МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное образовательное учреждение

высшего профессионального образования

«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ М.В. ЛОМОНОСОВА»

Факультет вычислительной математики и кибернетики

УТВЕРЖДАЮ

Декан факультета ВМК

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Е.И. Моисеев

«\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2013

**Учебно-методический комплекс**

**«Конструирование компиляторов»**

Направление подготовки

*010400\_Прикладная математика и информатика*

Профили бакалавриата:

3 поток "Системное программирование и компьютерные науки"

Квалификация (степень) выпускника

Бакалавр

Форма обучения

Очная

|  |  |  |
| --- | --- | --- |
| Разработчик (составитель) УМК проф. Гайсарян С. С.  | \_\_\_\_\_\_\_\_\_\_\_\_  | «\_\_\_»\_\_\_\_\_\_\_\_\_20\_\_ г. |

Москва

2013

***Содержание***

1. Место дисциплины в структуре основной образовательной программы 3

2. Цели освоения дисциплины 3

3. Компетенции обучаемого, формируемые в результате освоения дисциплины 3

4. Рабочая программа учебной дисциплины 4

4.1 Тематический план 4

4.2 Структура дисциплины по видам работ 7

4.3. Комплексное индивидуальное практическое задание 9

4.4 Содержание лекций 10

5. Образовательные технологии 10

6. Учебно-методическое и информационное обеспечение дисциплины 11

7. Материально-техническое обеспечение дисциплины 11

8. Учебно-методическое обеспечение самостоятельной работы студентов 11

8.1 Пример учебно-методического обеспечения для практического задания 11

9. Контрольно-оценочные материалы 15

9.1 Программа экзамена по курсу 15

10. Рейтинг-план дисциплины 16

# *1. Место дисциплины в структуре основной образовательной программы*

Дисциплина «Конструирование компиляторов» входит в профессиональный блок вариативной части ОС МГУ по направлению подготовки 010400.62 «Прикладная математика и информатика». Логически и содержательно-методически данная дисциплина связана с общефакультетскими курсами: «Алгоритмы и алгоритмические языки», «Общая алгебра».

Освоение дисциплины «Конструирование компиляторов» обеспечивает студентов необходимыми знаниями и навыками в области оптимизирующей компиляции, а также в таких смежных областях, как статический анализ для выявления дефектов, обратная инженерия, генерация тестовых покрытий и др.. В рамках курса студенты обучаются применять изученный ранее математический аппарат для решения прикладных задач.

# *2. Цели освоения дисциплины*

Целью освоения дисциплины является получение базовых знаний в области разработки и применения современных компиляторов и компиляторных сред для разработки программ и для решения некоторых задач по обеспечению безопасного функционирования программ, для решения которых применяются компиляторные технологии.

# *3. Компетенции обучаемого, формируемые в результате освоения дисциплины*

В результате освоения дисциплины обучающийся должен:

*Знать*: структуру современного оптимизирующего компилятора; структуру и состав современных компиляторных сред: GCC, LLVM и др.; промежуточные представления программы, удобные для статической оптимизации; :методы машинно-независимой оптимизации; методы статической машинно-ориентированной оптимизации; методы динамического анализа и динамической оптимизации; методы динамического анализа программ и принципы построения динамических (JIT) компиляторов.

*Уметь:* применять компиляторные методы и технологии для решения задач обратной инженерии, защиты программного кода, обнаружения дефектов в программах и др.; строить квази-оптимальную SSA-форму процедуры; «укрупнять» узлы графа потока управления, выделяя в нем суперблоки и другие области и пересчитывая передаточные функции; осознанно выбирать режим работы компилятора, оптимальный для его программы.

*Владеть:* методами статического анализа и оптимизации программ на основе исследования потока данных; методами и средствами статической машинно-ориентированной оптимизации (распределение регистров, планирование кода, использование параллелизма уровня машинных инструкций).

*Универсальные, профессиональные и специализированные компетенции, которыми должен обладать студент в результате освоения дисциплины*

Универсальные компетенции:

а) *общенаучные*: способность анализировать научные проблемы и физические процессы, использовать на практике фундаментальные знания, полученные в области естественных и гуманитарных наук (НК-1); способность осваивать новую проблематику, терминологию, методологию, овладевать научными знаниями, владеть навыками самостоятельного обучения (НК-2); способность логически точно, аргументировано и ясно формулировать свою точку зрения, владеть навыками научной и общекультурной дискуссий (НК-3); готовность к творческому взаимодействию с коллегами по работе и научным коллективом, способность и умение выстраивать межличностное взаимодействие, соблюдая уважение к товарищам и проявляя терпимость к иным точкам зрения (НК-4).

