ЮЖНО-УРАЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

УТВЕРЖДАЮ
Директор института
Высшая школа электроники и
компьютерных наук

_____Г. И. Радченко
11.10.2017

РАБОЧАЯ ПРОГРАММА
практики
к ОП ВО от 28.06.2017 №007-03-0354

Практика Учебная практика
для направления 09.03.01 Информатика и вычислительная техника
Уровень бакалавр **Тип программы** Академический бакалавриат
профиль подготовки Вычислительные машины, комплексы, системы и сети
форма обучения очная
кафедра-разработчик Электронные вычислительные машины

Рабочая программа составлена в соответствии с ФГОС ВО по направлению подготовки 09.03.01 Информатика и вычислительная техника, утверждённым приказом Минобрнауки от 12.01.2016 № 5

Зав.кафедрой разработчика,
к.физ-мат.н., доц.
(ученая степень, ученое звание) _____10.10.2017____ (подпись) Г. И. Радченко

Разработчик программы,
старший преподаватель
(ученая степень, ученое звание, должность) _____10.10.2017____ (подпись) А. А. Кирсанова

Челябинск

1. Общая характеристика

Вид практики

Учебная

Способ проведения

Стационарная или выездная

Тип практики

практика по получению первичных профессиональных умений и навыков, в том числе первичных умений и навыков научно-исследовательской деятельности

Форма проведения

Дискретная

Цель практики

Получить первичные навыки в обобщенном программировании и алгоритмизации сортировок

Задачи практики

Изучить методы и технологии обобщенного программирования, шаблонные функции, шаблонные классы, алгоритмы сортировок

Краткое содержание практики

Обобщенное программирование, понятие шаблона, шаблон функции, шаблон класса, алгоритмы сортировок

2. Компетенции обучающегося, формируемые в результате прохождения практики

Планируемые результаты освоения ОП ВО (компетенции)	Планируемые результаты обучения при прохождении практики (ЗУНы)
ОПК-2 способностью осваивать методики использования программных средств для решения практических задач	Знать:Методики обобщенного программирования
	Уметь:выбирать методику, соответствующую поставленной задаче
	Владеть:навыками освоения методик
ПК-1 способностью разрабатывать модели компонентов информационных систем, включая модели баз данных и модели интерфейсов "человек - электронно-вычислительная машина"	Знать:приемы и методы обобщенного программирования
	Уметь:Программировать элементы информационных систем, требующие обобщенного программирования
	Владеть:методикой сопоставления

	компонентов системы и средств их программной реализации
ПК-4 способностью готовить конспекты и проводить занятия по обучению работников применению программно-методических комплексов, используемых на предприятии	Знать:методологию подготовки конспектов
	Уметь:Анализировать информацию из открытых источников и систематизировать найденную информацию в виде конспекта
	Владеть:навыками отбора источников информации

3. Место практики в структуре ОП ВО

Перечень предшествующих дисциплин, видов работ	Перечень последующих дисциплин, видов работ
Б.1.15.03 Объектно-ориентированное программирование	В.1.19 Программная инженерия (методологии проектирования программного обеспечения, паттерны) ДВ.1.06.01 Программирование на языках объектно-ссылочной модели

Требования к «входным» знаниям, умениям, навыкам студента, необходимым для прохождения данной практики и приобретенным в результате освоения предшествующих дисциплин:

Дисциплина	Требования
Б.1.15.03 Объектно-ориентированное программирование	Знать основные принципы объектно-ориентированного программирования

4. Время проведения практики

Время проведения практики (номер уч. недели в соответствии с графиком) с 44 по 47

5. Структура практики

Общая трудоемкость практики составляет зачетных единиц 6, часов 216, недель 4.

№ раздела (этапа)	Наименование разделов (этапов) практики	Кол-во часов	Форма текущего контроля
1	Алгоритмы сортировки	50	Проверка созданной программы
2	Обобщенное программирование	166	Проверка созданной программы

6. Содержание практики

№ раздела	Наименование или краткое содержание вида работ на	Кол-во

(этапа)	практике	часов
1.1	Поиск информации по различных алгоритмам сортировки	25
2.2	Поиск информации об обобщенном программировании	46
1.1	Реализация различных алгоритмов сортировки	25
2.2	Реализация варианта задания с использованием обобщенного программирования	120

7. Формы отчетности по практике

По окончанию практики, студент предоставляет на кафедру пакет документов, который включает в себя:
- дневник прохождения практики, включая индивидуальное задание и характеристику работы практиканта организацией;
- отчет о прохождении практики.
Формы документов утверждены распоряжением заведующего кафедрой от 23.09.2016 №308-10-15.

8. Фонд оценочных средств для проведения промежуточной аттестации обучающихся по практике

Форма итогового контроля – оценка.

8.1. Паспорт фонда оценочных средств

Наименование разделов практики	Код контролируемой компетенции (или ее части)	Вид контроля
Все разделы	ПК-4 способностью готовить конспекты и проводить занятия по обучению работников применению программно-методических комплексов, используемых на предприятии	Проверка созданной программы
Все разделы	ПК-1 способностью разрабатывать модели компонентов информационных систем, включая модели баз данных и модели интерфейсов "человек - электронно-вычислительная машина"	Проверка созданной программы
Все разделы	ОПК-2 способностью осваивать методики использования программных средств для решения практических задач	Проверка созданной программы
Все разделы	ОПК-2 способностью осваивать методики использования программных средств для решения практических задач	Диф. зачет
Все разделы	ПК-1 способностью разрабатывать модели компонентов информационных систем, включая модели баз данных и модели интерфейсов "человек - электронно-вычислительная машина"	Диф. зачет
Все разделы	ПК-4 способностью готовить конспекты и проводить занятия по обучению работников	Диф. зачет

	применению программно-методических комплексов, используемых на предприятии	

8.2. Виды контроля, процедуры проведения, критерии оценивания

Вид контроля	Процедуры проведения и оценивания	Критерии оценивания
Проверка созданной программы	Проверка созданных программ по алгоритмам сортировки и обобщенному программированию. За каждую программу выставляется балл по 5-балльной оценке. За оба задания максимум 10 баллов	Отлично: 9-10 Хорошо: 7-8 Удовлетворительно: 5-6 Неудовлетворительно: 0-4
Диф. зачет	Оценка за диф. зачет выставляется по результатам проверки созданной программы	Отлично: Программа оценена на 9-10 баллов Хорошо: Программа оценена на 7-8 баллов Удовлетворительно: Программа оценена на 5-6 баллов Неудовлетворительно: Программа оценена на 0-4 баллов

