Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет информационных технологий и управления

Кафедра интеллектуальных информационных технологий

ТРАДИЦИОННЫЕ И ИНТЕЛЛЕКТУАЛЬНЫЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Рекомендовано УМО по образованию в области информатики и радиоэлектроники в качестве пособия для специальности 1-40 03 01 «Искусственный интеллект»

Минск БГУИР 2016

Библиотека БГУИР

УДК 004.8(076.5)
ББК 32.813я73
Т65

А в т о р ы:
В. В. Голенков, Н. А. Гулякина, И. Т. Давыденко, Д. В. Шункевич

Р е ц е н з е н т ы:
кафедра интеллектуальных систем
Белорусского государственного университета
(протокол №12 от 12.05.2015);

ведущий научный сотрудник государственного учреждения
«Объединенный институт проблем информатики
Национальной академии наук Беларуси»,
кандидат физико-математических наук, доцент Ю. В. Поттосин

Традиционные и интеллектуальные информационные технологии.
Т65 Лабораторный практикум : пособие / В. В. Голенков [и др.]. – Минск : БГУИР, 2016. – 64 с. : ил.
ISBN 978-985-543-193-1.

Сформулированы основные положения, касающиеся лабораторного практикума по дисциплине «Традиционные и интеллектуальные информационные технологии», даны рекомендации по выполнению лабораторных работ, рассмотрены примеры решения задач, поставленных в рамках лабораторных работ.

УДК 004.8(076.5)
ББК 32.813я73

ISBN 978-985-543-193-1

© УО «Белорусский государственный университет информатики и радиоэлектроники», 2016

Библиотека БГУИР

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 4

ЛАБОРАТОРНАЯ РАБОТА №1 РАБОТА В КОМАНДНОЙ СТРОКЕ. СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ 5
1.1 Теоретические сведения 5
1.2 Варианты индивидуальных заданий 18

ЛАБОРАТОРНАЯ РАБОТА №2 РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ 21
2.1 Теоретические сведения 21
2.2 Варианты индивидуальных заданий 27

ЛАБОРАТОРНАЯ РАБОТА №3 РАБОТА С ТЕКСТОВЫМ РЕДАКТОРОМ 28
3.1 Теоретические сведения 28
3.2 Варианты индивидуальных заданий 44

ЛАБОРАТОРНАЯ РАБОТА №4 РАЗРАБОТКА ФРАГМЕНТА ОНТОЛОГИИ ПРЕДМЕТНОЙ ОБЛАСТИ 45
4.1 Теоретические сведения 45
4.2 Варианты индивидуальных заданий 50

ЛАБОРАТОРНАЯ РАБОТА №5 СТРУКТУРЫ ДАННЫХ 52
5.1 Теоретические сведения 52
5.2 Варианты индивидуальных заданий 54

ЛАБОРАТОРНАЯ РАБОТА №6 ПРИМЕНЕНИЕ ТЕОРИИ ГРАФОВ 57
6.1 Теоретические сведения 57
6.2 Варианты индивидуальных заданий 58

ЛАБОРАТОРНАЯ РАБОТА №7 ОПЕРАЦИИ НАД МНОЖЕСТВАМИ 59
7.1 Теоретические сведения 59
7.2 Варианты индивидуальных заданий 62

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 64

Библиотека БГУИР

ВВЕДЕНИЕ

Целью преподавания дисциплины «Традиционные и интеллектуальные информационные технологии» является ознакомление первокурсника с широко используемыми пакетами прикладных программ, современными системами контроля версий, базовыми принципами функционирования интеллектуальных систем, привитие базовых навыков формализации знаний и программирования на специализированных языках обработки знаний.

Целью лабораторных занятий в первом семестре является закрепление теоретического курса, приобретение навыков работы с современными системами контроля версий и регулярными выражениями, формализация знаний выбранной предметной области.

На протяжении первого семестра студентам предлагается четыре лабораторные работы (1–4) на следующие темы:

- Работа в командной строке. Системы контроля версий.
- Регулярные выражения.
- Работа с текстовым редактором.
- Разработка фрагмента онтологии предметной области.