в) *системные*:

способность к творчеству, порождению инновационных идей, выдвижению самостоятельных гипотез (СК-1);

способность к поиску, критическому анализу, обобщению и систематизации научной информации, к постановке целей исследования и выбору оптимальных путей и методов их достижения (СК-2);

Профессиональные компетенции:

в области научно-исследовательской деятельности:

способность понимать и применять в исследовательской и прикладной деятельности современный математический аппарат (ПК-2);

в проектной и производственно-технологической деятельности: способность применять в профессиональной деятельности современные языки программирования и методы параллельной обработки данных, операционные системы, электронные библиотеки и пакеты программ, сетевые технологии (ПК-3);

способность приобретать новые научные и профессиональные знания, используя современные образовательные и информационные технологии (ПК-9);

способность осуществлять целенаправленный поиск информации о технологических достижениях в сети Интернет и из других источников (ПК-10).

Общая трудоёмкость дисциплины составляет 3,0 зачётных единиц (108 часов). Лекции – 32 часа, семинары – 32 часа, самостоятельная работа – 44 часа,

Экзамен в 7-м семестре.

За курс отвечает кафедра системного программирования.

# *4. Рабочая программа учебной дисциплины*

## 4.1 Тематический план

|  |  |  |  |
| --- | --- | --- | --- |
| № | Название темы | Аудиторные занятия (часы) | Самостоятельная работа студента |
| лекции | семинары |
|  | Введение. Описание процесса компиляции. Структура оптимизирующего компилятора. Основные вопросы, изучаемые в курсе. | 1 |  | 2 |
|  | Построение промежуточного представления программы. Базовые блоки и граф потока управления. Биткод среды LLVM – пример промежуточного представления.  | 1 | 2 | 2 |
|  | Локальная оптимизация. Метод нумерации значений. Представление базового блока в виде направленного ациклического графа | 2 | 2 | 2 |
|  | Анализ потока данных – основной метод глобальной оптимизации. Примеры анализа потока данных – анализ достигающих определений и анализ живых переменных.  | 4 | 2 | 2 |
|  | Граф потока управления: остовное дерево, обход, нумерация вершин, классификация дуг, отношение до-минирования и построение дерева доминаторов.  | 4 | 4 | 4 |
|  | SSA-форма промежуточного пред-ставления и ее построение. Граница доминирования. Анализ потока данных в SSA-форме. Доступные выражения.  | 2 | 2 | 2 |
|  | Обоснование анализа потока данных: полурешетки, передаточные функции, общий итерационный алгоритм.  |  |  |  |
|  | Методы ускорения анализа потока данных. Суперблоки и другие области графа потока управления. Вычисление передаточных функций областей по передаточным функци-ям составляющих их базовых блоков. | 4 | 4 | 4 |
|  | Глобальный метод нумерации значений  | 4 | 4 | 4 |
|  | Глобальный анализ указателей. Псевдонимы (алиасы). Недостаточность глобального анализа. Межпроцедурный анализ. Граф вызовов. Методы учета контекста.  | 4 | 4 | 4 |
|  | Задачи решаемые на этапе машинно-ориентированной оптимизации. Планирование кода. Распределение регистров | 2 | 2 | 2 |
|  | Другие методы оптимизации: оптимизация потока управления, возвраты из рекурсивных функций. Раскрутка циклов. Открытая вставка функций.  | 2 | 2 | 2 |
|  | Генерация объектного кода методом переписывания дерева  | 4 | 4 | 4 |
|  | Комплексное практическое задание. | 0 | 0 | 12 |
|  | Итого: | 32 | 32 | 44 |
|  | Всего (часы): (аудиторные занятия и самостоятельная работа) | 108 |

