Документ подписан простой электронной подписью
Информация о владельце:
ФИО: Тарасова Ирина Владимировна
Должность: Проректор по учебной работе
Дата подписания: 25.03.2022 16:38:16
Уникальный программный ключ:
8c45e14bf77dac42d4f8b124280a05e6949a00d3

ОБРАЗОВАТЕЛЬНОЕ ЧАСТНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
ПРАВОСЛАВНЫЙ СВЯТО-ТИХОНОВСКИЙ ГУМАНИТАРНЫЙ УНИВЕРСИТЕТ
(ПСТГУ)

Факультет информатики и прикладной математики
Кафедра информатики

ФОНД ОЦЕНОЧНЫХ СРЕДСТВ ДЛЯ ТЕКУЩЕГО КОНТРОЛЯ УСПЕВАЕМОСТИ ПО ДИСЦИПЛИНЕ

«Программирование»

02.03.03 «Математическое обеспечение и администрирование информационных систем»

Профиль:
Администрирование информационных систем

Квалификация выпускника: бакалавр

Форма обучения: очная

Москва, 2019 г.

Год начала обучения по учебному плану: 2019

Фонд оценочных средств для текущего контроля успеваемости разработан на основе рабочей программы дисциплины «Программирование», входящей в состав образовательной программы 02.03.03 «Математическое обеспечение и администрирование информационных систем».

Проверка результатов обучения студентов и сформированности их компетенций производится посредством следующих лабораторных работ и тестов.

За выполнение всех заданий начисляются баллы, которые учитываются при промежуточной аттестации по дисциплине (зачет в конце 1-го семестра, экзамены в конце 2 и 3-го семестров).

1 семестр

Всего за работу в семестре начисляется до 80 баллов

Лабораторные работы 1 , 2 и 3 – до 20 баллов за каждую

Тест1 – до 20 баллов

2 семестр

Всего за работу в семестре начисляется до 60 баллов

Лабораторные работы 4 и 5 – до 20 баллов за каждую

Тесты 2 и 3 – до 10 баллов за каждый

3 семестр

Всего за работу в семестре начисляется до 60 баллов

Лабораторные работы 6 и 7 – до 20 баллов за каждую

Тест 4 – до 20 баллов

Лабораторная работа №1.

Разработать программу, приветствующую пользователя.

После запуска программа выводит в консоль текст «Input your name», далее ожидает ввода строки пользователем. После ввода она выводит на экран фразу «Hello, {username}!» и завершает работу. Под «{username}» подразумевается текст, введённый до этого пользователем.

Лабораторная работа №2.

Разработать простой калькулятор.

После запуска программа предлагает выбрать операцию: 1 — сложение, 2 — вычитание, 3 — умножение, 4 — деление. После выбора операции программа предлагает

ввести 2 числа, далее выводит результат применения операции к этим числам и завершает работу.

Лабораторная работа №3.

Разработать программу для игры в «угадай число» с компьютером. Суть игры состоит в том, что одна из сторон загадывает целое число в заданном диапазоне, другая же старается угадать его за наименьшее количество попыток. Загадывающий на каждую попытку отвечает «больше загаданного», «меньше загаданного» и «угадал» в зависимости от соотношения загаданного и предложенного отгадывающим числа.

При запуске программа спрашивает, какую роль она исполняет – загадывающего или отгадывающего.

В режиме «загадывающий» программа предлагает ввести нижнюю и верхнюю границу диапазона для загадывания числа. Далее программа генерирует случайное число из этого диапазона и предлагает его отгадать. Каждую попытку пользователя отгадать число программа оценивает на точность (выводит «больше загаданного», «меньше загаданного» и «угадал»). Процесс длится до тех пор, пока пользователь не угадает число, либо пока он не сдастся, что также необходимо предусмотреть.

В режиме «угадывающий» программа просит ввести нижнюю и верхнюю границу диапазона, в котором пользователь загадает число. Далее программа предлагает число и просит пользователя ввести 1, если оно больше загаданного, -1 – если меньше и 0, если число угадано. Процесс длится до тех пор, пока пользователь не введёт 0, либо пока программа не решит, что число угадано (читай далее).

Необходимо разработать алгоритм угадывания и оценить количество попыток, достаточных для угадывания числа в заданном диапазоне. На основе этой оценки необходимо реализовать защиту от нечестных действий пользователя, когда загаданное число уже известно программе, но пользователь намеренно не принимает его как правильное.