8.3. Примерный перечень индивидуальных заданий

Задания для практического задания:
Задание А1 (Сортировка + анализ сложности по времени)
Сортировка Шелла.
1) Рассмотреть различные известные подходы (минимум три различных подхода) к выбору значений длин промежутков при сортировке Шелла, на которых будут находиться сортируемые элементы исходного массива ёмкостью N на каждом шаге алгоритма.
2) Написать программы, в которых реализуется сортировка Шелла с различными подходами к выбору значений длин промежутков.
3) Сформировать несколько наборов данных (целых чисел) различного объема. Построить экспериментальный график зависимости времени выполнения кода (с различными подходами к выбору длин промежутков) от объема входных данных.
4) Сопоставить результаты с теоретическими оценками сложности алгоритмов (по времени исполнения). Сделать выводы о факторах, которых могли повлиять на точность измерений.
5) Сделать выводы об эффективности каждого из подходов к выбору значений длин промежутков.

Задание В1 (Стек, управляемый программой)
Разработать класс (или классы), реализующий стек. Способ реализации стека на основе расширяемого массива. Выполнить требования инкапсуляции.
Разработать отдельный модуль, который будет управлять стеком с помощью

программы, считанной из файла.
В программе могут содержаться следующие команды:
• PUSH // поместить число в стек
• POP // извлечь число из стека и вывести на экран
• PRINT // вывести содержимое стека на экран
• RESET // очистить содержимое стека
При выполнении недопустимой команды выводить соответствующее сообщение об ошибке и переходить к следующей команде.
Входной текстовый файл содержит программу для управления стеком. Результаты выполнения программы вывести на консоль.

Задание A2 (Рекурсия и нерекурсия + анализ сложности по времени и памяти)
1) Написать две программы вычисления ряда чисел Фибоначчи, одна из которых реализует рекурсивный алгоритм, а вторая – нерекурсивный. Ряд чисел задается соотношением: Ф[1]=Ф[2]=1; Ф[n]=Ф[n-1]+Ф[n-2].
2) Построить экспериментальный график зависимости времени выполнения кода двух программ и размера потребляемой памяти от длины вычисляемой последовательности чисел.
3) Сопоставить результаты с теоретическими оценками сложности алгоритмов (по времени исполнения) и оценками по потреблению памяти. Сделать выводы о факторах, которых могли повлиять на точность измерений.
4) Сделать выводы об эффективности рекурсивного и нерекурсивного алгоритма по использованию памяти и по времени выполнения.

Задание B2 (Алгоритм сортировочной станции)
Инфиксная и постфиксная запись (обратная польская нотация). Обеспечить перевод инфиксного выражения в ОПЗ и вычислить его результат.
В инфиксную запись входят операнды, обозначаемы латинскими буквами, а также операции:
1. Арифметические операции (+, -, *, /);
2. Круглые скобки;
3. ^ (возведение в степени);
4. тригонометрические функции (sin, cos, tan, cot, asin, acos, atan, acot);
5. показательные функции (exp, ln, log);
6. степенные функции (sqrt - квадратный корень, sqrt3 - кубический корень);
7. Операции с массивами (обращение к элементу массива - это бинарная операция)
Программа должна считывать исходное выражение из файла expression.txt и выводить его вместе с результатом преобразования на экран.
В файле expression.txt помимо инфиксной записи в строках, начиная со второй, расположены записи вида
переменная = значение
из которых необходимо получить значения переменных для подстановки в выражение.
Для реализации написать собственный класс Stack.

Задание A3 (Рекурсия и нерекурсия + сортировка + анализ сложности по времени и памяти)
1) Написать две программы, реализующие алгоритм быстрой сортировки. Одна из программ должна быть реализацией рекурсивного алгоритм, вторая реализует нерекурсивный алгоритм с использованием стека отложенных заданий.
2) Оценить размер стека, необходимый для нерекурсивной версии алгоритма. Как обойтись стеком, глубина которого ограничена C log n, где n – число сортируемых элементов?
3) Сформировать несколько наборов данных (целых чисел) различного объема. Построить экспериментальный график зависимости времени выполнения кода двух программ от объема входных данных.
4) Сопоставить результаты с теоретическими оценками сложности алгоритмов (по времени исполнения). Сделать выводы о факторах, которых могли повлиять на точность измерений.
5) Сделать выводы об эффективности рекурсивного и нерекурсивного алгоритма по времени выполнения.

Задание B3 (Инфиксная форма записи выражения)
В файле задано выражение в инфиксной форме записи (в обычной скобочной форме). В выражении могут быть целые положительные числа, знаки операций (* / + –), круглые скобки. Написать программу, сохраняющую считанное выражение в двоичном дереве (в котором знаки операций будут родительскими вершинами, а числа - листьями), вычисляющую результат выражения. Дерево представить в виде отдельного класса (или классов). Результат выражения вывести в выходной файл.
Приоритеты операций в выражении:
• * / высокий
• + – низкий

Задание A4 (Поиск + анализ сложности по времени выполнения)
1) Написать программы, реализующие следующие методы поиска:
a. последовательный поиск;
b. бинарный поиск;
c. интерполяционный поиск.
2) Сформировать несколько наборов данных (целых чисел) различного объема. Исходный отсортированный по возрастанию массив целых чисел может генерироваться со случайным шагом. Построить экспериментальный график зависимости времени выполнения кода программ от объема входных данных.
3) Сопоставить результаты с теоретическими оценками сложности алгоритмов (по времени исполнения). Сделать выводы о факторах, которых могли повлиять на точность измерений.
4) Сделать выводы об эффективности методов поиска по времени выполнения.

Задание B4 (Рекурсия и нерекурсия)
Написать рекурсивную и нерекурсивную версию программы для нахождения

последовательности перемещений колец в задаче о ханойских башнях. При реализации нерекурсивного алгоритма использовать стек отложенных заданий, элементами которого будут тройки (i,m,n). Каждая такая тройка интерпретируется как заказ «переложить i верхних дисков с m-го стержня на n-ый».

