem. This task would be made significantly easier if the system itself could check for specific elements in the submitted programs. This change would probably have the biggest impact on the system as well as being the most challenging addition to implement. Another feature we could make use of is if one could ensure that any given code is unaltered and still existing. This way one could define the `main` method and therefore know the output is coming from the correct location in the output and formatted the intended way.

Result Extraction

In the admin interface, it should be possible to retrieve a list of submission information for each problem in a problem set, preferably as an Excel spreadsheet. This data should preferably contain the following:

- **User information** such as username, full name, or student identification number.
- **Result status** for each problem in the problem set; *success* or *failure*, depending on whether the submitted response was accepted by CMB or not. An overview of which students have attempted to solve the problem is interesting with regard to checking for almost-correct answers that should gain some credit.

Login via FEIDE

FEIDE is the Single Sign-On (SSO) service used by students and employees at NTNU for access to various internal platforms. Users should be able to log onto the CMB system through this service. This implies that all NTNU affiliates will have a user on CMB.

Additional User Information

When registering as a new user on CMB, a user is only required to enter a username and an e-mail address. In addition to these fields, users should be able to register their full name and student identification number, which is the number used when

evaluating exams at NTNU. These extra fields are to avoid personal bias and subjective assessment.

Additional Feedback

Currently, when uploading a submission to a problem on CMB, the user will receive feedback on code runtime and energy-efficiency. In courses like TDT4102 where one is taught the importance of memory management when programming in C++ , information about memory usage would also be interesting to receive when submitting code to the system.

Scheduled Problem Sets

It should be possible to schedule a date and time for publication of a certain problem set, so as to allow a professor to upload exam problems prior to the final examination date. An alternative feature to this is to be able to make a problem set hidden for all users apart from the creator and perhaps certain other users specified by the creator.

Submission Deadline

If CMB is going to be used for course exercise projects, the system should introduce a concept of deadlines for certain problems or problem sets. If a submission is uploaded after the deadline, it should be flagged as overdue.

Support for Additional Programming Languages

If digital examination using CMB becomes a reality at NTNU, other introductory programming courses such as TDT4100 – Object-Oriented Programming, which teaches Java, should also have the option to use the system.

Plagiarism control

As the intention of the system is to eliminate the manual process of evaluating code, the system needs to be able to detect plagiarism, both from external sources and from other students.

7.4 Design Improvements for CMB

Based on our analysis of the results from the midterm, as well as feedback and personal experiences, we suggest the following design improvements for the CMB system, which may improve the usability of the system for inexperienced users.

Visibility of System Status

Jakob Nielsen discusses heuristics for user interface design, and the first heuristic is visibility of system status [Nie95], which we feel that CMB is currently missing. System status in this context refers to the stages of uploading, compiling and running a problem. Because the server only has the capacity to run one submission at a time, a user who is waiting for the server sees the message "*in queue*", but there is no information of queue position. In an examination setting where there will be multiple uploads at the same time, dynamically updated queue position information would be very useful as one would get an indication of how long one should expect to wait for the acceptance status. Sindre Magnussen suggests a way of implementing this in his master thesis [Mag16].

Improve Feedback Visibility

As discussed in Section 6.3, we suspect that most of the participants were not aware of the fact that they could use the feedback messages as a "debugging" tool. As of now, a user must click a button called *Show Error*, which would seem obvious and visible enough. However, there is no guarantee that a user clicks this button when the submission is rejected. A suggestion is thus that the error message could pop up as a notification post near the top of the screen, similar to other messages that one can hide when they have been read. This way, the user is "forced" to read the error message.

Documentation of Common Errors

During our experiment, some of the students who monitored their feedback for failed submissions did not understand most of the error messages. In order to gain more insight to one's own submission, it may be useful to include better documentation of the most common error messages.

Optional Visibility of High Score List

For each problem page, there should be an option to make the public high score list visible only to an admin user, or the creator of the problem set. When using CMB for an exercise during the semester, the high score list is an important element for motivation; it provides an element of "gamification" and competition to the exercises. However, in an exam context, the list of users who have been able to solve the problem should be hidden to anyone else, as it might have a negative effect on the personal motivation of the students.

Multiple File Upload

Several of the experiment participants had issues with the pre-processing step of compressing a strict set of files before uploading it to the system. In order to make the upload process simpler and more intuitive, a user should be able to upload only relevant files in unknown quantities to the system, instead of just a single `zip`-file.

This suggestion is also relevant for CMB's admin interface. When adding a problem to CMB (discussed in Section 2.4.1), an admin user must upload five files to the system separately; this is quite tedious, especially when adding more than one problem. A multiple file uploader would save a significant amount of time.

Chapter 8
Conclusion

This chapter concludes our thesis. First, in Section 8.1, we will evaluate the subtasks introduced in Section 1.2 and discuss whether or not we feel that they have been fulfilled. Finally, we provide a summary of the thesis, as well as a formal conclusion in Section 8.2.