После завершения игры выводится загаданное число, а также затраченное количество попыток.

Лабораторная работа №4.

Разработать программу для демонстрации работы алгоритма сортировки массивов (любого на выбор).

В начале работы программа запрашивает размер массива, далее предлагает заполнить его вручную, либо автоматически – в этом случае задаётся диапазон для генерации случайных чисел. Массив выводится на экран.

Далее программа запрашивает порядок сортировки – по возрастанию или убыванию, после чего происходит сортировка и отсортированный массив выводится на экран.

Лабораторная работа №5.

Реализовать программу для заполнения двумерного массива размера M на N числами от 1 до M*N по спирали по часовой стрелке, начиная от верхнего левого угла.

В начале работы программа запрашивает числа M и N, далее заполняет массив и выводит его на экран.

Создавать двумерный массив следует динамически (т.к. M и N заранее неизвестны), а в конце работы программы следует освободить занятую им память.

Лабораторная работа №6.

Реализовать простую записную книжку. Количество записей – не более 500. Для каждой записи хранится следующая информация:

- ФИО
- Номер телефона
- E-mail
- Комментарий

Работа со строками производится посредством C-строк, использование класса std::string стандартной библиотеки не допускается.

Первые три поля можно хранить как массив char заданного размера (например, char[100]), последнее поле (Комментарий) не следует заранее ограничивать в размерах, посему следует использовать указатель char* и вручную управлять выделением и освобождением памяти для этого поля. Для всех полей необходимо предусмотреть проверку, чтобы не допустить выхода за границы массива при вводе данных.

Управление записной книжкой происходит посредством специальных команд. Общий вид команды: «*cmd data*», где cmd – команда, далее следует пробел, а далее data – данные, передаваемые обработчику команды. Программа должна поддерживать следующие команды:

- «+ ФИО» – добавить новую запись с указанным ФИО:
  - Если такое ФИО уже существует в записной книжке, выдать ошибку;
  - Иначе попросить ввести остальные поля и сохранить запись в книжке.

- «- ФИО» – удалить запись из записной книжки:
    - Если такого ФИО нет, выдать ошибку;
    - Иначе удалить запись из книжки.
- «! ФИО» – изменить запись с указанным ФИО:
    - Если такого ФИО нет, выдать ошибку;
    - Иначе попросить ввести новое ФИО:
        - Если новое ФИО уже существует, попросить ввести другое;
    - Попросить ввести остальные поля и сохранить изменения.
- «? подстрока» – найти все записи, в поле ФИО которых содержится указанная подстрока:
    - Если найдено более одной записи – вывести список ФИО найденных;
    - Если найдена одна запись, вывести все её поля.
- «exit» - завершить работу программы.

При запуске программа загружает данные из файла abook.txt (при его наличии), принимает и выполняет команды до тех пор, пока не будет введена команда «exit», после чего данные сохраняются в abook.txt и программа завершает выполнение. Опционально можно предусмотреть сохранение данных в файл и при завершении программы посредством закрытия окна консоли.

Лабораторная работа №7.

Необходимо реализовать простую поисковую систему, обеспечивающую эффективный поиск посредством построения поискового индекса.

Программа позволяет осуществлять поиск по произвольному количеству текстовых файлов. В качестве поискового запроса выступает набор слов, разделённых пробелом. Результатом поиска является набор абзацев, в каждом из которых встречаются все перечисленные в запросе слова.

Поиск должен быть реализован эффективно (т. е. работать быстро), поэтому решение, основанное на просмотре всех текстовых файлов на предмет наличия в них подходящих абзацев при выполнении каждого запроса не годится, т. к. будет работать слишком медленно.

Для решения этой проблемы следует заранее построить структуру данных, реализующую эффективный поиск — поисковый индекс. Это словарь, в котором каждому уникальному слову соответствует множество абзацев, в которых оно встречается. Имея такую структуру данных, легко реализовать выполнение описанных выше поисковых запросов: для каждого слова из запроса получаем множество абзацев, в которые входит это

слово. Пересечение всех этих множеств и будет множеством абзацев, в которых встречаются все искомые слова.

Использование поискового индекса позволяет реализовать быстрое выполнение поисковых запросов, но имеет свою цену:

- поисковый индекс требует дополнительную память;
- построение поискового индекса — операция вычислительно затратная.