Задание A5 (Последовательность + анализ сложности на время исполнения)
Реализовать динамическую структуру данных - последовательность в двух вариантах: на основе расширяемого массива (вектора) и на основе двусвязного списка.
В последовательности хранятся целые числа.
В обоих случаях структура должна поддерживать один набор операций:
1) Добавление элемента в начало, в конец или в середину списка (в последнем случае - с указанием индекса)
2) Удаление элемента по индексу
3) Чтение элемента по индексу и получение первого или последнего элемента списка
4) Перестановка двух элементов местами (по индексам)
5) Замена элемента (по индексу)
6) Определение является ли список пустым
7) Подсчет длины списка (число элементов)
8) Полная очистка списка
Предложить различные стратегии динамического увеличения длины списка при реализации его на основе расширяемого массива и выбрать один из вариантов.
2. Провести сравнительный анализ этих двух реализаций последовательности, оценив время выполнения каждой операции. При испытаниях выполнить тесты на различных длинах последовательностей. Результаты оформить в табличном виде и в виде графиков.
3. Сопоставить полученные результаты измерений с теоретической оценкой сложности алгоритмов (по времени исполнения) каждой операции над последовательностью. Результаты оформить в табличном виде. Сделать выводы о факторах, которых могли повлиять на точность измерений. По результатам тестов и теоретической оценки алгоритмов сделать выводы о применимости и эффективности использования вектора и двусвязного списка.

Задание B5 (Ассоциативный массив)
Разработайте класс, который бы позволял хранить данные целого типа в массиве, индексами которого являются строки. Таким образом, должен работать следующий код:

```
AssocArray var(20); //массив из 20 элементов
var["First"]= 5;
printf("%d",var["First"]);//должно напечатать 5
```

Задание A6 (Алгоритм сортировки вставками и быстрой сортировки)
1) Написать две программы, реализующие алгоритм сортировки вставками и алгоритм быстрой сортировки (рекурсивный) для массива целых чисел.

2) Экспериментально сравнить реализации этих двух алгоритмов по времени исполнения на различных наборах данных, различных по объему и различных по упорядоченности (обратить внимание на крайние случаи – уже отсортированный массив и массив, отсортированный в обратном порядке). Оформить результаты экспериментов в табличном виде и построить графики зависимости времени выполнения программ от объема входных данных и упорядоченности входных последовательностей. Проверить экспериментально алгоритмы на устойчивость.
3) Сопоставить результаты с теоретическими оценками сложности алгоритмов (по времени исполнения). Сделать выводы о факторах, которых могли повлиять на точность измерений.
4) Сделать выводы, в каких случаях какой из алгоритмов является наиболее эффективным.

Задание B6 (Нерекурсивный алгоритм + обход дерева)
Написать нерекурсивную программу, печатающую все вершины двоичного дерева. При реализации использовать стек отложенных заданий. (А. Шень Программирование: теоремы и задачи. Задача 8.2.4).
Узлы дерева – символы латинского алфавита. Дерево задается в файле в формате: m [e [c [a], g [k]], s [p [o,s], y]]

Задание A7 (Пирамидальная сортировка)
1) Реализовать алгоритм пирамидальной сортировки (англ. Heapsort, «Сортировка кучей») — алгоритм сортировки, работающий в худшем, в среднем и в лучшем случае (то есть гарантированно) за $\Theta(n \log n)$ операций при сортировке n элементов. Количество применяемой служебной памяти не зависит от размера массива (то есть, $O(1)$).
2) Сформировать несколько наборов данных (целых чисел) различного объема. Построить экспериментальный график зависимости времени выполнения кода программы от объема входных данных.
3) Сопоставить результаты с теоретическими оценками сложности алгоритмов (по времени исполнения). Сделать выводы о факторах, которых могли повлиять на точность измерений.
4) Обосновать, что в реализации алгоритма количество применяемой служебной памяти действительно не зависит от размера сортируемого массива.
5) Сделать выводы.

Задание B7 (очередь с приоритетами)
Приоритетная очередь это очередь, в которой пажио не то, кто встал последним (порядок помещения в неё не играет роли), а кто главнее. Более точно, при помещении в очередь указывается приоритет помещаемого объекта (будем считать приоритеты целыми числами), а при взятии из очереди выбирается элемент с наибольшим приоритетом (или один из таких элементов). Реализовать приоритетную очередь так, чтобы помещение и взятие элемента требовали логарифмического числа действий (от размера очереди).

Задание A8 (простые алгоритмы сортировки + анализ сложности по времени)
1) Написать три программы, реализующие простейшие алгоритмы сортировки:

алгоритм сортировки пузырьком, алгоритм сортировки вставками и алгоритм сортировки выбором.
2) Экспериментально сравнить реализации этих алгоритмов по времени исполнения на различных наборах данных, различных по объему и различных по упорядоченности (обратить внимание на крайние случаи – уже отсортированный массив и массив, отсортированный в обратном порядке). Оформить результаты экспериментов в табличном виде и построить графики зависимости времени выполнения программ от объема входных данных и упорядоченности входных последовательностей. Проверить экспериментально алгоритмы на устойчивость.
3) Сопоставить результаты с теоретическими оценками сложности алгоритмов (по времени исполнения). Сделать выводы о факторах, которых могли повлиять на точность измерений.
4) Сделать выводы, в каких случаях какой из алгоритмов является наиболее эффективным.
Задание B8 (двоичное дерево)
Напишите работающую за линейное время рекурсивную процедуру, которая печатает ключи всех вершин двоичного дерева. Двоичное дерево задается в файле в следующем виде:
index/ key/ left/ right
1 12 7 3
2 15 8 NULL
3 4 10 NULL
4 10 5 9
5 2 NULL NULL
б 18 1 4
7 7 NULL NULL
8 11 6 2
9 21 NULL NULL
10 5 NULL NULL
ROOT 6 1 NULL

В конце файла указывается индекс вершины – корня дерева.
В столбце key –указывается значение, которое хранится в вершине дерева (целое число)
В столбцах left и right указываются индексы дочерних вершин для данной вершины (Если в обоих столбцах – NULL, то вершина является листом)

Задание A9 (сортировка разделением + включением + слиянием)
1) Написать три программы, реализующие эти алгоритмы.
2) Экспериментально сравнить реализации этих алгоритмов по времени исполнения на различных наборах данных, различных по объему и различных по упорядоченности (обратить внимание на крайние случаи). Оформить результаты экспериментов в табличном виде и построить графики зависимости времени выполнения программ от объема входных данных и упорядоченности входных последовательностей. Проверить экспериментально алгоритмы на устойчивость.
3) Сопоставить результаты с теоретическими оценками сложности алгоритмов (по времени исполнения). Сделать выводы о факторах, которых могли повлиять на точность измерений.