## 4.2 Структура дисциплины по видам работ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | РазделДисциплины | Неделя семестра | Виды учебной работы, включая самостоятельную работу студентов и трудоемкость (в часах) | Формы текущего контроля успеваемости *(по неделям семестра)*Форма промежуточной аттестации *(по семестрам)* |
| лекц | прак | сам |
|  | Введение. Описание процесса компиляции. Структура оптимизирующего компилятора. Основные вопросы, изучаемые в курсе. | 1 | 1 |  | 1 |  |
|  | Построение промежуточного представления программы. Базовые блоки и граф потока управления. Биткод среды LLVM – пример промежуточного представления.  | 1 | 1 | 2 | 1 | Промежуточная проверка практического задания |
|  | Локальная оптимизация. Метод нумерации значений. Представление базового блока в виде направленного ациклического графа | 2 | 2 | 2 | 2 | Промежуточная проверка практического задания |
|  | Анализ потока данных – основной метод глобальной оптимизации. Примеры анализа потока данных – анализ достигающих определений и анализ живых переменных.  | 3 | 2 | 2 | 2 | Промежуточная проверка практического задания с выставлением технических баллов |
|  | Граф потока управления: остовное дерево, обход, нумерация вершин, классификация дуг, отношение до-минирования и построение дерева доминаторов.  | 4 | 4 | 4 | 4 | Промежуточная проверка практического задания |
|  | SSA-форма промежуточного пред-ставления и ее построение. Граница доминирования. Анализ потока данных в SSA-форме. Доступные выражения.  | 5-6 | 2 | 2 | 2 | Промежуточная проверка практического задания |
|  | Обоснование анализа потока данных: полурешетки, передаточные функции, общий итерационный алгоритм.  | 7 | 4 | 4 | 4 | Промежуточная проверка практического задания |
|  | Методы ускорения анализа потока данных. Суперблоки и другие области графа потока управления. Вычисление передаточных функций областей по передаточным функци-ям составляющих их базовых блоков. | 8-9 | 4 | 4 | 4 | Промежуточная проверка практического задания с выставлением технических баллов |
|  | Глобальный метод нумерации значений  | 10  | 2 | 2 | 2 | Промежуточная проверка практического задания |
|  | Глобальный анализ указателей. Псевдонимы (алиасы). Недостаточность глобального анализа. Межпроцедурный анализ. Граф вызовов. Методы учета контекста.  | 11-12 | 4 | 2 | 2 | Промежуточная проверка практического задания |
|  | Задачи решаемые на этапе машинно-ориентированной оптимизации. Планирование кода. Распределение регистров | 13-14 | 4 | 4 | 4 | Промежуточная проверка практического задания |
|  | Другие методы оптимизации: оптимизация потока управления, возвраты из рекурсивных функций. Раскрутка циклов. Открытая вставка функций.  | 15 | 2 | 2 | 2 | Промежуточная проверка практического задания |
|  | Генерация объектного кода методом переписывания дерева  | 16 | 2 | 2 | 2 | Итоговая проверка практического задания с выставлением технических баллов |
| 14 | Комплексное практическое задание. | 1-16 | 0 | 0 | 12 |  |

## 4.3. Комплексное индивидуальное практическое задание

В качестве практического задания студентам предлагается попробовать самостоятельно реализовать одно из оптимизирующих преобразований, используя инфраструктуру компилятора LLVM и его промежуточное представление – биткод.

Примеры оптимизирующих преобразований:

* Нахождение доступных выражений (для исключения избыточных вычислений)
* Планирование кода в суперблоке
* Распространение копий
* Вынос инварианьных вычислений в предзаголовок цикла.
* Распространение констант (с вычислением или без)

На первом этапе выполнения задания (первая - вторая недели) студент должен изучить среду LLVM, ее промежуточное представление (биткод), и, воспользовавшись компилятором Clang, получить биткод своего задания. Далее, пользуясь возможностями LLVM, он должен составить фазу (программу), выполняющую требуемое преобразование и включить ее в состав учебного компилятора на базе LLVM. Для проверки задания используется автоматическая система на удаленном сервере.

Срок выполнения задания – 12 недель. В конце семестра выставляются итоговые технические баллы.

## 4.4 Содержание лекций

Лекция 1. Введение. Описание процесса компиляции. Структура оптимизирующего компилятора. Основные вопросы, изучаемые в курсе. Построение промежуточного представления программы. Базовые блоки и граф потока управления. Биткод среды LLVM – пример промежуточного представления.