В связи с этим поисковый индекс нужно строить заранее (до выполнения поисковых запросов), также нужно реализовать его хранение между запусками программы на диске в файле «searchIndex».

Таким образом, программа при старте загружает поисковый индекс из файла «searchIndex» (при его наличии), в противном случае инициализируется пустой поисковый индекс. После старта в цикле предлагается ввести одну из следующих команд:

- + filename.txt — добавить в поисковый индекс текстовый файл по пути filename.txt
  - для добавления в поисковый индекс текст разбивается на абзацы (последовательность символов до символа перехода на следующую строку «\n»). Каждый абзац разбивается на слова (всё кроме букв и цифр отбрасывается, полученная строка разбивается по пробелам). Далее для каждого слова из абзаца проверяется, нет ли его в поисковом индексе (ключ словаря). Если есть — в уже имеющееся множество абзацев добавляется текущий. Если нет — создаётся новый ключ, значениям для которого будет множество, состоящее из текущего абзаца.
- слово1 слово2 слово3 — выполнение поискового запроса. В результате на экран выводятся все подходящие абзацы.
- ! exit — сохранение поискового индекса в файл «searchIndex» и выход из программы.

При выполнении лабораторной работы следует использовать возможности библиотеки STL:

- словарь может быть реализован при помощи класса std::map
- множество может быть реализовано как std::set
- пересечение множеств посчитать при помощи шаблонной функции std::set_intersection.

Также рекомендуется хранить в поисковом индексе не сами абзацы, а их номера. Сами же абзацы можно хранить в отдельном std::vector — тем самым номером абзаца будет его индекс в этом массиве. Это облегчит сохранение поискового индекса в файл, а также выполнение дополнительного задания к данной ЛР (см. ниже).

Сохранение поискового индекса в файл сводится к сохранению собственно поискового индекса (map) в файл, плюс сохранение массива (vector) абзацев. Разработать формат хранения предлагается самостоятельно.

При выполнении работы нельзя пользоваться какими-либо СУБД (например, встраиваемой СУБД SQLite) и любыми иными способами хранения структурированной информации на диске (B-деревья и проч.), если только этот код не реализован студентом самостоятельно.

В качестве текста, используемого для демонстрации работы программы, предлагается использовать текст романа «Война и мир» Л.Н. Толстого, дополненный текстом любого другого достаточно большого произведения классической литературы.

Дополнительно задание 1.

Доработайте программу так, чтобы поисковый индекс «помнил» текст, из которого взят абзац. Для этого в массиве абзацев надо хранить не сами абзацы, а пары (std::pair) <имя_текста, абзац>. Имя текста задаётся при индексации текстового файла следующим образом:

- Пользователю после запуска команды «+ filename.txt» предлагается ввести имя текста и нажать Enter.
- Если пользователь сразу нажал Enter (т. е. имя текста оказалось пустым, либо состоящим из пробелов) — в качестве имени текста следует взять имя файла filename.

Здесь с целью оптимизации сохранения поискового индекса в файл можно также хранить имена текстов в отдельном std::vector, а пары <имя_текста, абзац> заменить на <индекс_имени_текста, абзац>.

Дополнительное задание 2.

Некоторые слова несут мало информации, но встречаются слишком часто — это предлоги, местоимения, частицы и проч. Для таких слов (они называются *стоп*-словами) в поисковом индексе будут строиться очень длинные множества абзацев, что приводит к неэффективному потреблению ресурсов — как памяти на хранение, так и вычислительных при расчёте пересечения с такими множествами.

Общепринятой практикой является отбрасывание таких слов при построении поискового индекса и обработке запроса.

Реализуйте эту модификацию в Вашей поисковой системе. Список стоп слов можно найти в интернете — например взять его из исходных кодов библиотеки stop-words для

языка программирования Python (https://github.com/Alir3z4/stop-words/blob/0e438af98a88812ccc245cf31f93644709e70370/russian.txt).

Дополнительное задание 3.

Разработанная поисковая система в силу своей простоты обладает некоторым недостатком: она не видит ничего общего между разными формами одного слова, т. е. слова «кот», «кота», «коту» для неё совершенно разные слова. Поэтому если в абзаце и запросе слова находятся в разных формах, то абзац не будет удовлетворять запросу, что ограничивает полноту поиска. Хотелось бы при поиске не учитывать формы слов.