4) Сделать выводы, в каких случаях какой из алгоритмов является наиболее эффективным.
Задание B9 (двоичное дерево поиска)
Напишите работающую за линейное время рекурсивную процедуру, которая ищет вершину в двоичном дереве поиска. Дерево задается в файле.

Задание A10 (методы внешней сортировки: прямое слияние, естественное слияние, сбалансированное многопутевое слияние)
1) Написать три программы, реализующие эти алгоритмы.
2) Экспериментально сравнить реализации этих алгоритмов по времени исполнения на различных наборах данных, различных по объему и различных по упорядоченности (обратить внимание на крайние случаи). Оформить результаты экспериментов в табличном виде и построить графики зависимости времени выполнения программ от объема входных данных и упорядоченности входных последовательностей. Проверить экспериментально алгоритмы на устойчивость.
3) Сопоставить результаты с теоретическими оценками сложности алгоритмов (по времени исполнения). Сделать выводы о факторах, которых могли повлиять на точность измерений.
4) Сделать выводы, в каких случаях какой из алгоритмов является наиболее эффективным.
Задание B10 (классическое B-дерево)
Напишите работающую за линейное время рекурсивную процедуру, которая ищет вершину в двоичном дереве поиска. Дерево задается в файле.

Задание A11 (сортировка подсчетом + пораздрядная сортировка)
1) Написать программы, реализующие эти алгоритмы.
2) Экспериментально сравнить реализации этих алгоритмов по времени исполнения на различных наборах данных, различных по объему и различных по упорядоченности (обратить внимание на крайние случаи). Оформить результаты экспериментов в табличном виде и построить графики зависимости времени выполнения программ от объема входных данных и упорядоченности входных последовательностей. Проверить экспериментально алгоритмы на устойчивость.
3) Сопоставить результаты с теоретическими оценками сложности алгоритмов (по времени исполнения). Сделать выводы о факторах, которых могли повлиять на точность измерений.
4) Сделать выводы, в каких случаях какой из алгоритмов является наиболее эффективным.
Задание B11 (классическое B-дерево)
Напишите работающую за линейное время рекурсивную процедуру, которая ищет вершину в двоичном дереве поиска. Дерево задается в файле.

Задание A12 (Сортировка + анализ сложности по времени)
Сортировка Шелла.
1) Рассмотреть различные известные подходы (минимум три различных подхода) к выбору значений длин промежутков при сортировке Шелла, на которых будут находиться сортируемые элементы исходного массива ёмкостью N на каждом шаге алгоритма.
2) Написать программы, в которых реализуется сортировка Шелла с различными

подходами к выбору значений длин промежутков.
3) Сформировать несколько наборов данных (целых чисел) различного объема. Построить экспериментальный график зависимости времени выполнения кода (с различными подходами к выбору длин промежутков) от объема входных данных.
4) Сопоставить результаты с теоретическими оценками сложности алгоритмов (по времени исполнения). Сделать выводы о факторах, которых могли повлиять на точность измерений.
5) Сделать выводы об эффективности каждого из подходов к выбору значений длин промежутков.

Задание B12 (Дек, управляемый программой)
Дек (deq - double ended queue,т.е очередь с двумя концами) – структура данных, объединяющая стек и очередь. Элементы можно добавлять и брать с любого из двух концов.
Разработать класс (или классы), реализующий дек. Способ реализации дека на основе расширяемого массива. Выполнить требования инкапсуляции.
Разработать отдельный модуль, который будет управлять деком (стеком и очередью) с помощью программы, считанной из файла.
В программе могут содержаться следующие команды:
• PUSH (число) // поместить число в стек (в конец очереди)
• POP // извлечь число из стека (с конца очереди) и вывести на экран
• ADD (число) // поместить число в начало очереди (на вершину стека)
• REMOVE // извлечь число из начала очереди (с вершины стека) и вывести на экран
• PRINT // вывести содержимое дека на экран
• RESET // очистить содержимое дека
Предусмотреть решение проблемы излишнего расходования памяти при сравнительно большом количестве операций REMOVE против операций ADD.
При выполнении недопустимой команды выводить соответствующее сообщение об ошибке и переходить к следующей команде.
Входной текстовый файл содержит программу для управления деком. Результаты выполнения программы вывести на консоль.

Задание A13 (Рекурсия и нерекурсия + анализ сложности по времени и памяти)
1) Написать две программы вычисления ряда чисел Фибоначчи, одна из которых реализует рекурсивный алгоритм, а вторая – нерекурсивный. Ряд чисел задается соотношением: Ф[1]=Ф[2]=1; Ф[n]=Ф[n-1]+Ф[n-2].
2) Построить экспериментальный график зависимости времени выполнения кода двух программ и размера потребляемой памяти от длины вычисляемой последовательности чисел.
3) Сопоставить результаты с теоретическими оценками сложности алгоритмов (по времени исполнения) и оценками по потреблению памяти. Сделать выводы о факторах, которых могли повлиять на точность измерений.
4) Сделать выводы об эффективности рекурсивного и нерекурсивного алгоритма по использованию памяти и по времени выполнения.

Задание B13 (Инфиксная форма записи выражения)
В файле задано выражение в инфиксной форме записи (в обычной скобочной форме). В выражении могут быть целые положительные числа, знаки операций (* / +

–), круглые скобки. Написать программу, сохраняющую считанное выражение в двоичном дереве (в котором знаки операций будут родительскими вершинами, а числа - листьями), а также вычисляющую результат выражения. Дерево представить в виде отдельного класса (или классов). Результат выражения вывести в выходной файл.
Приоритеты операций в выражении:
• * / высокий
• + – низкий

Задание A14 (Рекурсия и нерекурсия + сортировка + анализ сложности по времени и памяти)
1) Написать две программы, реализующие алгоритм быстрой сортировки. Одна из программ должна быть реализацией рекурсивного алгоритм, вторая реализует нерекурсивный алгоритм с использованием стека отложенных заданий.
2) Оценить размер стека, необходимый для нерекурсивной версии алгоритма. Как обойтись стеком, глубина которого ограничена C log n, где n – число сортируемых элементов?
3) Сформировать несколько наборов данных (целых чисел) различного объема. Построить экспериментальный график зависимости времени выполнения кода двух программ от объема входных данных.
4) Сопоставить результаты с теоретическими оценками сложности алгоритмов (по времени исполнения). Сделать выводы о факторах, которых могли повлиять на точность измерений.
5) Сделать выводы об эффективности рекурсивного и нерекурсивного алгоритма по времени выполнения.