Лекция 2. Локальная оптимизация. Метод нумерации значений: представление базового блока в виде направленного ациклического графа.

Лекция 3. Анализ потока данных – основной метод глобальной оптимизации. Примеры анализа потока данных – анализ достигающих определений и анализ живых переменных. Вынесение инвариантных вычислений за пределы цикла.

Лекция 4. Граф потока управления: остовное дерево, обход, нумерация вершин, классификация дуг. Отношение доминирования и построение дерева доминаторов.Построение естественных циклов и гнезд циклов.

Лекция 5. SSA-форма промежуточного представления и ее построение. Граница доминирования.

Лекция 6. Анализ потока данных в SSA-форме. Выявление доступных выражений. Исключение избыточности.

Лекция 7. Обоснование анализа потока данных: полурешетки, передаточные функции, общий итерационный алгоритм.

Лекция 8. Методы ускорения анализа потока данных. Суперблоки и другие области графа потока управления.

Лекция 9. Вычисление передаточных функций областей по передаточным функциям составляющих их базовых блоков. Пример – анализ достигающих определений.

Лекция 10. Глобальный метод нумерации значений с использованием дерева доминаторов.

Лекция 11. Глобальный анализ указателей. Псевдонимы (алиасы). Недостаточность глобального анализа.

Лекция 12. Межпроцедурный анализ. Граф вызовов. Методы учета контекста..

Лекция 13. Задачи, решаемые на этапе машинно-ориентированной оптимизации. Планирование кода.

Лекция 14. Задачи решаемые на этапе машинно-ориентированной оптимизации. Распределение регистров.

Лекция 15. Другие методы оптимизации: оптимизация потока управления, возвраты из рекурсивных функций. Раскрутка циклов. Открытая вставка функций.

Лекция 16. Генерация объектного кода методом переписывания дерева

# 5. Образовательные технологии

Используются традиционные технологии проведения лекций. Учебные и методические материалы опубликованы на веб-странице http://algcourse.cs.msu.su/?page\_id=409.

Для консультирования, приёма заданий и информирования студентов используется электронная почта, блог и система автоматической проверки решений на упомянутой веб-странице.

# *6. Учебно-методическое и информационное обеспечение дисциплины*

а) основная литература

1. Keith D. Cooper, Linda Torczon. Engineering a Compiler, Second Edition. 2012 Elsevier, Inc..
2. Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. Compilers: Principles, Techniques, and Tools. 2nd Edition Addison-Wesley.2006 (last modified 2008) Есть русский перевод 2010.

б) дополнительная литература:

1. Steven S. Muchnick Advanced Compiler Design & Implementation. Morgan Kaufman Publishers, 1997
2. Y.N. Srikant, Priti Shankar. The Compiler Design Handbook, Second Edition, CRC Press, 2008.

в) программное обеспечение и Интернет-ресурсы

1. Веб-страница дисциплины: ://algcourse.cs.msu.su/?page\_id=409./
2. Веб-сайт LLVM http://llvm.org

# *7. Материально-техническое обеспечение дисциплины*

Используются компьютер и мультимедийный проектор для демонстрации слайдов в ходе лекций. Для самостоятельной работы студенты используют компьютеры в машинных залах и/или личные компьютеры.

# *8. Учебно-методическое обеспечение самостоятельной работы студентов*

## 8.1 Пример учебно-методического обеспечения для практического задания

### Постановка задачи

Целью работы является создание оптимизирующего преобразования, позволяющего повысить качество кода, генерируемого учебным компилятором на базе LLVM. Задача оптимизации кода состоит в повышении его быстродействия и/или сокращения его размера.

Предлагается разработать преобразование, которое на вход получает программу в промежуточном представлении LLVM, а на выходе генерирует ее версию, оптимизированную по размеру и скорости выполнения путем удаления недостижимого и бесполезного кода. Например:

## Первоначальная программа:

**int main (void) {**

**printf(“First message \n”);**

**int a = 34+48;**

**if (0)**

 **printf(“Unreachable code\n”);**

**a = a \* 2;**

**printf(“Second message\n”);**

**return 0;**

**}**

## Программа после удаления мертвого и недостижимого кода:

**int main (void) {**

**printf(“First message \n”);**

**printf(“Second message\n”);**

**return 0;**

**}**

### Решение задачи

## Теоретические аспекты

Предполагается реализация алгоритмов удаления мертвого и недостижимого кода. *Недостижимым кодом*называют часть кода программы, которая ни при каких условиях не может быть исполнена, поскольку является недостижимой в графе потока управления.
*Мертвый (неиспользуемый, бесполезный) код* - команды, вычисляющие никогда не использующиеся значения.