В качестве простого решения предлагается при построении индекса и при обработке поискового запроса приводить все слова к некоторой начальной форме. Под начальной формой могут подразумеваться разные вещи:

- каноническая форма слова (ед.ч, И.п), например «котами → кот». Такая процедура называется «*лемматизация*».
- неизменяемая часть снова (основа), которая сама по себе словом может и не являться: «самолетный → самолетн». Такая процедура называется «*стемминг*».

К преимуществам алгоритмов *стемминга* относится их компактность, простота и быстрота работы. Но они далеко не всегда точны, особенно для русского языка. В то же время алгоритмы *лемматизации* требуют более глубоких знаний о языке, могут использовать большие словари, более точны, но сравнительно медленны.

Предлагается воспользоваться стеммером Портера, адаптированным для русского языка. Исходный код на C++ можно взять, например, здесь: http://manunich.blogspot.ru/2016/01/c-cpp.html — это результат переписывания реализации для языка Java, которую можно найти здесь: http://www.algorithmist.ru/2010/12/porter-stemmer-russian.html.

Отметим, что отказ от учёта формы слов вкупе с особенностями алгоритмов стемминга может приводить к тому, что близкие по написанию но разные по смыслу слова сольются для системы поиска в одно целое. В результате в ответ на запрос пользователя будут возвращены казалось бы совершенно неподходящие абзацы. Эта проблема особенно ярко проявляется, когда мы не учитываем регистр символов («слова с большой буквы»): «Пушкин» может стать синонимом «пушка», «Безухов» - «безухий», «Толстой» - «толстый».

Фундаментальным недостатком данной поисковой системы является отбрасывание информации о всяких взаимоотношениях между словами, что может приводить к неожиданным результатам: на запрос «Горький о трудной жизни» можно получить ответ наподобие «студент Сидоров писал о счастливой жизни здешних аборигенов, которая

лишена трудных математических и вообще каких-либо иных задач, а вся горечь её заключалась разве что в том, что их праздничной пищей по какой-то непонятной традиции были горькие плоды с кустарников, растущих в середине острова».

Несмотря на приведённые недостатки, подобный подход вполне применим на практике и очень часто используется для добавления возможности полнотекстового поиска в системы, основной задачей которых является отнюдь не «идеальный» поиск. Такого рода механизмы включены в популярные СУБД (MS SQL Server, PostgreSQL и т. д.). Другое дело, если поиск — основная задача системы, тогда используются гораздо более интересные подходы, но это уже гораздо более сложная задача, для которой нет простого универсального решения, реализованного в виде готовой системы.

Тест 1.

Правильные варианты ответов помечены знаком (+)

1) Как называется набор инструкций, описывающих порядок действий исполнителя для достижения некоторого результата:

- рецепт
- руководство
- программа
- (+) алгоритм

2) Что из перечисленного не является языком программирования

- Assembler
- (+) Paris
- C++
- Pascal

3) Какие существуют базовые виды алгоритмов (выберите несколько правильных ответов)

- (+) линейный
- (+) циклический
- (+) разветвляющийся
- сходящийся

4) Массовость (одно из свойств алгоритма) означает:

- (+) способность алгоритма решать поставленную задачу при разных исходных данных
- алгоритм требует наличия большого количества исполнителей
- алгоритм способен обрабатывать большое количество входных данных за раз

- алгоритм относится к числу общеизвестных

5) Что из перечисленного не является формой записи алгоритма

- псевдокод
- блок-схема
- (+) набросок
- запись на формальном языке (языке программирования)

6) Какие существуют типы циклов в языке программирование Pascal?

- (+) со счётчиком
- бесконечный
- (+) с предусловием
- (+) с постусловием

7) В какой форме должен быть представлен алгоритм, чтобы компьютер смог его непосредственно выполнить?

- текст на языке программирования
- текст на языке ассемблера
- (+) машинные инструкции
- в любой

8) Оператор в контексте языков программирования - это

- (+) наименьшая автономная часть языка программирования
- разработчик программы
- пользователь
- исполнитель программы

9) Что из перечисленного описывает разновидность оператора

- (+) условный
- (+) цикла
- расширенный
- (+) составной

10) Какое из перечисленных утверждений про переменные верно?