Four null hypothesis were introduced which were tested to see if their statements involving the means of two data sets could be rejected. Since the hypotheses suggest that two means are equal, one can reject them if one detects a significant difference in the means. If this is the case, one cannot assume that the hypothesis in focus is correct. Below is a summary of the testing of our hypotheses.

Hypothesis	**Description**	**Conclusion**
$H0_1$	Having a computer when answering multiple choice questions is not an advantage with regard to performance.	Rejected
$H0_2$	Having a computer when answering programming problems is an advantage with regard to performance.	Rejected*
$H0_3$	The use of CMB is equally efficient as the current method of solving an exam.	Rejected
$H0_4$	The use of CMB is equally motivating to use as the current method of solving programming problems.	Rejected

* When excluding the zero point answers for both groups, the hypothesis, $H0_2$, could be rejected, otherwise it could not. We believe that the testing of the hypothesis without the zero points answers give the conclusion closest to the truth because of the amount of failed submissions to CMB.

8.1 Evaluation of Subtasks

Each subtask defined in Section 1.2 is evaluated as follows.

R1 *Identify alternative question types that are suited for an autograding system for C++ and other programming languages.*

A complete research study was conducted on LMS-appropriate question types and common programming problem types, was completed and presented in Chapter 3.

R2 *Discuss which of the question types identified in R1 are best suited to the TDT4102 course at NTNU.*

For each question type presented in Chapter 3, we provided a discussion concerning the suitability of each question type. All of the examples presented in the same chapter was related to the curriculum in the TDT4102 course.

R3 *Outline ideas for a realistic 5-year track making the vision "digital exam with autograding" a reality at NTNU.*

Sections 7.1 and 7.2 consists of a set of ideas that can be implemented in the CMB project during the next five years. These ideas are based on the experiment we conducted during the spring semester of 2016.

R4* *Identify how user interface improvements can be implemented to better reach the overall goal, and describe these as prioritized proposals to the CMB project.*

A list of both functional and design improvement suggestions are provided in Sections 7.3 and 7.4.

E1 *Organize an experimental midterm test in the TDT4102 course at NTNU.*

Chapter 4 addresses the voluntary midterm experiment we completed in March 2016 with students from the TDT4102 course. This chapter includes a detailed description of the planning and execution phases of the experiment.

E2 *Develop and assemble a set of questions to be used in the midterm test.*

Based on the question types we researched in Chapter 3, we developed a midterm test to be used in our experiment. The test is presented in Section 4.3, and the Norwegian version which was distributed to the experiment participants is included in Appendix B.

E3 *Administer a workshop to educate potential experiment participants about the CMB system.*

To educate the students in the use of CMB we organized a workshop prior to the experiment. Details about the workshop are presented in Section 4.2.1, which contains the preparation and execution process.

E4 *Perform a thorough system test of CMB.*

Prior to the experiment we tested the CMB system and identified a few issues, discussed in Section 4.2.3.

E5* *Discuss how the experiment results could be correlated with other more traditional exam-like tests with the purpose of assessing the goodness of our approach.*

Initially, the plan for our experiment was to only use the evaluation form of Group B. However, we decided that the results would be more significant if we were able to incorporate the current standard, and compare the two approaches. Thus, this subtask falls under **E1**, and the results are presented in Chapter 5.

I1 *Implement a solution for extracting the complete results from the tests to data in an Excel spreadshet.*

itslearning has a built-in result extraction feature which exports the data to an Excel spreadsheet; this process is described in Section 2.2.2. We also created a custom result extraction application to export data from CMB to an Excel spreadsheet, covered in Section 4.4.2. Setup and usage of the application is described in Appendix A.

8.2 Summary and Conclusion

In this project, we have conduct research on how Climbing Mont Blanc (CMB), an online judge system, can work as an assessment tool in the TDT4102 course at the Norwegian University of Science and Technology (NTNU). The system is currently in development, so the purpose of our research has been to help the progression further. The current method of evaluating programming exams is both costly and with varying reliability, and one can never assume that human evaluators are completely consistent in their evaluation. A system for automatic assessment and grading would make the process of evaluating programming exams significantly more efficient and consistent.

Our first contribution towards a digital evaluation form is to describe different problem types used to evaluate programming problems in detail. Also, we reason about whether or not each of them is suitable to be used with CMB and online judge systems

in general. We have also reviewed question types suitable for learning management platforms that have a strict evaluation model and could easily be implemented in any exam tool, such as multiple choice and multiple response questions.

Subsequently, we provide a midterm assignment with multiple choice questions and programming problems well-suited to be evaluated by the system.

In-depth information about the usage of CMB as well as other online judge systems as a tool for digital examination is thoroughly described, which can be used to educate any new members to the project.