Целью лабораторных занятий во втором семестре является закрепление навыков обработки сложных структур данных, применения теории графов на практике.

На протяжении второго семестра студентам предлагается три лабораторные работы (5–7) на следующие темы:

- Структуры данных.
- Применение теории графов.
- Операции над множествами.

Библиотека БГУИР

ЛАБОРАТОРНАЯ РАБОТА №1 РАБОТА В КОМАНДНОЙ СТРОКЕ. СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ

Цель работы: изучить команды командной строки ОС Windows, ОС Linux; приобрести навыки работы в системе контроля версий GitBash.

1.1 Теоретические сведения

1.1.1 Работа в командной строке

В процессе выполнения первой части лабораторной работы студенту необходимо:

- освоить командные оболочки shell (для OS семейства Unix) и cmd (для OS семейства MS Windows); изучить основные встроенные команды; научиться писать файлы сценариев;
- научиться соотносить командные оболочки для разных OS; освоить командное окружение для OS семейства Unix (утилиты из пакета GNU Core Utilities) и соответствующие им утилиты для OS семейства MS Windows;
- ознакомиться с программами git и gitk, освоить их программные аргументы.

Указания к работе

Для работы необходимо использовать файлы сценариев git-bash.bat git-cmd.bat. Для получения помощи используйте следующие команды:

1) help в cmd.exe;
2) help, man, info в Linux и <имя команды> --help.

Необходимо на практике уметь выполнять всевозможные действия в командной оболочке и отвечать на теоретические вопросы. Обратить внимание:

1) на команды, выполняющие одинаковые действия в cmd.exe и bash;
2) использование нескольких команд и символов условной обработки;
3) вложенные командные оболочки;
4) на использование переменных среды в cmd.exe:
   a) перечень основных переменных;
   b) установка переменных среды;
   c) подстановка значений в переменные среды;
5) использование операторов перенаправления команд;
6) использование фильтров;
7) на перечень всевозможных команд командных оболочек и примеры их использования.

Библиотека БГУИР

Интерфейс командной строки – разновидность текстового интерфейса (CUI) между человеком и компьютером, в котором инструкции компьютеру даются в основном путем ввода с клавиатуры текстовых строк (команд), в UNIX-системах возможно применение мыши. Также известен под названием «консоль».

Интерфейс командной строки противопоставляется системам управления программой на основе меню, а также различным реализациям графического интерфейса.

Интерфейс командной строки применяется по следующим причинам:

- небольшой расход памяти по сравнению с системой меню;
- наличие большого числа команд, многие из которых нужны крайне редко. Поэтому даже в некоторых программах с графическим интерфейсом применяется командная строка, т. к. набор команды (при условии, что пользователь знает эту команду) осуществляется гораздо быстрее, чем, например, навигация по меню.

Естественное расширение интерфейса командной строки – пакетный интерфейс. Его суть в том, что в файл обычного текстового формата записывается последовательность команд, после чего этот файл можно выполнить в программе, что возымеет такой же (не меньший) эффект, как если бы эти команды были по очереди введены в командную строку. Примерами могут служить .bat-файлы в DOS и Windows, shell-скрипты в UNIX-системах.

Достоинства интерфейса командной строки:

- легкость автоматизации. Shell script в UNIX-подобных системах является полноценным интерпретируемым языком программирования и способен автоматизировать любую системную задачу. В Windows присутствует их примитивный аналог – пакетные файлы, и более мощный аналог – powershell. По сути, это – простейшая программируемость. С графическим интерфейсом без поддержки программой командной строки это сделать почти невозможно;
- возможность управления программами, не имеющими графического интерфейса (например, выделенным сервером);
- возможность вызова любой команды небольшим количеством нажатий;
- мгновенное и непосредственное обращение к командам для разных исполнимых файлов, тогда как в GUI приходится сначала запускать, а затем закрывать графический интерфейс для каждого исполнимого файла;
- просмотрев содержимое консоли, можно повторно увидеть промелькнувшее сообщение, которое вы не успели прочитать;

Библиотека БГУИР

- возможность использования удаленного компьютера с любого устройства, подключаемого к Интернету или локальной сети (ПК, субноутбук, КПК, сотовый телефон, портативная игровая консоль), без особых затрат трафика;
- отсутствие деталей интерфейса, таких как пусковые панели и рамки окон, что при равных разрешениях позволяет вместить значительно больше текста на страницу.