Задание B14 (Алгоритм сортировочной станции)
Инфиксная и постфиксная запись (обратная польская нотация). Обеспечить перевод инфиксного выражения в ОПЗ и вычислить его результат.
В инфиксную запись входят операнды, обозначаемы латинскими буквами, а также операции:
1. Арифметические операции (+, -, *, /);
2. Круглые скобки;
3. ^ (возведение в степени);
4. тригонометрические функции (sin, cos, tan, cot, asin, acos, atan, acot);
5. показательные функции (exp, ln, log);
6. степенные функции (sqrt - квадратный корень, sqrt3 - кубический корень);
7. Операции с массивами (обращение к элементу массива - это бинарная операция)
Программа должна считывать исходное выражение из файла expression.txt и выводить его вместе с результатом преобразования на экран.
В файле expression.txt помимо инфиксной записи в строках, начиная со второй, расположены записи вида
переменная = значение
из которых необходимо получить значения переменных для подстановки в выражение.
Для реализации написать собственный класс Stack.

Задание A15 (Поиск + анализ сложности по времени выполнения)
1) Написать программы, реализующие следующие методы поиска:
a. последовательный поиск;
b. бинарный поиск;
c. интерполяционный поиск.
2) Сформировать несколько наборов данных (целых чисел) различного объема. Исходный отсортированный по возрастанию массив целых чисел может генерироваться со случайным шагом. Построить экспериментальный график зависимости времени выполнения кода программ от объема входных данных.
3) Сопоставить результаты с теоретическими оценками сложности алгоритмов (по времени исполнения). Сделать выводы о факторах, которых могли повлиять на точность измерений.
4) Сделать выводы об эффективности методов поиска по времени выполнения.

Задание B15 (Ассоциативный индексированный массив)
Ассоциативный массив (словарь) – структура данных, позволяющая хранить пары вида «(ключ, значение)», при этом предполагается уникальность ключа в массиве.
Разработать класс (или классы), который бы позволял хранить данные целого типа в массиве, индексами (ключами) которого являются строки.
Спроектировать ассоциативный массив таким образом, чтобы доступ до элемента выполнялся за время O(log n). Использовать в основе массива двоичное дерево поиска.
Разработать отдельный модуль, который будет управлять массивом с помощью программы, считанной из файла.
В программе могут содержаться следующие команды:
• INSERT(“ключ”, значение) // добавить элемент в массив или заменить
// существующий
• FIND(“ключ”) // вывести значение или сообщить об его
// отсутствии
• REMOVE(“ключ”) // удалить элемент или сообщить об его отсутствии
При выполнении недопустимой команды выводить соответствующее сообщение об ошибке и переходить к следующей команде.
Входной текстовый файл содержит программу для управления массивом. Результаты выполнения программы вывести на консоль.

Задание A16 (Последовательность + анализ сложности на время исполнения)
Реализовать динамическую структуру данных - последовательность в двух вариантах: на основе расширяемого массива (вектора) и на основе двусвязного списка. В последовательности хранятся целые числа.
В обоих случаях структура должна поддерживать один набор операций:
1) Добавление элемента в начало, в конец или в середину списка (в последнем случае - с указанием индекса)
2) Удаление элемента по индексу
3) Чтение элемента по индексу и получение первого или последнего элемента списка

4) Перестановка двух элементов местами (по индексам)
5) Замена элемента (по индексу)
6) Определение является ли список пустым
7) Подсчет длины списка (число элементов)
8) Полная очистка списка
Предложить различные стратегии динамического увеличения длины списка при реализации его на основе расширяемого массива и выбрать один из вариантов.
2. Провести сравнительный анализ этих двух реализаций последовательности, оценив время выполнения каждой операции. При испытаниях выполнить тесты на различных длинах последовательностей. Результаты оформить в табличном виде и в виде графиков.
3. Сопоставить полученные результаты измерений с теоретической оценкой сложности алгоритмов (по времени исполнения) каждой операции над последовательностью. Результаты оформить в табличном виде. Сделать выводы о факторах, которых могли повлиять на точность измерений. По результатам тестов и теоретической оценки алгоритмов сделать выводы о применимости и эффективности использования вектора и двусвязного списка.
Задание B16 (Нерекурсивный алгоритм + обход дерева)
Написать нерекурсивную программу, печатающую все вершины двоичного дерева. При реализации использовать стек отложенных заданий. (А. Шень Программирование: теоремы и задачи. Задача 8.2.4).
Узлы дерева – символы латинского алфавита. Дерево задается в файле в формате: m [e [c [a], g [k]], s [p [o,s], y]]
Рисунок, поясниющий пример:

Задание A17 (Алгоритм сортировки вставками и быстрой сортировки)
1) Написать две программы, реализующие алгоритм сортировки вставками и алгоритм быстрой сортировки (рекурсивный) для массива целых чисел.
2) Экспериментально сравнить реализации этих двух алгоритмов по времени исполнения на различных наборах данных, различных по объему и различных по упорядоченности (обратить внимание на крайние случаи – уже отсортированный массив и массив, отсортированный в обратном порядке). Оформить результаты экспериментов в табличном виде и построить графики зависимости времени выполнения программ от объема входных данных и упорядоченности входных последовательностей. Проверить экспериментально алгоритмы на устойчивость.
3) Сопоставить результаты с теоретическими оценками сложности алгоритмов (по времени исполнения). Сделать выводы о факторах, которых могли повлиять на точность измерений.
4) Сделать выводы, в каких случаях какой из алгоритмов является наиболее эффективным.

Задание B17 (Рекурсия и нерекурсия)
Написать рекурсивную и нерекурсивную версию программы для нахождения последовательности перемещений колец в задаче о ханойских башнях. При реализации нерекурсивного алгоритма использовать стек отложенных заданий, элементами которого будут тройки (i,m,n). Каждая такая тройка интерпретируется как заказ «переложить i верхних дисков с m-го стержня на n-ый».

