УТВЕРЖДЕНО
Проректор по учебной работе
и довузовскому образованию
А. А. Воронов
09 января 2018 г.

ПРОГРАММА

по дисциплине: **Компьютерные технологии**
по направлению подготовки: 03.03.01 «Прикладные математика и физика»
физтех-школа: **ФАКТ**
факультет: **ФАЛТ**
кафедра: **информатики и вычислительной математики**
курс: 2
семестр: 4

Трудоёмкость:	Экзамен – нет
вариативная часть – 2 зачет. ед.	Зачет дифф. – 4 семестр
лекции – 15 (час)	Задания – 2
практические (семинарские)	Контрольные работы – 1
занятия – 30 (час)	Курсовая работа – 1
лабораторные занятия – нет	Самостоятельная работа – 45 часов

ВСЕГО АУДИТОРНЫХ ЧАСОВ – 45

Программу составил профессор, д.ф.-м.н. А. Г. Тормасов

Программа принята на заседании
кафедры информатики и вычислительной математики
14 ноября 2017 г.

Заведующий кафедрой
чл.-корр. РАН И. Б. Петров

1. Процесс написания программ. Оформление текстов, требования к текстам и комментариям. Сопровождение программ. Документация на ПО, *SDP* (*software development plan* – план разработки ПО).

2. Базовые основы элементарной техники программирования. Технические основы программной реализации формальных структур данных, итераторы. Списки, очереди, стеки, наборы, упорядоченные наборы, массивы, деревья, хранение графов, *hash*-таблицы; примеры использования структур данных, как выбрать структуру, соответствующую задаче. Применение *hash*-таблиц.

3. Безопасность ПО. Написание программ без «дырок». Техника кодирования защищенных программ и типичные ошибки. Переполнение буфера, определение уровня доступа, работа с минимально возможными привилегиями, криптография и ее корректное применение, предохранение секретных данных, работа с входными данными, проблемы разных путей доступа к одним и тем же данным, запросов к базам данных и веб-страницам, а также проблемы поддержки интернационального ПО. Моделирование угроз. Классификация опасностей *STRIDE* – *Spoofing* (подмена данных), *Tampering* (подделка и изменение содержания данных), *Repudiation* (незаконный отказ от проведенной операции), *Information disclosure* (разглашение информации), *Denial of service* (отказ в обслуживании), *Elevation of privilege* (незаконное поднятие привилегий). Методика оценки риска *DREAD* – *Damage potential* (что может быть сломано), *Reproducibility* (повторяемость), *Exploitability* (пригодность угрозы для использования), *Affected users* (на каких пользователей повлияет), *Discoverability* (возможно ли детектировать факт использования).

4. Эволюция современного аппаратного обеспечения и ее влияние на программное обеспечение. Гипертредовые (многопоточные) и многоядерные процессоры, универсальные графические (*GPU*) процессоры и новые возможности по их использованию.

5. Параллельное программирование. Параллельные программы – от работы с разделяемой памятью, использования массивно-параллельных компьютеров и до распределенных расчетов на многих физических компьютерах. Декомпозиция задач на параллельные куски. Параллелизм данных, параллелизм кода. Паттерны параллельного программирования: параллелизм на уровне задач – декомпозиция задачи, «разделяй и властвуй» – декомпозиция задач и данных, геометрическая декомпозиция – декомпозиция данных, конвейерное исполнение – декомпозиция потока данных, «фронт волны» – декомпозиция данных с «многомерными» зависимостями. Пример типового шаблона программирования – пул нитей.

6. Работа с разделяемой памятью. Синхронизационные примитивы (низкоуровневые атомарные команды, критические секции, взаимоисключающая блокировка – *mutex*, рекурсивная блокировка – *lock*, блокировка чтения-записи – *read/write lock*, многопроцессорная блокировка – *spinlock*, семафоры, барьеры). Реализация одних примитивов через другие, относительная «мощность» примитивов. Задача о консенсусе как способ оценки примитивов.

7. Техническая специфика параллельных программ – производительность, условия «гонки» – *race condition*, взаимная блокировка – *deadlock*, повторно-входимые программы, потоко-безопасные (*thread-safe*) библиотеки. Неблокирующие примитивы синхронизации, транзакционная память. Организация высоко-производительных параллельных вычислений.