Недостатки интерфейса командной строки:

- не является «дружелюбным» для пользователей, которые начали знакомство с компьютером с графического режима;
- необходимость изучения синтаксиса команд и запоминания сокращений, осложняющаяся тем, что каждая команда может иметь свои собственные обозначения;
- отсутствие автодополнения, ввод длинных и содержащих спецсимволы параметров с клавиатуры может быть затруднительным;
- отсутствие «аналогового» ввода. Например, подбор громкости с помощью озвученного ползунка позволяет выставить подходящую громкость быстрее, чем командой вроде aumix -v 90.

Для обеспечения интерфейса командной строки в операционных системах часто используются командные интерпретаторы, которые могут представлять собой самостоятельные языки программирования, с собственным синтаксисом и отличительными функциональными возможностями.

В операционные системы MS-DOS и Windows 9x включен командный интерпретатор command.com, в Windows NT включен cmd.exe, начиная с Windows XP (пакет обновления 2) доступен PowerShell, который является встроенным компонентом ОС, начиная с Windows 7 и Windows 2008 Server.

В UNIX-подобных системах у пользователя есть возможность менять командный интерпретатор, используемый по умолчанию. Из командных оболочек UNIX наиболее популярны bash, csh, ksh, zsh.

Рассмотрим наиболее распространенные командные интерпретаторы.

COMMAND.COM

COMMAND.COM – интерпретатор командной строки в операционных системах DOS, OS/2, семейства Windows 9x и ряда других. Загружается при старте системы и выполняет команды из файла AUTOEXEC.BAT.

В операционных системах семейства Windows NT (начиная с Windows NT 3.1 и заканчивая Windows 8/Windows Server 2012) и OS/2 интерпретатором командной строки является программа cmd.exe. Однако для

Библиотека БГУИР

совместимости с DOS-приложениями COMMAND.COM присутствует и в версиях этих систем для процессоров архитектуры IA-32.

COMMAND.COM имеет два режима работы. Первый режим – интерактивный, когда пользователь вводит с клавиатуры команды, которые немедленно выполняются. Второй режим – пакетный, когда COMMAND.COM выполняет последовательность команд, заранее сохраненную в пакетном файле с расширением .BAT

Команды COMMAND.COM делятся на внутренние и внешние. Внутренние команды поддерживаются самим COMMAND.COM, внешние команды являются файлами, которые хранятся на дисках и имеют расширение .COM, .EXE или .BAT.

Пакетные файлы для COMMAND.COM имеют четыре типа переменных:

1 ERRORLEVEL содержит код возврата последней из запущенных программ (к примеру, в языке программирования Си код можно вернуть с помощью return в функции main). Как правило, ERRORLEVEL используется для индикации ошибок при работе программы и код 0 означает успешное завершение. Но это относится в основном к утилитам командной строки (которые ориентированы на использование в пакетных файлах), прикладные программы обычно не заботятся о возврате конкретных значений, поэтому после них в ERRORLEVEL всегда оказывается нулевое значение или даже мусор. В оригинальном COMMAND.COM код возврата можно было проверить только с помощью конструкции IF ERRORLEVEL, однако в некоторых клонах DOS, а также Windows семейства NT добавлена возможность обращения к ERRORLEVEL как к обычной переменной.

2 Переменные могут быть заданы с помощью команды SET. Чтобы получить их значение, нужно имя переменной окружить знаками % (например, %path%), в этом случае в месте использования такой конструкции будет подставлено значение переменной. Некоторые из этих переменных стандартизованы (PROMPT, PATH, TEMP и т. п.), некоторые задаются системой (CONFIG), остальные задаются и используются пользователями. Хранятся эти переменные в «окружении» (environment) и называются «переменными окружения».