## Практические аспекты

Решения должны быть написаны на языке С++ с использованием контейнеров и алгоритмов стандартной библиотеки, а также средств, предоставляемых LLVM. Оптимизирующие преобразования должны быть выполнены в учебном компиляторе, который представляет собой модифицированную версию LLVM, не содержащую оптимизирующих преобразований.

Пример сборки LLVM в ОС Linux:

В каталоге с исходными кодами следует создать каталог с названием **build** и перейти в него:

**$ cd llvm/**

**$ mkdir build**

**$ cd build**

Затем следует осуществить сборку LLVM с указанием каталога сборки и желаемого типа сборки (Debug/Release). Сборка осуществляется с помощью утилит **configure** и **make**. Ключи **configure**:

--prefix=<путь к каталогу инсталляции>

--disable-assertions/--enable-assertion – включить/выключить проверку утверждений

--enable-optimized/--disable-optimized – выбрать тип сборки: оптимизированная (Release) или без оптимизаций (Debug)

Для ускорения компиляции рекомендуется использовать ключ “**-jN**” утилиты **make**, где N -- число желаемых потоков сборки (как правило, равное количеству ядер процессора или количеству ядер процессора + 1).

Пример сборки Debug с выключенной проверкой утверждений и установкой в каталог </home/user/llvm> и сборкой в 8 потоков:

**$ ../configure –prefix=/home/user/llvm –disable-optimized – disable-assertions**

**$ make –j8**

**$ make install**

После завершения установки и компиляции требуется добавить путь до каталога с LLVM в переменную окружения PATH:

**$ export PATH=/home/user/llvm/bin:$PATH**

Получить промежуточное представление LLVM для программы можно, выполнив команду:

**$ clang –c –O0 –emit-llvm test.c –o test.bc**

Перевод бинарного представления в ассемблер LLVM

**$ llvm-dis test.bc –o test.ll**

Ассемблирование в бинарное представление

**$ llvm-as test.ll –o newTest.bc**

Запуск преобразования pass\_name из динамической библиотеки **pass\_name.so**

**$ opt -load <путь/до/динамической/библиотеки/>pass\_name.so -<pass\_name> test.bc –o transformedTest.bc**

test.bc - файл, содержащий бинарную версию промежуточного представления LLVM.

test.ll – файл, содержащий ассемблер LLVM в читаемом виде.

TH

**Написание оптимизирующего прохода LLVM**

Для выполнения работы могут потребоваться проходы следующих типов: FunctionPass (по функциям), ModulePass (по модулям), BasicBlockPass (по базовым блокам).

В качестве примера приведен проход по функциям и именем Hello, вызываемым из командной строки с помощью ключа “–hello”:

**#include "llvm/Pass.h"**

**#include "llvm/IR/Function.h"**

**#include "llvm/Support/raw\_ostream.h"**

**using namespace llvm;**

**namespace {**

 **struct Hello : public FunctionPass {**

 **static char ID;**

 **Hello() : FunctionPass(ID) {}**

 **virtual bool runOnFunction(Function &F) {**

 **errs() << "Hello: ";**

 **errs().write\_escaped(F.getName()) << '\n';**

 **return false;**

 **}**

 **};**

**}**

**char Hello::ID = 0;**

**static RegisterPass<Hello> X("hello", "Hello World Pass", false, false);**

**Листинг 1. Пример компиляторного прохода**

Данный код выводит на экран сообщения вида “Hello: <function\_name>”.

Более детально изучить построение оптимизирующих проходов можно посмотреть в документации на компилятор (страница WritingAnLLVMPass.html в каталоге docs в поставке учебного компилятора).

### Тестирование

На личной странице расположена форма загрузки файла, а также информация о результатах тестирования и минимальный набор синтетических тестов.

**Загрузка решения.** Загружаемый файл должен представлять собой текст программы на языке С++, содержащий компиляторный проход, запускаемый по ключу “-**dce**”. После загрузки решение будет скомпилировано и запущенно с помощью утилиты “**opt**” на тестовом наборе данных.