- (+) переменные нужны для хранения входных, промежуточных и выходных данных алгоритма
- количество переменных напрямую влияет на выполнение свойства массовости алгоритма
- переменные нужны для тонкой настройки работы алгоритма до начала его выполнения

- переменные нельзя использовать, т.к. они нарушают свойство детерменированности алгоритма

11) Какой оператор C++ не относится к числу арифметических ?

- +
- -
- *
- (+) &&

12) Что из перечисленного не является стандартным типом данных языка C++

- int
- float
- (+) anytype
- char

Тест 2

1) Какие существуют способы перевода программы с языка программирования в машинные инструкции

- (+) компиляция
- пошаговое выполнение
- (+) интерпретация
- интерактивный

2) Что из перечисленного не является оператором C++?

- ++
- --
- %
- (+) ??

3) Что из перечисленного необходимо для объявления переменной в Pascal?

- (+) имя переменной
- начальное значение
- (+) тип данных
- диапазон допустимых значений

4) При помощи каких ключевых слов описываются структуры в языках Паскаль и C++

- array

- (+) struct
- int[]
- (+) record

5) Найдите пример объявления массива на языке C++

- a: array of integer [1..10];
- int[10] array b;
- int[1..10] a;
- (+) double nmb[10];

6) Найдите логический оператор в приведённом списке

- +
- /
- (+) ==
- $

7) Что из перечисленного относится к составным типам данных

- (+) массив
- указатель
- (+) структура
- ссылка

8) Какое утверждение про массивы не верно?

- быстрый доступ к элементу
- неэффективность операции вставки/удаления элемента
- (+) элементы могут быть разных типов
- элементы располагаются в памяти последовательно

9) Какое утверждение про структуры не верно?

- быстрый доступ к полю
- каждое поле имеет уникальное имя
- (+) все поля должны быть одного типа
- список полей задаётся на этапе разработки программы

10) Какие способы передачи параметров в функцию/процедуру существуют?

- косвенный
- (+) по значению
- прямой
- (+) по ссылке

11) Как выглядит оператор присваивания в C++?

- ==
- :=
- <=
- (+) =

12) Вы разрабатываете электронную записную книжку. Выберите наилучшие архитектурные решения.

- Необходимо завести отдельные массивы для каждого поля (ФИО, телефон) и синхронизировать количество элементов между ними, чтобы не возникло путаницы.
- (+) Сведения об одном человеке следует хранить в структуре с полями ФИО, телефон и т. д.
- Для каждого человека нужно создавать динамический массив. По чётным индексам будут лежать имена полей, а по нечётным — их значения. Это позволит обеспечить гибкость структуры записной книжки, простоту поиска и лёгкость реализации.
- (+) Все записи о людях можно хранить в массиве. Но тогда неэффективна операция удаления записи. Следует поискать более подходящую структуру данных для хранения всей записной книжки — например, воспользоваться списками.

13) Что из перечисленного не является разновидностью алгоритма сортировки массивов

- сортировка вставками
- пирамидальная сортировка
- быстрая сортировка
- (+) эффективная сортировка

14) Каким образом в C++ можно задать функцию, меняющую местами значения двух переданных ей целочисленных переменных

- void swap(int a, int b)
- void swap(int &a, int b)
- void swap(int a, int **b)
- (+) void swap(int* a, int* b)

15) Какие утверждения о механизмах динамического выделения памяти C++ не верны?

- память может быть выделена в процессе выполнения программы
- необходима для реализации динамических структур данных (списки, деревья и проч.)
- (+) среда выполнения автоматически освобождает неиспользуемую программой динамическую память

- доступ к динамической памяти производится посредством указателя

Тест 3.

1) Структурное программирование поощряет использование следующих конструкций языка программирования:

- (+) условный оператор
- (+) оператор цикла
- оператор присваивания
- оператор безусловного перехода (go to)

2) Какие утверждения соответствуют парадигме процедурного программирования

- (+) программа представляет из себя последовательность шагов
- (+) каждый шаг может состоять из подшагов
- каждый шаг программы - простая неделимая операция
- (+) составной шаг в программе представляет из себя процедуру/функцию

3) Модульное программирование подразумевает:

- вынос в отдельные файлы необязательных частей программы
- запрет на использование отрицательных значений числовых переменных
- (+) организацию программы в виде набора законченных независимых компонент (модулей), взаимодействие между которыми подчиняется строго определённым правилам

4) Как соотносятся парадигмы структурного и процедурного программирования?