After the midterm had been conducted, we initiated extensive statistical analysis and discussion of the results. This information is further used to provide both valuable insight about the current status of the system, and suggestions for future improvements to the system as well as future experiments. In addition, a rough outline for the continuation of the project has been suggested which can give our predecessors an indication of where to initiate the next iteration of the project.

This thesis also describes possible improvements to the user interface. These suggestions will hopefully contribute to further engaging the students in using the system.

Our final contribution contains a method for extracting the results from CMB to an Excel spreadsheet. The extraction is in the form of a Python application, which converts JSON-data to the appropriate spreadsheet format.

In conclusion, our research has contributed to the development towards digital examination with an automatic grading of programming exams at NTNU. We have covered how to prepare for and execute experiments relevant to the project. Also, we describe how to create problems for both LMS systems and the CMB system so that they can be evaluated digitally with a focus on keeping the validity of the assessment at an acceptable level.

References

[Aik87] Lewis R Aiken. Testing with multiple-choice items. *Journal of Research & Development in Education*, 1987.

[Ama] Amazon. Amazon Web Services. `https://aws.amazon.com/`. Accessed: December 7th, 2015.

[Ast03] Owen Astrachan. Bubble sort: an archaeological algorithmic analysis. In *ACM SIGCSE Bulletin*, volume 35, pages 1–5. ACM, 2003.

[Ber15] Berit Kjeldstad. Digitalisering av eksamen angår oss alle. `https://www.ntnu.no/blogger/rektoratet/2015/08/digitalisering-av-eksamen-angar-oss-alle/`, 2015. Accessed: December 10th, 2015.

[Bio] Biomath. Sample size/power calulations. `http://www.biomath.info/power/ttest.htm`. Accessed: May 24th, 2016.

[BS87] James O Berger and Thomas Sellke. Testing a point null hypothesis: the irreconcilability of p values and evidence. *Journal of the American statistical Association*, 82(397):112–122, 1987.

[Bur04] Richard F Burton. Multiple choice and true/false tests: reliability measures and some implications of negative marking. *Assessment & Evaluation in Higher Education*, 29(5):585–595, 2004.

[CKLO03] Brenda Cheang, Andy Kurnia, Andrew Lim, and Wee-Chong Oon. On automated grading of programming assignments in an academic institution. *Computers & Education*, 41(2):121–131, 2003.

[CMB] CMB. Climbing Mont Blanc – How To. `https://climb.idi.ntnu.no/#/howto`. Accessed: April 28th, 2016.

[Coh92] Jacob Cohen. A power primer. *Psychological bulletin*, 112(1):155, 1992.

[CTL97] Christian Collberg, Clark Thomborson, and Douglas Low. A taxonomy of obfuscating transformations. Technical report, Department of Computer Science, The University of Auckland, New Zealand, 1997.

[DLT+95] Fritz Drasgow, Michael V Levine, Sherman Tsien, Bruce Williams, and Alan D Mead. Fitting polytomous item response theory models to multiple-choice tests. *Applied Psychological Measurement*, 19(2):143–166, 1995.

[DM95] François-Nicola Demers and Jacques Malenfant. Reflection in logic, functional and object-oriented programming: a short comparative study. In *Proceedings of the IJCAI*, volume 95, pages 29–38, 1995.

[EKN+11] Emma Enström, Gunnar Kreitz, Fredrik Niemelä, Pehr Söderman, and Viggo Kann. Five years with kattis—using an automated assessment system in teaching. In *Frontiers in Education Conference (FIE), 2011*, pages T3J–1. IEEE, 2011.

[Eph] Ephorus. Ephorus – Homepage. `http://www.ephorus.com/`. Accessed: December 5th, 2015.

[Fei] Feide. Introducing Feide. `https://www.feide.no/introducing-feide`. Accessed: December 8th, 2015.

[Fel] Felles Studentsystem. Felles Studentsystem – Homepage. `http://www.fellesstudentsystem.no/`. Accessed: December 7th, 2015.

[Fou] Python Software Foundation. pip x.x.x: Python package index. `https://pypi.python.org/pypi/pip`. Accessed: February 9th, 2016.

[Fri02] Jeffrey EF Friedl. *Mastering regular expressions*. "O'Reilly Media, Inc.", 2002.

[FS15] Torbjørn Follan and Simen Støa. Climbing Mont Blanc – A Prototype System for Online Energy Efficient Based Programming Competitions on ARM Platforms. Master's thesis, Norwegian University of Science and Technology (NTNU), 2015.

[GC] Eric Gazoni and Charlie Clark. openpyxl – A Python library to read/write Excel 2010 xlsx/xlsm files —- openpyxl 2.4.0 documentation. `https://openpyxl.readthedocs.org/en/2.4`. Accessed: February 9th, 2016.

[Goo] Google. Google Forms – create and analyze surveys, for free. `https://www.google.com/forms/about/`. Accessed: May 2nd, 2016.

[Har05] Wynne Harlen. Trusting teachers' judgement: Research evidence of the reliability and validity of teachers' assessment used for summative purposes. *Research Papers in Education*, 20(3):245–270, 2005.