8. Проблемы, специфические для параллельного исполнения многонитевых программ – синхронизация (между несколькими нитями), коммуникация (проблемы полосы пропускания и задержек, связанных с обменом данными между нитями), балансировка нагрузки (между нитями), масштабируемость (эффективность использования многих нитей). Написание высокопроизводительных программ, оценка условий и выбор способов реализации.

9. Принципы и философия ООП в языках, программных системах и операционных системах. Инкапсуляция, полиморфизм и наследование/агрегирование. Динамический и статический подход в описании классов.

10. Краткий обзор ООП реализации в языках C++. Интерфейсы, полиморфизм и перегрузка операторов в C++. Параметризованные классы. Дружественные функции. Потоки ввода-вывода в C++. Наследование как один из вариантов комбинирования объектов.

11. Адресное пространство приложения: куча, стек и статические объекты. Динамическая инициализация и клонирование объектов. Хранение объектов в адресном пространстве. Виртуальные функции. Наследование абстрактных классов в C++, чисто виртуальные функции.

12. Динамическая идентификация и приведение типов (*RTTI*). Обработка исключительных ситуаций. Разные способы «универсализации» алгоритмов – от абстрактных типов данных «*ADT*» до стандартной библиотеки шаблонов (*Standard Template Library*) языка программирования C++ (*STL*).

13. Краткий сравнительный обзор ООП реализации в языках C++ и *ObjectiveC*, позднее и раннее связывание. Техническая организация ООП поддержки языков программирования для позднего связывания.

14. *Event-driven* и *message-driven* программирование на примере *XWindows Widgets*, *Mac OS X Interface Builder* и подсистемы *GDI MS Windows*. Прин-

ципы агрегирования *COM*-объектов. Принципы поддержки ООП в операционных системах и языках, общее и различное. Реализация исключений в языке C++ и в ОС *Windows*.

Контрольные вопросы и задания по базовой и вариативной части дисциплины для промежуточной аттестации по итогам освоения дисциплины

Задачи по курсу

2 задания на семестр и курсовая работа. Задачи для первых двух заданий указаны ниже.

Каждый студент должен сделать по одной задаче из каждого задания.

Тема курсовой работы дается по усмотрению преподавателя (с учетом пожеланий студентов, которые могут предлагать свои темы), результат реализации должен включать взаимодействие с пользователем (пользовательский интерфейс в виде меню, желательно графического, хотя возможны и текстовые версии при условии использования ООП библиотек). Курсовая работа может выполняться коллективно (2–3 чел.), должна включать параллельное исполнение компонент (например, включать нити или несколько взаимодействующих процессов) и использовать примитивы синхронизации (обязательно). Размер курсовой – не менее 600 строк на C# или C++, написанных одним человеком. Также должны использоваться коллекции данных по выбору автора (АТД – *ATL*, *STL* и т.д.).

Тема работы утверждается не позже, чем в феврале.

Требования к оформлению

Текст программы должен быть оформлен профессиональным образом. Отдельно должен быть предоставлен файл (файлы) с текстом решения задачи, тестом на нее (если нужен) и вставляемые файлы заголовков. То есть решение задачи состоит не менее чем из двух файлов (*.h, .CPP*), чаще трех (включая тест). Запрещается вставка одного *C/CPP* файла в другой, поощряется использование make-файла или проектов (но не обязательно).

Комментарии в тексте программы: обязаны присутствовать в файлах в начале, должны быть отдельно написаны к каждой функции и в коде по тексту (в среднем каждая третья строка должна содержать комментарий); текст должен выглядеть красиво (отступы и т.д. должны быть оформлены нормально, не должно использоваться более 132 символов в строке), программа не должна иметь неиспользуемых или ненужных кусков (закомментированных). Примером оформления являются *samples*, приложенные к соответствующим дистрибутивам компиляторов (примеры *SDK Windows*, *Linux kernel sources* и т.д.).

Обязательно:
необходимо проверять все параметры функций, используя *assert()* или аналоги. Необходимо проверять все коды возврата функций (особенно *malloc* и системных вызовов).
Желательно:
составление для курсовой работы *Software Development Plan* (*SDP*) – по усмотрению преподавателя.