- Эти подходы вообще никак не связаны.
- Структурное программирование основывается на парадигме процедурного программирования, т. к. это по сути иерархическое упорядочение процедур, из которых состоит программа.
- (+) Структурное программирование запрещает использовать оператор безусловного перехода (go to) для организации ветвлений и циклов. Однако go to также может использоваться для выделения набора инструкций, решающих определённую подзадачу. Этот сценарий использования оператора go to покрывается парадигмой процедурного программирования.
- Процедурное программирование не существует без структурного программирования.

5) Как соотносятся парадигмы модульного и процедурного программирования

- Это независимые подходы
- (+) Модульное программирование не существует без процедурного, т. к. оно подразумевает объединение процедур в модули.
- Модульное программирование — это подход к решению вычислительных задач, в рамках которого запрещено использование отрицательных чисел.
- Это одно и то же.

6) В объектно-ориентированной парадигме программирования используются следующие понятия:

- (+) класс
- (+) объект
- (+) наследование
- естественный отбор

7) Сколько принципов объектно-ориентированного программирования принято выделять?

- (+) 3
- 2
- 6
- в рамках ООП нет общепринятого набора принципов.

8) Какая структура данных наиболее близка к понятию класса?

- массив
- файл
- заголовочный файл
- (+) структура

9) Какой из принципов является общим для ООП и модульного программирования?

- полиморфизм
- (+) инкапсуляция
- наследование
- все перечисленные принципы

10) Выберите верные утверждения про принципы ООП

- (+) Принцип инкапсуляции уже присутствует в модульном программировании
- (+) Принцип наследования сам по себе не вносит ничего кардинально нового, т. к. это просто способ избежать копирования кода. Но при соединении с принципом полиморфизма получается гибкий инструмент разработки.

- (+) Полиморфизм приводит к тому, что разработчик не знает, какие именно операции будут выполнены при исполнении написанного им кода. Это связано с тем, что вызываемые им методы могут быть переопределены в классе-наследнике.
- Все предыдущие утверждения ложны.

11) Какое понятие наиболее близко к понятию метода класса?

- Условный оператор
- (+) Процедура
- Точка останова
- Модуль

12) Перечислите принципы ООП.

- (+) Инкапсуляция
- (+) Наследование
- Изменчивость
- (+) Полиморфизм

Тест 4.

1) Каким образом можно реализовать вычисление N-го числа Фибоначчи?

- Это алгоритмически неразрешимая задача
- (+) Рекурсивной процедурой
- (+) Циклом
- Таблицей из первый 10 значений

2) В чём главный недостаток рекурсивного алгоритма вычисления N-го числа Фибоначчи?

- Это самый эффективный алгоритм, т. к. он наиболее близок к математическому определению.
- Такого алгоритма не существует.
- (+) При рекурсивной реализации много раз считаются одни и те же числа Фибоначчи, что вычислительно неэффективно
- Данный алгоритм требует слишком много памяти

3) Возможно ли реализовать рекурсивный алгоритм, не используя в явном виде рекурсию?

- Нет
- Да, любой рекурсивный алгоритм может быть легко реализован при помощи цикла.
- Да, т. к. рекурсивный алгоритм это частный случай циклического.

- (+) Да, любой рекурсивный алгоритм может быть реализован при помощи цикла, но во многих случаях это непростая задача.

4) Вам известен рекурсивный алгоритм решения задачи, но он вычислительно неэффективен, т. к. много раз вычисляет одно и то же в процессе работы. Циклический алгоритм слишком сложен для реализации и не рассматривается. Что следует предпринять, чтобы повысить эффективность программы?

- Ничего нельзя сделать, необходимо запускать программу на более мощном ПК.
- (+) Необходимо сохранять результат работы рекурсивной функции для каждого набора входных параметров. В дальнейшем при вызове функции с этими же параметрами достаточно выдать сохранённое значение.
- Рекурсивные алгоритмы очень хорошо поддаются распараллеливанию, поэтому следует модифицировать программу так, чтобы она могла использовать несколько процессорных ядер.

5) Для решения задачи разработан рекурсивный алгоритм, но он вычислительно неэффективен (много раз считается одно и то же). Для решения этой проблемы программист реализовал построение таблицы уже вычисленных значений, которая позволяет по параметрам рекурсивной процедуры получить готовый ответ, если он уже вычислялся. Какие проблемы могут возникнуть в дальнейшем?