[HD89] Thomas M Haladyna and Steven M Downing. A taxonomy of multiple-choice item-writing rules. *Applied measurement in education*, 2(1):37–50, 1989.

[HD97] James D Hansen and Lee Dexter. Quality multiple-choice test questions: Item-writing guidelines and an analysis of auditing testbanks. *Journal of Education for Business*, 73(2):94–97, 1997.

[HDR02] Thomas M Haladyna, Steven M Downing, and Michael C Rodriguez. A review of multiple-choice item-writing guidelines for classroom assessment. *Applied measurement in education*, 15(3):309–333, 2002.

[HO14] Larry V Hedges and Ingram Olkin. *Statistical methods for meta-analysis*. Academic press, 2014.

[Hol24] Karl J Holzinger. On scoring multiple response tests. *Journal of Educational Psychology*, 15(7):445, 1924.

[ICP] ICPC. The ACM-ICPC International Collegiate Programming Contest. `https://icpc.baylor.edu/`. Accessed: October 23rd, 2015.

[Inf] Information Technology Laboratory. Normal Distribution. `http://www.itl.nist.gov/div898/handbook/eda/section3/eda3661.htm`. Accessed: May 16th, 2016.

[its] itslearning. itslearning: How It All Began. `http://www.itslearning.net/our-story`. Accessed: May 3rd, 2016.

[Jef] Jeff Atwood. The Great Newline Schism. `http://blog.codinghorror.com/the-great-newline-schism/`. Accessed: May 3rd, 2016.

[JSO] JSON. json.org. `http://www.json.org/`. Accessed: May 31st, 2016.

[KLC01] Andy Kurnia, Andrew Lim, and Brenda Cheang. Online judge. *Computers & Education*, 36(4):299–315, 2001.

[Ltd14] Samsung Electronics Co. Ltd. Samsung Exynos Octa. `http://www.samsung.com/global/business/semiconductor/minisite/Exynos/w/solution.html`, 2014. Accessed: October 7th, 2015.

[Mag16] Sindre Magnussen. Improving System Usability of Climbing Mont Blanc – An Online Judge Focusing on Energy Efficient Programming. Master's thesis, Norwegian University of Science and Technology (NTNU), 2016. Note: Title may have changed.

[Mica] Microsoft. Load the analysis toolpak. `https://support.office.com/en-us/article/Load-the-Analysis-ToolPak-6a63e598-cd6d-42e3-9317-6b40ba1a66b4`. Accessed: May 21st, 2016.

[Micb] Microsoft. What's new in excel 2013. `https://support.office.com/en-IE/article/what-s-new-in-excel-2013-1cbc42cd-bfaf-43d7-9031-5688ef1392fd`. Accessed: May 21st, 2016.

[ML06] Morris L Marx and Richard J Larsen. *Introduction to mathematical statistics and its applications*, volume 31. Pearson/Prentice Hall Upper Saddle River, NJ, USA, 2006.

[Nat] Lasse Natvig. Climbing Mont Blanc project – NTNU. `https://www.ntnu.edu/idi/card/cmb`. Accessed: February 5th, 2016.

[NFS+15] Lasse Natvig, Torbjørn Follan, Simen Støa, Sindre Magnussen, and Antonio García-Guirado. Climbing mont blanc - A training site for energy efficient programming on heterogeneous multicore processors. *CoRR*, abs/1511.02240, 2015.

[Nie95] Jakob Nielsen. 10 usability heuristics for user interface design. *Fremont: Nielsen Norman Group.[Consult. 20 maio 2014]. Disponível na Internet*, 1995.

[Nora] Norsk Senter for Forskningsdata. Meldeplikt. `http://www.nsd.uib.no/personvern/meldeplikt/`. Accessed: May 13th, 2016.

[Norb] Norwegian University of Science and Technology (NTNU). IDI Open. `https://idiopen.idi.ntnu.no`. Accessed: October 23rd, 2015.

[Norc] Norwegian University of Science and Technology (NTNU). Nyhet – blackboard valgt som nytt e-læringssystem ved ntnu. `https://www.ntnu.no/aktuelt/2016/els`. Accessed: May 3rd, 2016.

[Nord] Norwegian University of Science and Technology (NTNU). Prosentvurderingsmetoden. `https://innsida.ntnu.no/wiki/-/wiki/Norsk/Prosentvurderingsmetoden`. Accessed: May 11th, 2016.

[oSNa] Norwegian University of Science and Technology (NTNU). Course – Procedural and Object-Oriented Programming – TDT4102 – NTNU. `https://www.ntnu.edu/studies/courses/TDT4102`. Accessed: November 3rd, 2015.

[oSNb] Norwegian University of Science and Technology (NTNU). Gradestatistics. `http://www.ntnu.no/karstat/makeReport.do`. Accessed: November 12th, 2015.