Задание 1

(срок сдачи 13–18 марта)

Основная цель – правильное оформление текста задачи, проверка всех требуемых параметров и т.д. Реализовать только на языке С (не С++/C#).
Реализовать одну из структур данных, итератор по ней и тест (три файла: алгоритм, файл заголовков, тест). Тест должен демонстрировать работоспособность структуры. Помните, что требуемых структур может быть много (более одного экземпляра!):
1) односвязный список;
2) двусвязный список;
3) очередь;
4) очередь с приоритетами;
5) стек;
6) набор (*set*);
7) упорядоченный набор с введением функции отношения;
8) массив произвольного размера;
9) массив упорядоченный;
10) бинарное дерево;
11) сбалансированное дерево;
12) хэш-таблица с открытой адресацией (хэшировать текстовые строки произвольной длины);
13) хэш-таблица с цепочками (алгоритм из книги Д. Кнута «Искусство программирования. Сортировка и поиск», хэшировать бинарные данные произвольной длины типа *BLOB – Binary Large Object*);
14) битовый массив (то есть значения принимаются битами, а адресация – по номеру бита);
15) направленный граф (в виде квадратной таблицы отношений $M(i,j) = 1$, если есть путь из i в j узел).

Задание 2

(*срок сдачи 3–8 апреля*)

Требования к оформлению – такие же. Программа должна быть выполнена на *C++* и в обязательном порядке должна использовать чужие библиотеки абстрактных типов данных (например, *Borland ClassLib, MS MFC/Atl/Stl, GNU libstdc++, STL* и др.) – любую из них.

Задачи должны быть выполнены на C++ или C# (для платформы *Windows*) с использованием более одной нити или процесса по выбору студента, согласованному с преподавателем. Для этого под *Windows* может быть использована библиотека функций *Win32 API* (или более высокоуровневые библиотеки). Под *Unix*: *POSIX Threads*.

По желанию студента задача может быть дополнена графическим интерфейсом.

Единицу исполнения (нить или процесс) ниже будем называть *вычислителем*.

Задачи с использованием деревьев

Задачи этого раздела используют класс *Tree* для создания древесной структуры данных.

1. Поиск ключа узла дерева

Исходные данные: дерево, каждый элемент которого хранит уникальный, случайно сгенерированный ключ; ключ, узел с которым требуется найти, количество используемых вычислителей.

Цель: написать параллельную программу для поиска данного ключа.

Программа должна создать заданное пользователем дерево, считать количество используемых вычислителей и ключ для поиска, найти элемент дерева, содержащий этот ключ, и вывести информацию об этом узле.

2. Поиск максимального элемента дерева

Исходные данные: дерево, каждый элемент которого хранит уникальный, случайно сгенерированный ключ, количество используемых вычислителей.

Цель: написать параллельную программу для поиска максимального ключа.

Программа должна создать заданное пользователем дерево, считать количество используемых вычислителей, найти элемент дерева, содержащий максимальный из ключей и вывести информацию об этом узле.

Переменная, содержащая максимальный элемент, должна быть общей для всех вычислителей. Доступ к ней – синхронизован.

3. Поиск суммы всех элементов дерева

Исходные данные: дерево, каждый элемент которого хранит уникальный, случайно сгенерированный ключ, количество используемых вычислителей.

Цель: написать параллельную программу для поиска суммы всех ключей дерева.

Программа должна создать заданное пользователем дерево, считать количество используемых вычислителей, найти и вывести сумму всех ключей дерева.
Переменная, содержащая сумму ключей, должна быть общей для всех вычислителей. Доступ к ней – синхронизован.

4. Поиск количества узлов дерева

Исходные данные: дерево, количество используемых вычислителей.
Цель: написать параллельную программу для поиска количества узлов в дереве.
Программа должна создать заданное пользователем дерево, считать количество используемых вычислителей, найти и вывести количество всех узлов в дереве.
Переменная, содержащая максимальный элемент, должна быть общей для всех вычислителей. Доступ к ней – синхронизован.
Возможные алгоритмы распараллеливания работы для задач 1–4:
a) Пусть есть *N* вычислителей. У первого – ссылка на корень дерева. Он берет одно из поддеревьев для поиска себе, остальные раздает свободным вычислителям. Далее вычислители поступают точно так же. Когда свободных вычислителей не остается, каждый выполняет обычный поиск по своему поддереву.
b) Пусть есть *N* вычислителей, и выбран вычислитель-мастер. Он сам проводит поиск ключа по дереву в ширину до тех пор, пока не найдет *N* независимых поддеревьев, после чего раздает их для поиска остальным вычислителям.
Усложнение для задач 1–3: пусть каждый узел дерева содержит дополнительную информацию – количество узлов под ним. К основной цели задачи добавляется равномерное распределение загрузки вычислителей: каждый должен получить поддерево (поддеревья) с примерно одинаковым количеством узлов (отличающимся не более чем на единицу).