3 Аргументы пакетных файлов в самих пакетных файлах доступны как %1…%9. Переменная %0 содержит текст команды (без аргументов), использованной для запуска пакетного файла [18].

4 Переменные для команды FOR имеют вид %%a и используются в пакетных файлах совместно с этой командой

Библиотека БГУИР

Cmd.exe

Cmd.exe – интерпретатор командной строки (англ. *command line interpreter*) для операционных систем OS/2, Windows CE и для семейства операционных систем, базирующихся на Windows NT (англ. *Windows NT-based*). Cmd.exe является аналогом COMMAND.COM

В отличие от command.com, cmd.exe в системах OS/2 и семействе Windows NT имеет более детальные сообщения, чем общее «Неверная команда или имя файла» (англ. «*Bad command or file name*») в случае неправильно введенных команд. Сообщения об ошибках cmd.exe выводит на том языке, который установлен в системе как текущий.

PowerShell

Windows PowerShell – расширяемое средство автоматизации от Microsoft, состоящее из оболочки с интерфейсом командной строки и сопутствующего языка сценариев.

Windows PowerShell 2.0 был выпущен в составе Windows 7, Windows 8 и Windows Server 2008 R2 как неотъемлемый компонент системы. Кроме того, вторая версия доступна и для других систем, таких как Windows XP SP3, Windows Server 2003 SP2, Windows Vista SP1, Windows Vista SP2 и Windows Server 2008.

Windows PowerShell построен на базе Microsoft .NET Framework и интегрирован с ним. Дополнительно PowerShell предоставляет удобный доступ к COM, WMI и ADSI, равно как и позволяет выполнять обычные команды командной строки, чтобы создать единое окружение, в котором администраторы смогли бы выполнять различные задачи на локальных и удаленных системах.

Windows PowerShell также предоставляет механизм встраивания, благодаря которому исполняемые компоненты PowerShell могут быть встроены в другие приложения. Эти приложения затем могут использовать функциональность PowerShell для реализации различных операций, включая предоставляемые через графический интерфейс.

Командлеты (англ. *cmdlets*) – это специализированные команды PowerShell, которые реализуют различную функциональность. Это встроенные в PowerShell команды. Командлеты именуются по правилу «глагол + существительное», например Get-ChildItem, благодаря чему их предназначение понятно из названия. Командлеты выводят результаты в виде объектов или их коллекций. Дополнительно командлеты могут получать входные данные в такой же форме и, соответственно, использоваться как получатели в конвейере. Хотя PowerShell

Библиотека БГУИР

позволяет передавать по конвейеру массивы и другие коллекции, командлеты всегда обрабатывают объекты поочередно. Для коллекции объектов обработчик командлета вызывается для каждого объекта в коллекции по очереди.

Bash

Bash (от англ. *Bourne again shell* – «возрожденный» shell) – усовершенствованная и модернизированная вариация командной оболочки Bourne shell. Одна из наиболее популярных современных разновидностей командной оболочки UNIX. Особенно популярна в среде Linux, где она часто используется в качестве предустановленной командной оболочки.

Bash – это командный процессор, работающий, как правило, в интерактивном режиме в текстовом окне. Bash также может читать команды из файла, который называется скриптом (или сценарием). Как и все UNIX-оболочки, он поддерживает автодополнение названий файлов и папок, подстановку вывода результата команд, переменные, контроль за порядком выполнения, операторы ветвления и цикла. Ключевые слова, синтаксис и другие основные особенности языка были заимствованы из sh. Другие функции, например история, были скопированы из csh и ksh.

Более подробно номенклатура команд cmd и bash изложена на ресурсах http://cmdhelp.ru/ и http://lib.ru/unixhelp/unixshell.txt соответственно.