[oSNc] Norwegian University of Science and Technology (NTNU). Language in examination question papers. `https://innsida.ntnu.no/wiki/-/wiki/English/Language+in+examination+question+papers`. Accessed: June 2nd, 2016.

[oSNd] Norwegian University of Science and Technology (NTNU). NTNU – Digital eksamen. `https://www.ntnu.no/wiki/display/ppfntnuit/-Digital+eksamen`. Accessed: December 7th, 2015.

[oSNe] Norwegian University of Science and Technology (NTNU). Permitted Examination Aids – Wiki (Inside NTNU). `https://innsida.ntnu.no/wiki/-/wiki/English/Permitted+examination+aids`. Accessed: March 2016.

[PW08] De-chang PI and Qing-xian WU. Acm international collegiate programming contest and cultivation of innovative talent [j]. *Journal of Electrical & Electronic Education*, 3:019, 2008.

[Ros14] Sheldon M Ross. *Introduction to probability and statistics for engineers and scientists*. Academic Press, 2014.

[Sø15] Thea Marie Søgaard. Cheating threats in digital byod exams: A preliminary investigation. Master's thesis, Norwegian University of Science and Technology (NTNU), 2015.

[Saf] Safe Exam Browser. Safe Exam Browser – Overview. `http://safeexambrowser.org/about_overview_en.html`. Accessed: December 7th, 2015.

[SDM10] John Shalf, Sudip Dosanjh, and John Morrison. Exascale computing technology challenges. In *High Performance Computing for Computational Science–VECPAR 2010*, pages 1–25. Springer, 2010.

[SF12] Gail M Sullivan and Richard Feinn. Using effect size-or why the p value is not enough. *Journal of graduate medical education*, 4(3):279–282, 2012.

[Sol16] Solveig Mikkelsen. Vraker itslearning til fordel for amerikanske blackboard. *Universitetsavisa*, 2016. Available at `http://www.universitetsavisa.no/campus/2016/02/05/Vraker-Itslearning-til-fordel-for-amerikanske-Blackboard-55052.ece`. Accessed: May 3rd, 2016.

[Staa] Real Statistics. Real statistics resource pack. `http://www.real-statistics.com/free-download/real-statistics-resource-pack/`. Accessed: May 20th, 2016.

[Stab] Social Science Statistics. Mann-whitney u test calculator. `http://www.socscistatistics.com/tests/mannwhitney/Default2.aspx`. Accessed: May 24th, 2016.

[SV15] Guttorm Sindre and Aparna Vegendla. E-exams and exam process improvement. *Norsk Informatikkonferanse (NIK)*, 2015.

[SW65] Samuel Sanford Shapiro and Martin B Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611, 1965.

[Taf13] Daniele Tafani. The mont-blanc project. *Leibniz Supercomputing Centre*, 2013.

[Uni] The Hong Kong Polytechnic University. Effect size calculator. `http://www.polyu.edu.hk/mm/effectsizefaqs/calculator/calculator.html`. Accessed: May 24th, 2016.

[Wik] Wikipedia. Snake (video game) – Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Snake_(video_game)`. Accessed: April 20th, 2016.

[Win] Scott R. Winters. Score Normalization as a Fair Grading Practice. ERIC Digest. `http://www.ericdigests.org/2003-4/score-normilization.html`. Accessed: May 10th, 2016.

Appendix A
json2excel

This appendix documents the setup and usage of the Python-application we created in order to extract results from CMB to Excel, called `json2excel`, which is compatible with both Python 2.7 and Python 3.

A.1 Setup

Our application makes use of an external library called `openpyxl` [GC]. One must therefore install this library using `pip`, Python's package manager [Fou]. This can be done in the command line with the following command:

```
$ pip install openpyxl
```

A.2 Usage

Using the program simply requires a JSON-file with group data from CMB. Running the application from the command line can be done with the following command, where `data.json` should be replaced with the name of the JSON-file.

```
$ python json2excel.py data.json
```

This produces an `.xlsx`-file with the same name as the CMB group from which the data is obtained, located in the same directory as where the code was run.

A.3 JSON Object Structure

As its name implies, the `json2excel` application is based on JSON group data from CMB, structured as exemplified in Figure A.1.

```
group = {
  "description": "Group description",
  "id": 1,
  "members": [
    {
      "group_id": 1,
      "id": 1,
      "role": "leader",
      "user_id": 1,
      "username": "thea"
    }
  ],
  "name": "Test Group",
  "problems": [
    {
      "group_id": 1,
      "id": 1,
      "problem_id": 1,
      "problem_name": "The shortest path problem",
      "submissions": [
        {
          "edp": 5614.3759737,
          "energy": 124.68079,
          "goodness": null,
          "id": 1,
          "msg": null,
          "name": "shortPath.zip",
          "problem_id": 1,
          "state": "finished",
          "time": 45.03,
          "user_id": 1,
          "user_name": "thea",
          "visible": true
        }
      ]
    }
  ],
  "public": false
}
```


Figure A.1: Example of JSON group data from CMB.