Вычислительные задачи

5. Вычисление интеграла численными методами

Исходные данные: функция одной переменной, отрезок интегрирования, количество вычислителей.
Цель: написать параллельную программу для вычисления интеграла от данной функции по данному отрезку.
Программа должна считать функцию, отрезок интегрирования, количество вычислителей, посчитать интеграл в несколько вычислителей, вывести результат.
Предполагаемый алгоритм распараллеливания: использовать параллелизм по отрезку. Каждый вычислитель считает интеграл по своей части отрезка

с помощью одного из стандартных методов (трапеции, Симпсона, Ньютона–Лейбница и т.д.). Результаты отдельных вычислителей суммируются. Численный метод выбирается преподавателем.

6. Вычисление интеграла методом Монте-Карло

Исходные данные: функция одной переменной, отрезок интегрирования, количество вычислителей.

Цель: написать параллельную программу для вычисления интеграла от данной функции по данному отрезку методом Монте-Карло.

Программа должна считать функцию, отрезок интегрирования, количество вычислителей, посчитать интеграл в несколько вычислителей, вывести результат.

Предполагаемый алгоритм распараллеливания:

a) Использовать параллелизм по отрезку. Каждый вычислитель считает интеграл по своей части отрезка. Результаты отдельных вычислителей суммируются.

b) Использовать параллелизм по количеству испытаний в методе Монте-Карло. Необходимое количество испытаний делится между N вычислителями. Результаты испытаний обобщаются.

7. Вычисление n-мерного интеграла от функции $f(x_1,x_2, \ldots, x_n) = f_1(x_1)\ldots f_n(x_n)$ методом Монте-Карло

Исходные данные: функция одной переменной, отрезок интегрирования, количество вычислителей.

Цель: написать параллельную программу для вычисления интеграла от данной функции по данному отрезку методом Монте-Карло.

Программа должна считать функцию, отрезок интегрирования, количество вычислителей, посчитать интеграл в несколько вычислителей, вывести результат.

Вариации:

a) Область имеет вид $[a_1, b_1]\times\ldots\times[a_n, b_n]$.

Предполагаемый алгоритм распараллеливания: использовать параллелизм количеству измерений. Каждый вычислитель просто вычисляет свой интеграл.

b) Область имеет произвольный вид.

Предполагаемый алгоритм распараллеливания: использовать параллелизм количеству измерений. В формуле метода Монте-Карло вычислители-рабочие считают значения функций (расчет может быть громоздким) и отсылают их вычислителю-мастеру, который производит суммирование и вычисление окончательного результата.

8. Параллельное перемножение матриц

Исходные данные: две матрицы $n\times m$ и $m\times n$, количество вычислителей.

Цель: написать программу для перемножения матриц параллельно несколькими вычислителями.
Программа должна считать *m, n*, две матрицы, количество вычислителей; распараллелить процесс их перемножения.
Предполагаемый алгоритм распараллеливания: поскольку каждый элемент матрицы-результата рассчитывается независимо от других, он может быть определен отдельным вычислителем. Вычислитель-мастер занимается распределением работы между остальными. Он хранит базу свободных вычислителей, куда синхронизовано заносятся не имеющие работы и откуда берутся рабочие для вычисления следующего элемента матрицы.

9. Параллельное вычисление детерминанта матрицы заданного размера (параллелизм по дереву рекурсии)

Исходные данные: матрица $n \times n$, количество вычислителей.
Цель: написать программу для расчета детерминанта матрицы параллельно несколькими вычислителями.
Программа должна считать *n*, матрицу, количество вычислителей; распараллелить процесс вычисления детерминанта матрицы.
Предполагаемый алгоритм распараллеливания: предполагается вычислять определитель методом Гаусса – приведением к верхнетреугольной матрице и далее перемножением элементов диагонали. Надо придумать способ распределения подзадач по вычислителям, например, получив дерево вычислений, оно же дерево рекурсии. А примеры распараллеливания (раздачи элементов дерева вычислителям) описаны в задаче 1 раздела *Задачи с использованием деревьев*.