Оценивается последнее присланное решение.

Для тестирования на локальной машине во время разработки предлагается осуществлять компиляцию программы с уровнем оптимизации “**O0**”. Локальное тестирование предлагается осуществлять с помощью программ с открытыми исходными кодами и минимального набора синтетических тестов.

## Оценка

Обязательное условие:

Присланное решение должно проходить проверку на корректность на программах **SQlite** и **Lzma.**

Оценка эффективности будет производиться на расширенном синтетическом наборе тестов.

«**Удовлетворительно**» – реализовано удаление бесполезного кода на уровне функций.

 «**Хорошо**» - реализовано удаление бесполезного и недостижимого кода на уровне функций. Удаление избыточных вызовов “pure” функций (не имеющих побочных эффектов и зависящих только от аргумента), например в выражении **double y=sin(x),** если y нигде более не используется вызов **sin(x)** можно удалить.

«**Отлично**» - реализовано удаление бесполезного и недостижимого кода на уровне функций. Реализован алгоритм удаления частично избыточного кода, алгоритм описан в публикации: Jens Knoop, Oliver Rüthing, and Bernhard Steffen. 1994. Partial dead code elimination.*SIGPLAN Not.* 29, 6 (June 1994), 147-158. DOI=10.1145/773473.178256 http://doi.acm.org/10.1145/773473.17825.

# *9. Контрольно-оценочные материалы*

## 9.1 Программа экзамена по курсу

1. Структура оптимизирующего компилятора.. Построение промежуточного представления программы.

2. Базовые блоки и граф потока управления. Биткод среды LLVM – пример промежуточного представления.

3. Локальная оптимизация. Метод нумерации значений: представление базового блока в виде направленного ациклического графа.

4. Анализ потока данных – основной метод глобальной оптимизации. Примеры анализа потока данных – анализ достигающих определений

5. Анализ живых переменных. Исключение мерьвого кода.

6. . Вынесение инвариантных вычислений за пределы цикла.

7. Граф потока управления: остовное дерево, обход, нумерация вершин, классификация дуг.

8. Отношение доминирования и построение дерева доминаторов

9. .Построение естественных циклов и гнезд циклов.

10. SSA-форма промежуточного представления и ее построение. Граница доминирования.

11. Анализ потока данных в SSA-форме. Выявление доступных выражений. Исключение избыточности.

12. Обоснование анализа потока данных: полурешетки, передаточные функции, общий итерационный алгоритм.

13. Методы ускорения анализа потока данных. Суперблоки и другие области графа потока управления.

14. Вычисление передаточных функций областей по передаточным функциям составляющих их базовых блоков. Пример – анализ достигающих определений.

15. Вычисление передаточных функций областей по передаточным функциям составляющих их базовых блоков на примере анализа достигающих определений.

16. Глобальный метод нумерации значений – использование дерева доминаторов.

17. Глобальный анализ указателей. Псевдонимы (алиасы). Недостаточность глобального анализа.

18. Межпроцедурный анализ. Использование графа вызовов.

19. Межпроцедурный анализ. Методы учета контекста вызова.

20. Задачи, решаемые на этапе машинно-ориентированной оптимизации.

21. Планирование кода.

22. Распределение регистров.

23. Оптимизация потока управления, возвраты из рекурсивных функций.

24. Раскрутка циклов.

25. Открытая вставка функций.

26. Генерация объектного кода методом переписывания дерева

# *10. Рейтинг-план дисциплины*

Оценка по курсу устанавливается в зависимости от суммы технических баллов, набранных студентом в ходе семестра. За решение практического задания можно заработать до 60 технических баллов, за экзамен до 40 технических баллов. Технические баллы за практическое задание выставляются в зависимости от качества решения задачи (метрика качества зависит от задания) и времени присланного решения. Таким образом, максимально возможная сумма набранных технических баллов составляет 100. Оценка «отлично» ставится студентам, набравшим от 80 баллов и выше. Оценка «хорошо» ставится студентам, набравшим от 60 до 79 технических баллов. Оценка «удовлетворительно» ставится студентам, набравшим от 40 до 59 технических баллов. Оценка «неудовлетворительно» ставится студентам, набравшим менее 40 технических баллов.