It is very important that the data is structured this way; if not, one would have to modify the source code, which is presented below in Appendix A.4.

A.4 Source Code

```
import json
import sys
from openpyxl import Workbook

def read_json(filename):
    with open(filename, 'r') as data:
        return json.loads(data.read())

def find_best_submissions(submissions):
    best_submissions = {}
    for submission in submissions:
        username = submission['user_name']
        if (username in best_submissions):
            prev_submission = best_submissions[username]

            if not submission['time']:
                continue

            if not prev_submission['time']:
                best_submissions[username] = submission

            elif prev_submission['time'] > submission['time']:
                best_submissions[username] = submission
        else:
            best_submissions[username] = submission

    return best_submissions

def fill_cell(ws, col, row, val):
    ws.cell(column = col, row = row, value = val)

def write_excel_file(json_data):
    wb = Workbook()
    ws = wb.active

    user_rows = {}

    # Setup user column
    ws['A1'] = 'USER'
    row_counter = 2
```

```
41      for user in json_data['members']:
42          fill_cell(ws, 1, row_counter, user['username'])
43          user_rows[user['username']] = row_counter
44          row_counter += 1
45
46      # Setup problem and submission data
47      col = 1
48      for problem in json_data['problems']:
49          fill_cell(ws, col + 1, 1, problem['problem_name'])
50          ws.merge_cells(start_row = 1, start_column = col + 1,\
51          end_row = 1, end_column = col + 4)
52
53          best_submissions = find_best_submissions(problem['submissions'])
54
55          for key in best_submissions.keys():
56              row = user_rows[key]
57              values = [0, 0, 0]
58              if best_submissions[key]['time']:
59                  values = [best_submissions[key]['time'],\
60                  best_submissions[key]['energy'],\
61                  best_submissions[key]['edp']]
62
63              fill_cell(ws, col + 1, row, values[0])
64              fill_cell(ws, col + 2, row, values[1])
65              fill_cell(ws, col + 3, row, values[2])
66
67          col += 4
68
69      # Save Excel-file
70      wb.save('{0}.xlsx'.format(json_data['name']))
71
72  def main():
73      input_filename = sys.argv[1]
74
75      data = read_json(input_filename)
76      write_excel_file(data)
77
78  main()
```

Appendix B
Midterm

The following pages enclose the midterm questions used in the experiment, as well as the solution to each question.

Del 1 – Flervalg og kortsvar

1. (25 poeng) **Flervalgsoppgaver.**
   Runde svarbokser indikerer at nøyaktig ett av svaralternativene er riktig. Firkantede svarbokser indikerer at mer enn ett svaralternativ kan være riktig.

   (a) (5 poeng) Hvilke av påstandene er `true` etter at følgende kodesnutt har kjørt?

```
int x = 5;
int y = 42;
int* ptr1 = &x; // ptr1 peker til x
int* ptr2 = ptr1; // ptr2 er en kopi av ptr1; ptr2 = &x
*ptr2 = y; // *ptr2 = x og y = 42; x blir satt til 42
```

   ☐ `x == 5`
   ☒ `x == 42`
   ☐ `ptr2 == &y`
   ☒ `*ptr1 == y`

   (b) (5 poeng) Ifølge klassehierarkiet gitt i koden under, hvilke deklarasjoner er lovlige?

```
class A {};
class B: public A {};
class C: public B {};
class D: public A {};
```

   ☒ `A a = C();`
   ☐ `C c = A();`
   ☐ `D d = B();`
   ☒ `B b = C();`

(c) (5 poeng) En av måtene å sjekke om to C-string variabler er like, er å bruke operatoren `==`.

◯ Sant ⊗ **Usant**

Løsning: En C-string er i realiteten et array av `char`-instanser. Ved å skrive `cstring1 == cstring2` er det adressene til første element i hhv. `cstring1` og `cstring2` som sammenliknes, ikke selve innholdet.

(d) (5 poeng) Hvordan kan vi deklarere en funksjon som skal returnere et `int`-array av størrelse `n`?

◯ Det er ikke mulig. ⊗ `int* createArray(int n);`
◯ `int[] createArray(int n);` ◯ Begge er gyldige.

Løsning: Vi returnerer en peker til det første elementet i arrayet.

(e) (5 poeng) I hvilke tilfeller vil kopikonstruktøren til `MinKlasse` bli kalt? Anta at `minInstans` er av typen `MinKlasse`.

☒ `MinKlasse nyInstans = MinKlasse(minInstans);`
☒ `void minFunksjon(MinKlasse k);`
☐ `void minFunksjon(MinKlasse& k);`
☐ `MinKlasse nyInstans;`

2. (5 poeng) Hva blir output av følgende program?