10. Решение уравнения $Ax = b$ (A – матрица) методом Крамера: параллелизм по элементам столбца решения

Исходные данные: невырожденная матрица $n \times m$, столбец $n \times 1$, количество вычислителей.
Цель: написать программу для расчета решения уравнения $Ax = b$ параллельно несколькими вычислителями.
Программа должна считать *m, n*, матрицу A, столбец b, количество вычислителей; распараллелить решение уравнения $Ax = b$ методом Крамера.
Предполагаемый алгоритм решения. Методом Крамера каждый элемент столбца-результата вычисляется независимо от остальных. Поэтому его можно передать для расчета отдельному вычислителю.

11. Решение дифференциального уравнения $y' + y = f(x)$ методами вычислительной математики

Исходные данные: функция $f(x)$ (правая часть), отрезок, на котором ищется решение, количество вычислителей.
Цель: написать программу для расчета y решения уравнения $y' + y = f(x)$ параллельно несколькими вычислителями.

Программа должна считать правую часть, отрезок интегрирования; распараллелить процесс решения дифференциального уравнения $y' + y = f(x)$ заданным методом вычислительной математики.
Способ распараллеливания: стандартный, по отрезку. Отрезок делится на количество вычислителей, и каждому отдается своя часть.
Используемый метод вычислительной математики выбирается преподавателем.

Сортировка

12. Сортировка слиянием

Исходные данные: массив объектов, функция для их сравнения, количество вычислителей.
Цель: написать программу для упорядочивания массива параллельно несколькими вычислителями.
Предполагаемый алгоритм распараллеливания: при сортировке слиянием массив делится на две равных по размеру части, которые сортируются отдельно и сливаются в один. Таким образом, снова получаем дерево рекурсии. Примеры распараллеливания по дереву см. в задаче 1 раздела *Задачи с использованием деревьев*.

Вывод на экран

13. Контроль за выводом на экран нескольких потоков

Исходные данные: нет.
Цель: добиться с помощью средств синхронизации, чтобы несколько вычислителей выводили информацию на экран в строго определенном порядке.
Программа должна создать 26 потоков, каждый из которых выводит свою букву алфавита в бесконечном цикле. Добиться, чтобы все буквы выводились в алфавитном порядке.

Работа с файлами

14. Копирование файла в несколько потоков

Исходные данные: файл, директория его будущего расположения, количество вычислителей.
Цель: написать программу для копирования файла в другое место параллельно несколькими вычислителями.
Предполагаемый алгоритм распараллеливания:
а) Каждый вычислитель открывает файл отдельно. Они делят файл на равные части, каждый копирует свою часть (читая из копируемого файла и записывая в файл результата блоки данных фиксированного размера).

b) (Для нитей одного процесса.) Файл открывается один раз. Каждый вычислитель также должен скопировать свою часть файла. Но, поскольку файловый указатель общий, доступ к нему должен быть синхронизован. После использования указателя (т.е. копирования очередного блока данных из своей части) вычислитель должен вернуть его в первоначальное положение и снова включиться в борьбу за общий ресурс.

15. Копирование директории вместе со всем ее содержимым (параллелизм по файлам)

Исходные данные: директория для копирования, директория ее будущего расположения.

Цель: написать программу для копирования директории вместе со всем ее содержимым в другое место параллельно несколькими вычислителями.

Предполагаемый алгоритм распараллеливания: снова для операции копирования можно составить дерево рекурсии => распараллеливание по дереву (задача 1 раздела *Задачи с деревьями*).

Моделирование

В данном разделе студентам предлагается смоделировать следующие ситуации:

16. «Тренировка футболистов»

Участвующие вычислители: футболисты.

Общий ресурс: мяч.

Ситуация: команда футболистов отрабатывает индивидуальные действия. Каждый играет за себя. Цель футболиста – завладеть мячом и забить гол. Таким образом, он может выполнять два действия:

a) бороться за мяч.

b) бить по воротам, как только мяч оказался у него.

После удара по воротам вратарь выбивает мяч в поле. Футболист забивает гол с определенной вероятностью. Команда играет определенное время, потом определяется победитель – по забитым мячам.