Задание A18 (Пирамидальная сортировка)
1) Реализовать алгоритм пирамидальной сортировки (англ. Heapsort, «Сортировка кучей») — алгоритм сортировки, работающий в худшем, в среднем и в лучшем случае (то есть гарантированно) за Θ(n log n) операций при сортировке n элементов. Количество применяемой служебной памяти не зависит от размера массива (то есть, O(1)).
2) Сформировать несколько наборов данных (целых чисел) различного объема. Построить экспериментальный график зависимости времени выполнения кода программы от объема входных данных.
3) Сопоставить результаты с теоретическими оценками сложности алгоритмов (по времени исполнения). Сделать выводы о факторах, которых могли повлиять на точность измерений.
4) Обосновать, что в реализации алгоритма количество применяемой служебной памяти действительно не зависит от размера сортируемого массива.
5) Сделать выводы.

Задание B18 (Очередь с приоритетами)
Приоритетная очередь это очередь, в которой важно не то, кто встал последним (порядок помещения в неё не играет роли), а кто главнее. Более точно, при помещении в очередь указывается приоритет помещаемого объекта (будем считать приоритеты целыми числами), а при взятии из очереди выбирается элемент с наибольшим приоритетом (или один из таких элементов). Реализовать приоритетную очередь так, чтобы помещение и взятие элемента требовали логарифмического числа действий (от размера очереди).
Задание A19 (Простые алгоритмы сортировки + анализ сложности по времени)
1) Написать три программы, реализующие простейшие алгоритмы сортировки: алгоритм сортировки пузырьком, алгоритм сортировки вставками и алгоритм сортировки выбором.
2) Экспериментально сравнить реализации этих алгоритмов по времени исполнения на различных наборах данных, различных по объему и различных по упорядоченности (обратить внимание на крайние случаи – уже отсортированный массив и массив, отсортированный в обратном порядке). Оформить результаты экспериментов в табличном виде и построить графики зависимости времени выполнения программ от объема входных данных и упорядоченности входных последовательностей. Проверить экспериментально алгоритмы на устойчивость.
3) Сопоставить результаты с теоретическими оценками сложности алгоритмов (по времени исполнения). Сделать выводы о факторах, которых могли повлиять на точность измерений.
4) Сделать выводы, в каких случаях какой из алгоритмов является наиболее эффективным.

Задание B19 (Ассоциативный массив, хэш-таблица)
Ассоциативный массив (словарь) – структура данных, позволяющая хранить пары вида «(ключ, значение)», при этом предполагается уникальность ключа в массиве. Разработать класс (или классы), который бы позволял хранить данные целого типа в массиве, индексами (ключами) которого являются строки.
Спроектировать ассоциативный массив таким образом, чтобы доступ до элемента выполнялся за время O(1), т.е. не зависел от количества элементов в массиве.

Использовать в основе массива реализацию в виде хэш-таблицы. Предусмотреть возникновение коллизий.
Разработать отдельный модуль, который будет управлять массивом с помощью программы, считанной из файла.
В программе могут содержаться следующие команды:
• INSERT(“ключ”, значение) // добавить элемент в массив или заменить
// существующий
• FIND(“ключ”) // вывести значение или сообщить об его
// отсутствии
• REMOVE(“ключ”) // удалить элемент или сообщить об его отсутствии
При выполнении недопустимой команды выводить соответствующее сообщение об ошибке и переходить к следующей команде.
Входной текстовый файл содержит программу для управления массивом. Результаты выполнения программы вывести на консоль.

Задание A20 (Внешние алгоритмы сортировки слиянием + анализ сложности по времени)
Внешняя сортировка – сортировка больших объемов данных, которые не помещаются целиком в оперативной памяти и не могут быть отсортированы обычными (внутренними) алгоритмами сортировки, например, пузырьком. Активно применяется в базах данных. Использует исходный файл с данными (он же и файл с результатом) и несколько вспомогательных файлов.
1) Написать три программы, реализующие алгоритмы внешней сортировки: прямым слиянием, естественным слиянием и многопутевым слиянием.
2) Исходный файл содержит данные о странах в виде таблицы формата (*.csv) , столбцы: название страны, континент, столица, площадь, численность населения.
3) Экспериментально сравнить реализации этих алгоритмов по времени исполнения на различных наборах данных, различных по объему и типу (разных столбцах таблицы) и различных по упорядоченности (обратить внимание на крайние случаи – уже отсортированный набор и набор, отсортированный в обратном порядке). Оформить результаты экспериментов в табличном виде и построить графики зависимости времени выполнения программ от объема входных данных и упорядоченности входных последовательностей.
4) Сопоставить результаты с теоретическими оценками сложности алгоритмов (по времени исполнения). Сделать выводы о факторах, которых могли повлиять на точность измерений.
5) Сделать выводы, в каких случаях какой из алгоритмов является наиболее эффективным.
Задание B20 (Двоичное дерево)
Напишите работающую за линейное время (O(n)) рекурсивную процедуру, которая печатает ключи всех вершин двоичного дерева. Двоичное дерево задается в файле в следующем виде:
index/ key/ left/ right
1 12 7 3
2 15 8 NULL
3 4 10 NULL
4 10 5 9
5 2 NULL NULL

б 18 1 4
7 7 NULL NULL
8 11 6 2
9 21 NULL NULL
10 5 NULL NULL
ROOT 6 1 NULL

В конце файла указывается индекс вершины – корня дерева.
В столбце key –указывается значение, которое хранится в вершине дерева (целое число)
В столбцах left и right указываются индексы дочерних вершин для данной вершины (Если в обоих столбцах – NULL, то вершина является листом)

Задание A21 (Сортировка с помощью двоичного дерева (древесная сортировка) + анализ сложности по времени)
1) Реализовать алгоритм древесной сортировки — универсальный алгоритм сортировки, заключающийся в построении двоичного дерева поиска по ключам массива (списка), с последующей сборкой результирующего массива путём обхода узлов построенного дерева в необходимом порядке следования ключей. Данная сортировка является оптимальной при получении данных путём непосредственного чтения с потока (например с файла, сокета или консоли).
2) В качестве источника данных для сортировки выбрать файл с однопроходным последовательным чтением и одновременным формированием дерева.
3) Сформировать несколько наборов данных (целых чисел) различного объема. Построить экспериментальный график зависимости времени выполнения кода программы от объема входных данных.
4) Сопоставить результаты с теоретическими оценками сложности алгоритмов (по времени исполнения). Сделать выводы о факторах, которых могли повлиять на точность измерений.
5) Сделать выводы.
Задание B21 (Ориентированный граф + обход в глубину и ширину)
1. Задана матрица смежности или инцидентности (по выбору).