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     int x = 7;
6     int y = 5;
7     int c = y++ + --x;
8     cout << c;
9 }
```

Løsning: `y` inkrementeres *etter* at den evalueres og `x` dekrementeres *før* den evalueres. $5 + 6 = 11$.

3. (5 poeng) Hvilken `char` brukes for å terminere en C-string?

> **Løsning:** `'\0'` (null-character)

Del 2 – Programmering

4. (25 poeng) **Overlasting av operatorer**

I denne oppgaven skal du implementere en klasse `MyInteger`. Denne klassen skal kun ha én medlemsvariabel, `value`, av typen `int`, samt følgende operatorer:

- addisjons-operatorene `+` og `+=`
- unær-operatoren `-` (brukes til å representere et negativt tall, f.eks. -10)
- insertion-operatoren `<<`

Lag en konstruktør for `MyInteger` som tar inn en tallverdi, og implementér operatorene. Du kan bruke header-filen (oppgitt under) som utgangspunkt, men merk at du selv må finne ut hvordan deklarasjonene til operatorene `+` og `+=` skal se ut.

```
1  // MyInteger.h
2  #include <iostream>
3  using namespace std;
4
5  class MyInteger {
6      int value;
7  public:
8      MyInteger(int val);
9      int getValue() const;
10     MyInteger operator-() const;
11     friend ostream& operator <<(ostream& cout, const MyInteger &myInt);
12 };
```

> **Løsning:** Se `MyInteger.cpp` og `MyInteger.h` i vedlagt .zip for løsning. Merk at den digitale versjonen ba om implementasjon av flere operatorer (`*= - /` etc.), så disse er også implementert i løsningen.

5. (40 poeng) **Implementér en klasse med tilhørende metoder og hjelpefunksjoner**

   (a) (15 poeng) Implementér en funksjon `split` som splitter opp en tekststreng basert på et gitt tegn. Funksjonen skal ta inn en `string` (strengen som skal splittes) og en `char` (tegnet som bestemmer hvor man skal splitte), og skal returnere en `vector<string>` bestående av hver enkelt delstreng. Bruk vedlegget om `vector`-klassen for hint om hvilke metoder du kan benytte deg av.

   Eksempel: Funksjonskallet `split("Ja_vi_elsker",'_')` skal returnere en `vector` med elementene `["Ja","vi","elsker"]`.
   Hint: `string`*-klassen har en innebygget metode kalt* `substr(startPos, length)`.

   Løsning: Se `main-split.cpp` i vedlagt .zip for løsning.

   (b) (25 poeng) **Roommates**

   1. Implementér klassen `Person`. En instans av **Person**-klassen skal ha et navn og en liste med romkamerater. Implementér metoder for å aksessere feltene. Du kan bruke følgende header-fil som utgangspunkt.

```
// Person.h
class Person {

    string name;
    vector<Person*> roomies;


public:
    Person(string name);
    void addRoomie(Person* roomie);
    vector<Person*> getRoomies() const;
    string getName() const;
};
```

   2. Implementér metoden `Person::addRoommate`. Denne metoden skal ta inn en instans av `Person` som skal legges til som romkamerat i listen over romkamerater. **Husk at dersom *PersonA* bor med *PersonB* og *PersonB* bor med *PersonC*, impliserer det at *PersonA* bor med *PersonC*.** Som et lite utgangspunkt har vi lagt inn en `for`-løkke som forhindrer en potensiell evig loop.