- Данная таблица бесполезна, т. к. важным свойством рекурсивных алгоритмов является уникальность используемых в рекурсивной процедуре параметров.
- (+) Таблица может стать слишком большой, что скажется на потреблении памяти.
- (+) Таблица может стать слишком большой. Поиск готового ответа в ней может стать сравнимым с повторным вычислением.

6) С целью оптимизации вычислительной эффективности рекурсивного алгоритма можно строить таблицу уже вычисленных значений. Однако она может стать слишком большой, что отрицательно скажется как на потреблении памяти, так и на вычислительной эффективности (долго искать готовый ответ в таблице). Что в этом случае можно предпринять?

- Ничего нельзя сделать, следует использовать более мощный ПК
- (+) Можно ограничить размер таблицы. При нехватке места можно удалять записи, к которым реже всего обращались.
- Можно хранить таблицу в файле — это решить проблему потребления памяти, т. к. вместительность жёсткого диска гораздо больше, чем оперативной памяти. Хранение

таблицы на жёстком диске слабо влияет на вычислительную эффективность программы.

Критерии оценивания заданий текущего контроля.

При оценке лабораторных работ оцениваются следующие параметры программы:

- Работоспособность программы, удобство пользования.
- Правильное оформление кода (отступы, пробелы в выражениях и т.д.).
- Осмысленные названия процедур, функций, переменных, наличие комментариев к основным элементам программы, а также сложным местам.
- Использование функций и процедур для избегания дублирования кода:
  - Функция/процедура должна отвечать за решение одной вполне определённой подзадачи.
  - Функция/процедура должна быть как можно меньше зависима от внешнего контекста (глобальных переменных) – необходимые данные следует передавать как аргументы функции/процедуры.
- Разбиение программы на модули в случае необходимости.
- Программа должна проверять корректность входных данных.
- Следует рационально использовать ресурсы компьютера (память, процессор).

Студент демонстрирует работоспособность программы, после чего предоставляет исходный код. Студент должен ответить на все возникающие в процессе проверки вопросы, в частности, должен уметь разъяснить структуру всего исходного кода и назначение отдельных его частей, а также суть используемых алгоритмов.

Критерии оценивания лабораторных работ

Шкала оценки		**Критерии оценки**
Оценка	**Баллы**	
17-20 (отлично)	17-20	Обучающийся: • полностью реализовал требуемый функционал • смог объяснить принцип работы программы • программа работает без ошибок • программный код организован грамотно и содержит поясняющие комментарии • программа работает без ошибок на произвольном наборе входных данных

13-16 (хорошо)	13-16	Обучающийся: • полностью реализовал требуемый функционал • смог объяснить принцип работы программы • но, программа работает с небольшими ошибками • программный код организован грамотно, но не содержит поясняющие комментарии • программа работает с небольшими ошибками ошибок на произвольном наборе входных данных
9-12 (удовлетворительно)	9-12	Обучающийся: • не полностью реализовал требуемый функционал • не смог четко и грамотно объяснить принцип работы программы • программа работает с ошибками • программный код плохо читаем, не содержит поясняющие комментарии • программа работает с ошибками ошибок и не на произвольном наборе входных данных
(неудовлетворительно)	0 (баллы не начисляются)	Обучающийся: • не реализовал требуемый функционал • не смог четко и грамотно объяснить принцип работы программы • программа работает с грубыми ошибками • программный код плохо читаем, не содержит поясняющие комментарии • программа работает только на одном наборе входных данных

При оценке тестов вычисляется число правильных ответов. За каждый правильный ответ начисляется 1 первичный балл. Если вопрос подразумевает выбор нескольких ответов и студент выбрал только правильные ответы, но не все, ему начисляется 0,5 балла. Далее баллы суммируются и делятся на общее число вопросов. Оценка за тест считается неудовлетворительной и баллы не начисляются, если это число меньше 0,4. Если полученное число не меньше 0,4, то оно умножается на максимальное число баллов, назначенное за тест, округляется до целых обычным способом и полученное число баллов начисляется студенту.

Зачёт в 1 семестре студент получает автоматически при выполнении (с оценкой не ниже «удовлетворительно») всех лабораторных работ и прохождении тестов с результатом не ниже 0,4.

Для допуска к зачету должно быть сдано не менее половины лабораторных работ.

Автор – Евсеев А.С.

Одобрено на заседании кафедры Информатики от «31» мая 2019 года, протокол №05-19