2. Построить граф, каждый элемент которого будет представлен в виде:

Дополнительно в структуре предусмотреть хранение веса дуг графа, заданных в матрицах.
Реализовать алгоритм обхода в глубину (рекурсивно и не используя рекурсию, применяя стек отложенных заданий) и ширину (применяя очередь) полученного графа. Показать на экране процесс перебора вершин.
Задание A22 (Индексный поиск (обычные и инвертированные индексы) + анализ сложности по времени выполнения)
Прямой индекс – древовидная структура ключей и указателей на данные, связанные с ключами. Поиск по ключу в индексе осуществляется за O(log n).
Инвертированный индекс (полнотекстовый индекс) – структура данных, в которой для каждого слова коллекции документов в соответствующем списке перечислены все места в коллекции, в которых оно встретилось. Инвертированный индекс используется для поиска по текстам, например в поисковых системах Yandex или

Google.
1) Файл с исходными данными в формате (*.csv) представляет таблицу с двумя колонками: название страны и её краткое описание (небольшой текст до 5 предложений). Количество стран пусть варьирует согласно пункту 3 (см. ниже).
2) Написать программу, строящую в оперативной памяти прямой индекс по названию страны и инвертированный индекс по словам в тексте её описания. Предусмотреть операцию нормализации простого индекса после его построения.
3) Сформировать несколько наборов данных различного объема. Построить экспериментальный график зависимости времени поиска по каждому из индексов от объема данных. Поиск названия страны в таблице (прямой индекс) и поиск страны по словам из её описания (инвертированный индекс).
4) Сопоставить результаты с теоретическими оценками сложности алгоритмов (по времени исполнения). Сделать выводы о факторах, которых могли повлиять на точность измерений.
5) Сделать выводы об эффективности методов поиска по времени выполнения.

Задание B22 (Массив с возможностью быстрого поиска элементов)
Реализовать структуру данных, которая имеет все те же операции, что и массив длины n, а именно:
1. начать работу,
2. положить в i-ю ячейку число x,
3. узнать, что лежит в i-й ячейки,
а также операцию
4. указать номер минимального элемента
(точнее номер одного из минимальных элементов). Количество действий для всех операций должно быть не более C log n, не считая операцию «начать работу» (которая требует не более Cn действий).
// ==========
Вопросы для самопроверки:

1. Вычислительная сложность алгоритмов
Алгоритмы и их классификация. Вычислительная сложность алгоритмов.
Контрольные вопросы
1. Какие критерии используются при выборе алгоритмов?
2. Как оценивается трудоемкость алгоритма?
3. Что такое О-нотация и для чего она используется?
4. Какие группы функций можно выделить с помощью О-нотации?
5. Какие рекомендации следует использовать при выборе алгоритмов с помощью О-нотации?
2. NP-полные и труднорешаемые задачи
Массовая и индивидуальная задачи. Сложность алгоритма. Полиномиальные алгоритмы и класс P. Недетерминированные алгоритмы и класс NP. Полиномиальная преобразуемость задач. NP-трудные и NP-полные задачи.
Контрольные вопросы
1. Как оценивается сложность алгоритма?
2. Чем отличаются алгоритмы класса P от алгоритмов класса NP?
3. Что такое «полиномиальная приводимость задач»?
4. Что такое «Задача принятия решения»?

5. Какая задача является NP-трудной?
3. Метод ветвей и границ
Метод декомпозиции. Пример.
Поиск с возвратом. Общий алгоритм. Пример.
Метод ветвей и границ. Применение метода при решении задачи о рюкзаке, задаче о назначениях и задаче коммивояжера.
Динамическое программирование. Пример. Нисходящее и восходящее динамическое программирование. Задача определения наиболее длинной общей подпоследовательности.
Задача построения оптимального бинарного дерева поиска.
Контрольные вопросы
1. В чем заключается метод ветвей и границ.
2. Как строятся оценочные функции состояний? Приведите примеры.
3. Что такое динамическое программирование?
4. В каких случаях применяется динамическое программирование?
5. В чем заключается восходящий и нисходящий методы динамического программирования?
4. Поиск подстрок
Задача поиска подстрок. Простейший алгоритм. Алгоритм Кнута-Морриса-Пратта. Алгорит Бойера-Мура. Алгоритм Рабина-Карпа.
Контрольные вопросы
1. Что такое префикс–функция, ассоциированная с образцом?
2. Что такое эвристика стоп–символа?
3. В чем состоит эвристика безопасного суффикса?
4. Что такое собственный префикс?
5. Что такое собственный суффикс?
5. Фундаментальные алгоритмы на графах
Графы: определения и примеры. Представления графов: матрица инциденций, матрица смежности, список пар, списки смежности.
Фундаментальные алгоритмы на графах: Поиск в графе. Поиск в ширину. Поиск в глубину.
Остовные деревья графа. Связные компоненты. Построение и свойства остовных деревьев при поиске в глубину и в ширину.
Эйлеров путь в графе. Алгоритм построения Эйлерова пути.
Циклы: Фундаментальное множество циклов графа. Алгоритм отыскания фундаментального множества циклов в графе.
Минимальное остовное дерево. Алгоритм Прима. Алгоритм Крускала.
Гамильтонов путь в графе. Нахождение Гамильтонова пути в графе с помощью алгоритма с возвратом
Контрольные вопросы
1. Какие способы можно использовать для представления графов как структур данных?
2. Какие структуры данных необходимы для реализации списков смежности?
3. Как реализуются операции добавления и удаления ребер в графе?
4. Как реализуются операции добавления и удаления вершин в графе?
5. Какие шаги включает в себя алгоритм поиска в глубину?
6. Кратчайшие пути во взвешенном графе
Кратчайшие пути в графе. Кратчайшие пути от фиксированной вершины. Алгоритм

Форда-Беллмана. Случай неотрицательных весов: алгоритм Дейкстры. Кратчайшие пути в бесконтурном графе. Топологическая сортировка. Кратчайшие пути между всеми парами вершин. Матрица смежности, матрица достижимости и транзитивное замыкание отношения, алгоритм Уоршалла. Алгоритм Флойда-Уоршалла вычисления расстояний между всеми парами вершин, одновременное построение путей.