```
// Person.cpp
/* Metode for å legge til en roomie */
```

```
3  void Person::addRoomie(Person* roomie) {
4      /* LA STÅ: For å unngå evig loop */
5      for (int i = 0; i < roomies.size(); i++) {
6          if (roomies[i] == roomie || roomie == this) return;
7      }
8
9      // DIN KODE HER
10 }
```

3. Implementér funksjonen `findPersonByName` som tar inn et navn som en tekststreng og en vektor bestående av `Person`-instanser, og returnerer en peker til det passende `Person`-objektet fra vektoren.

Løsning: Se `Person.cpp` og `Person.h` i vedlagt .zip for løsning.

Vedlegg 1 – `vector`

En `vector<T>` er en liste som kan endre størrelse dynamisk.
Metoder som kan være nødvendige:

- `int size()`: Returnerer antall elementer i vectoren.
- `T at(int index)`: Returnerer elementet med indeks lik *index*
- `push_back(T element)`: Legger til objektet *element* til vektoren.

`T` er typen til objektet i vektoren.

Appendix C
Questionnaire

The following pages contains the questionnaire the participants received after the midterm test.

Spørreskjema midtsemesterprøve i C++

Til dette spørreskjemaet finnes det ingen rette eller gale svar, vi ønsker bare at du gir din ærlige mening til hvert spørsmål. Første del omhandler utførelsen av denne midtsemesterprøven og kan besvares av alle. Andre del omhandler Climbing Mont Blanc systemet og kan kun besvares av de som gjennomførte prøven digitalt.

*Må fylles ut

Utførelse av midtsemesterprøven

I hvilken grad...

1. **...føler du at du har fått vist dine kunnskaper i C++? ***
   Markér bare én oval.

	1	2	3	4	5	
Svært liten grad	◯	◯	◯	◯	◯	Svært stor grad

2. **...føler du at du har fått vist kjente metoder/teknikker i C++? ***
   Markér bare én oval.

	1	2	3	4	5	
Svært liten grad	◯	◯	◯	◯	◯	Svært stor grad

3. **...synes du at å løse oppgavene på den formen du fikk tildelt en effektiv måte å løse oppgaver? ***
   Markér bare én oval.

	1	2	3	4	5	
Svært liten grad	◯	◯	◯	◯	◯	Svært stor grad

4. **...synes du at den formen du fikk tildelt motiverte deg til å løse oppgavene? ***
   Markér bare én oval.

	1	2	3	4	5	
Svært liten grad	◯	◯	◯	◯	◯	Svært stor grad

5. **...føler du at den formen du fikk tildelt er en passende måte å løse programmeringsoppgaver? ***
   Markér bare én oval.

	1	2	3	4	5	
Svært liten grad	◯	◯	◯	◯	◯	Svært stor grad

6. **...føler du at den formen du fikk tildelt er en passende måte å bli evaluert på programmeringsoppgaver? ***
   Markér bare én oval.

	1	2	3	4	5	
Svært liten grad	◯	◯	◯	◯	◯	Svært stor grad

Hvor enig eller uenig er du i påstandene:

7. **Min tildelte prøveform (penn-og-papir eller digitalt) påvirket min prestasjon positivt. ***
   Markér bare én oval.

	1	2	3	4	5	
Helt uenig	◯	◯	◯	◯	◯	Helt enig

8. **Min tildelte prøveform (penn-og-papir eller digitalt) påvirket min prestasjon negativt. ***
   Markér bare én oval.

	1	2	3	4	5	
Helt uenig	◯	◯	◯	◯	◯	Helt enig

9. **Av de to mulige formene for å løse oppgavene (på papir og digitalt) hvilken form mener du er mest utsatt for juks? ***
   Markér bare én oval.

   - ◯ Itslearning/CMB
   - ◯ Penn-og-papir

10. **På hvilken form utførte du midtsemesterprøven? ***
    Markér bare én oval.

    - ◯ Itslearning/CMB
    - ◯ Penn-og-papir

Avslutt utfyllingen av dette skjemaet etter det siste spørsmålet.

11. **Hvor god tid følte du at du hadde på å fullføre prøven?**
Markér bare én oval.

	1	2	3	4	5	
Veldig dårlig tid	◯	◯	◯	◯	◯	Veldig god tid

Spørsmål angående itslearning/CMB

Hvor enig eller uenig er du i påstandene:

12. **Digital evaluering (itslearning/CMB) vil bidra positivt til videre læring i faget.** *
Markér bare én oval.

	1	2	3	4	5	
Helt uenig	◯	◯	◯	◯	◯	Helt enig

13. **Digital evaluering (itslearning/CMB) vil bidra til en forbedret eksamensprestasjon.** *
Markér bare én oval.

	1	2	3	4	5	
Helt uenig	◯	◯	◯	◯	◯	Helt enig

14. **Jeg ønsker å ta i bruk digitale evalueringsformer (itslearning/CMB) fremfor dagens evaluering av øvinger.** *
Markér bare én oval.

	1	2	3	4	5	
Helt uenig	◯	◯	◯	◯	◯	Helt enig

15. **Jeg ønsker å ta i bruk digitale evalueringsformer (itslearning/CMB) fremfor dagens evaluering av eksamen som skjer ved å levere inn håndskreven kode.** *
Markér bare én oval.

	1	2	3	4	5	
Helt uenig	◯	◯	◯	◯	◯	Helt enig

I hvilken grad...

16. **...føler du at en umiddelbar respons på innleveringer bidrar til å effektivisere oppgaveløsning som for eksempel en øving eller eksamen?**
Markér bare én oval.

	1	2	3	4	5	
Svært liten grad	◯	◯	◯	◯	◯	Svært stor grad

17. **...føler du at en umiddelbar respons på innleveringer bidrar til å motivere i oppgaveløsning som for eksempel en øving eller eksamen?**
Markér bare én oval.

	1	2	3	4	5	
Svært liten grad	◯	◯	◯	◯	◯	Svært stor grad

18. **...synes du at å levere oppgaver til CMB er mer effektivt enn å levere kode skrevet på PC til en sensor (som man gjør idag)?**
Markér bare én oval.

	1	2	3	4	5	
Svært liten grad	◯	◯	◯	◯	◯	Svært stor grad

19. **...synes du at å levere oppgaver til CMB er mer effektivt enn å levere kode skrevet for hånd på papir?**
Markér bare én oval.

	1	2	3	4	5	
Svært liten grad	◯	◯	◯	◯	◯	Svært stor grad