17. «Семейный автомобиль»

Участвующие вычислители: члены семьи.

Общий ресурс: автомобиль.

Ситуация: семья из 4-х (N) человек: отец, мать, сын и дочь – имеет в распоряжении автомобиль. У каждого из них периодически появляется желание/необходимость использовать его. Пользование длится случайное время. По истечении определенного времени (известного заранее) бензин автомобиля заканчивается, и тот, кто в этот момент пользуется автомобилем, едет его заправлять. Кроме того, сын ответственен за мытье автомо-

биля. Также по истечению известного заранее времени автомобиль загрязняется. Если сын начинает пользоваться грязным автомобилем, он его моет перед этим.
Итого, все члены семьи «умеют» ездить на автомобиле, заправлять его, ожидать его освобождения. Сын помимо прочего «умеет» мыть автомобиль.

18. «Аэропорт»

Участвующие вычислители: самолет.
Общий ресурс: взлетно-посадочные полосы.
Ситуация: в течение каждого часа в аэропорту появляется 5 самолетов, готовых к вылету, и 5 самолетов, готовых приземлиться (каждый из них в случайное время в пределах этого часа). Чтобы взлететь или приземлиться самолету, требуется 3 минуты. В аэропорту 2 взлетно-посадочные полосы.

19. «Воробьи»

Участвующие вычислители: воробей.
Общий ресурс: хлебные крошки.
Ситуация: бабушка на скамейке периодически (пусть раз в Q минут) бросает воробьям по одной ровно N хлебных крошек. Когда бросают крошку хлеба, первый подлетевший к ней воробей хватает крошку, относит ее в сторону и съедает. За счет этого он пропускает «розыгрыш» следующей крошки. Первоначально у скамейки находится M воробьев. Каждую минуту к стае прилетает еще один воробей. Съев P крошек, воробей наедается и улетает. Чтобы количество воробьев не увеличивалось, должно выполняться соотношение $Q = N/P$.
Итак, воробей может выполнять действия:

- Прилететь (рождение процесса).
- Ждать раздачи крошек.
- Драться за крошку.
- Отлететь с крошкой в сторону и съесть ее.
- Улететь (смерть процесса).

20. «Раздача карт»

Участвующие вычислители: игрок.
Общий ресурс: колода карт.
Ситуация: N игроков тянут карты по одной из лежащей перед ними колоды.
Необходимо обеспечить:

- Очередность доступа к колоде.
- Равномерное распределение карт в колоде.

Можно ради интереса после раздачи посчитать количество очков в вытянутых ими картах и определить победителя.

Необходимое оборудование для лекций и практических занятий

Ноутбуки и проекторы.

Необходимое программное обеспечение

Системы программирования на языках C++, Java, C#.

Обеспечение самостоятельной работы

Доступ студентов в локальную сеть института и в Интернет.

Литература

Основная

1. *Буч Г.* Объектно-ориентированный анализ и проектирование. – 2-е изд. – М.: БИНОМ, 1998.
2. *Страуструп Б.* Язык программирования C++. – 3-е изд. – М.: БИНОМ, 1999.
3. *Секунов Н.* Visual C++.NET. – СПб.: БХВ-Петербург, 2002.
4. *Топп У., Форд У.* Структуры данных в C++. – М.: Изд. дом «Вильямс», 2000.
5. *Майерс С.* Эффективное использование STL. – СПб.: ПИТЕР, 2002.
6. *Александреску А.* Современное проектирование на C++. Обобщенное программирование и прикладные шаблоны проектирования. – М.: Изд. дом «Вильямс», 2002.
7. *Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.* Приемы объектно-ориентированного проектирования. Паттерны проектирования. – СПб.: ПИТЕР, 2003.
8. *Влиссидес Дж.* Применение паттернов проектирования. Дополнительные штрихи. – М.: Изд. дом «Вильямс», 2003.

Дополнительная

1. *Руссинович М., Соломон Д.* Внутреннее устройство Microsoft Windows: Windows Server 2003, Windows XP, Windows 2000. Мастер-класс. Серия: Мастер-класс. – СПб.: Питер, Русская Редакция, 2005.
2. *Лебланк Д., Ховард М.* Защищенный код (Writing Secure Code). Второе издание. – Издательство: Русская редакция. Серия: Фундаментальные знания, 2005.