Контрольные вопросы

1. Поясните, в чем состоит алгоритм Форда-Беллмана.
2. Поясните, в чем состоит алгоритм Дейкстры.
3. На каком принципе строится алгоритм нахождения кратчайших путей в бесконтурном графе?
4. Что такое «транзитивное замыкание».
5. Поясните, в чем состоит алгоритм Уоршалла.

7. Задачи о потоках

Задача о максимальном потоке. Введение. Транспортные сети и потоки. Алгоритм Форда-Фалкерсона. Алгоритм Эдмондса-Карпа. Алгоритм Диница.

Задача о потоке минимальной стоимости.

Контрольные вопросы

1. Что такое «Транспортная сеть»?
2. Что такое «Остаточная сеть»?
3. Что такое «Увеличивающий путь»?
4. В чем заключается алгоритм Форда-Фалкерсона?
5. В чем заключается алгоритм Эдмондса-Карпа?

8. Задача о паросочетании

Паросочетание. Алгоритм определение максимального паросочетания. Задача о полном паросочетании.

Контрольные вопросы

1. Что такое «Двудольный граф»?
2. Что такое «Паросочетание»?
3. Что такое «Чередующаяся цепь»?
4. В чем заключается задача о полном паросочетании?
5. Опишите алгоритм решения задачи о полном паросочетании.

9. Решение задач с помощью алгоритма с возвратом

Гамильтонов путь. Клики графа.

Контрольные вопросы

1. В чем заключается алгоритм с возвратом?
2. Как применяется алгоритм с возвратом при нахождении гамильтонового пути в графе?
3. Как применяется алгоритм поиска с возвратом при решении задачи нахождения всех клик графа?

10. Методы внутренней сортировки

Сортировка включением

Обменная сортировка

Сортировка выбором

Сортировка разделением (Quicksort)

Сортировка с помощью дерева (Heapsort)

Сортировка со слиянием

Сравнение методов внутренней сортировки

11. Методы внешней сортировки
Прямое слияние
Естественное слияние
Сбалансированное многопутевое слияние
Многофазная сортировка
Улучшение эффективности внешней сортировки за счет использования основной памяти

12. Методы поиска в основной памяти на основе деревьев
Двоичные деревья
Сбалансированные двоичные деревья
Деревья оптимального поиска
Деревья цифрового поиска

13. Методы хэширования для поиска в основной памяти
Совершенное хэширование
Коллизии при хэшировании и способы их разрешения
Линейное зондирование
Двойное хэширование
Использование цепочек переполнения

14. Методы поиска во внешней памяти на основе деревьев
Классические B-деревья
B+-деревья
Разновидности B+-деревьев для организации индексов в базах данных
R-деревья и их использование для организации индексов в пространственных базах данных

15. Методы хэширования для поиска во внешней памяти
Расширяемое хэширование
Линейное хэширование

9. Учебно-методическое и информационное обеспечение практики

Печатная учебно-методическая документация

а) основная литература:

1. Шилдт, Г. Теория и практика C++ Руководство для профессионалов. - СПб.: BHV-Санкт-Петербург, 1996. - 412 с.
2. Подбельский, В. В. Язык Си++ Учеб. пособие для вузов по спец."Прикл. математика" и "Вычисл. машины, комплексы, системы и сети". - 2-е изд., перераб. и доп. - М.: Финансы и статистика, 1996. - 559,[1] с. ил., табл.
3. Лафоре, Р. Объектно-ориентированное программирование в C++ [Текст] Р. Лафоре ; пер. с англ. А. Кузнецов и др. - 4-е изд. - СПб. и др.: Питер, 2011. - 923 с. ил.

б) дополнительная литература:

1. Фаулер, М. Рефакторинг: улучшение существующего кода Пер. с англ. М. Фаулер при участии К. Бека и др. - СПб.: Символ-Плюс, 2004. - 430 с.
2. Алгоритмы : построение и анализ [Текст] Т. Кормен и др.; пер. с англ. И. В. Красикова. - 3-е изд. - М. и др.: Вильямс, 2015. - 1323 с. ил.
3. Кнут, Д. Э. Искусство программирования Т. 1 Основные алгоритмы Учеб. пособие: Пер. с англ. Под общ. ред. Ю. В. Козаченко. - 3-е изд., испр. и доп. - М. и др.: Вильямс, 2000. - 712 с.
4. Ахо, А. В. Структуры данных и алгоритмы [Текст] учеб. пособие для вузов А. В. Ахо, Д. Э. Хопкрофт, Д. Д. Ульман ; пер. с англ. А. А. Манько. - М. и др.: Вильямс, 2003. - 382 с. ил.

из них методические указания для самостоятельной работы студента:

1. Программирование в объектах на СИ++ : Учеб. пособие / Е. А. Конова, Е. М. Сартасов, Б. М. Суховилов; Юж.-Урал. гос. ун-т, Каф. Информатика; Юж.-Урал. гос. ун-т, Каф. Информатика; ЮУрГУ

Электронная учебно-методическая документация

№	Вид литературы	Наименование разработки	Наименование ресурса в электронной форме	Доступность (сеть Интернет / локальная сеть; авторизованный / свободный доступ)
1	Основная литература	Подбельский, В.В. Курс программирования на языке Си [Электронный ресурс] : учеб. / В.В. Подбельский, С.С. Фомин. — Электрон. дан. — Москва : ДМК Пресс, 2012. — 384 с.	Электронно-библиотечная система Издательства Лань	Интернет / Свободный
2	Дополнительная литература	Ашарина, И.В. Объектно-ориентированное программирование в C++: лекции и упражнения [Электронный ресурс] : учеб. пособие — Электрон. дан. — Москва : Горячая линия-Телеком, 2012. — 320 с.	Электронно-библиотечная система Издательства Лань	Интернет / Свободный

10. Информационные технологии, используемые при проведении практики

Перечень используемого программного обеспечения:

1. Microsoft-Visual Studio(бессрочно)

Перечень используемых информационных справочных систем:

Нет

11. Материально-техническое обеспечение практики

Место прохождения практики	Адрес места прохождения	Основное оборудование, стенды, макеты, компьютерная техника,

		предустановленное программное обеспечение, обеспечивающие прохождение практики
Кафедра Электронные вычислительные машины ЮУрГУ		Компьютер кафедры или собственная машина студента