1.1.2 Системы контроля версий

В процессе выполнения второй части лабораторной работы студенту необходимо ознакомиться с программами git и gitk, освоить их программные аргументы.

Порядок выполнения лабораторной работы

Создать текстовый документ с использованием git, содержащий текст на тему «Почему я выбрал специальность „Искусственный интеллект”, и что я жду от своей профессиональной карьеры». Выполнить указанную ниже последовательность шагов в соответствии с вариантом.

Последовательность шагов: создайте новый репозиторий, добавьте туда пустой текстовый документ, сформируйте этот документ, создавая commit для каждого абзаца.

Для защиты второй части лабораторной работы необходимо на практике уметь выполнять всевозможные действия с репозиторием и отвечать на теоретические вопросы.

Библиотека БГУИР

Система контроля версий

Система контроля версий (СКВ) – это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определенным старым версиям этих файлов. Для примеров в данном пособии мы будем использовать исходные коды программ, но на самом деле под версионный контроль можно поместить файлы практически любого типа.

Если вы графический или веб-дизайнер и хотели бы хранить каждую версию изображения или макета, а этого вам наверняка хочется, то пользоваться системой контроля версий будет очень мудрым решением. СКВ дает возможность возвращать отдельные файлы к прежнему виду, возвращать к прежнему состоянию весь проект, просматривать происходящие со временем изменения, определять, кто последним вносил изменения во внезапно переставший работать модуль, кто и когда внес в код какую-то ошибку, и многое другое. Вообще, если, пользуясь СКВ, вы все испортите или потеряете файлы, их можно будет легко восстановить. Вдобавок накладные расходы за все, что вы получаете, будут очень маленькими.

Локальные системы контроля версий

Многие предпочитают контролировать версии, просто копируя файлы в другой каталог (как правило, добавляя текущую дату к названию каталога). Такой подход очень распространен, потому что прост, но он и чаще дает сбои. Очень легко забыть, что ты не в том каталоге, и случайно изменить не тот файл либо скопировать файлы не туда, куда хотел, и затереть нужные файлы.

Чтобы решить эту проблему, программисты уже давно разработали локальные СКВ с простой базой данных, в которой хранятся все изменения нужных файлов (рисунок 1).

Библиотека БГУИР

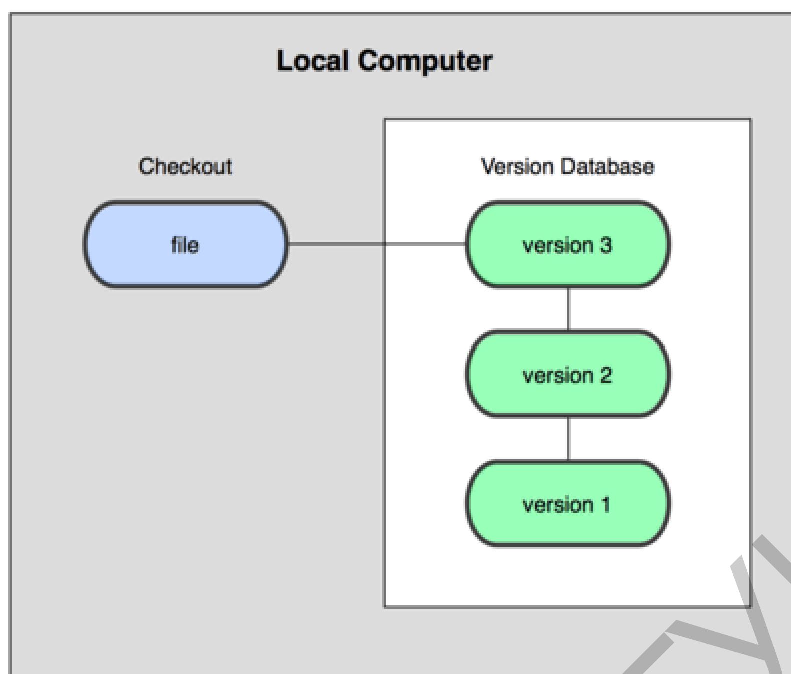

Рисунок 1 – Схема локальной СКВ

Одной из наиболее популярных СКВ такого типа является rcs, которая до сих пор устанавливается на многие компьютеры. Даже в современной операционной системе Mac OS X утилита rcs устанавливается вместе с Developer Tools. Эта утилита основана на работе с наборами патчей между парами версий (патч – файл, описывающий различие между файлами), которые хранятся в специальном формате на диске. Это позволяет пересоздать любой файл в любой момент времени, последовательно накладывая патчи.

Централизованные системы контроля версий

Следующей основной проблемой оказалась необходимость сотрудничать с разработчиками за другими компьютерами. Чтобы решить ее, были созданы централизованные системы контроля версий (ЦСКВ). В таких системах, например CVS, Subversion и Perforce, есть центральный сервер, на котором хранятся все файлы под версионным контролем, и ряд клиентов, которые получают копии файлов из него. Много лет это было стандартом для систем контроля версий (рисунок 2).

Библиотека БГУИР

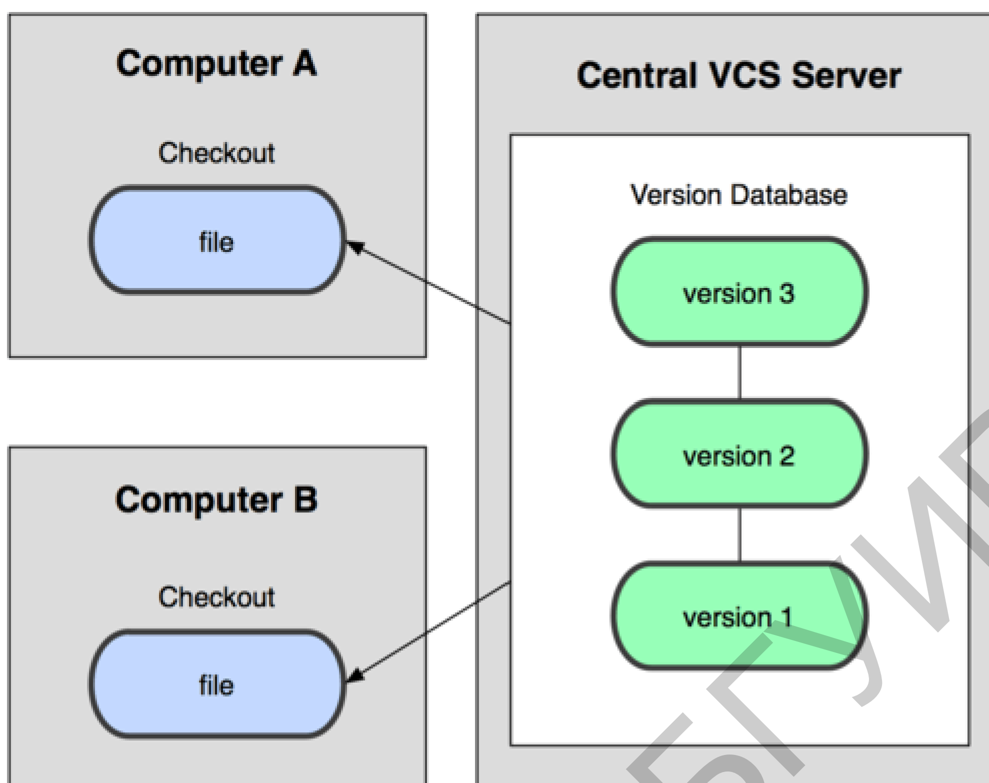

Рисунок 2 – Схема централизованного контроля версий

Библиотека БГУИР

Такой подход имеет множество преимуществ, особенно над локальными СКВ. К примеру, все знают, кто и чем занимается в проекте. У администраторов есть четкий контроль над тем, кто и что может делать, и, конечно, администрировать ЦСКВ намного легче, чем локальные базы на каждом клиенте.

Однако при таком подходе есть и несколько серьезных недостатков. Наиболее очевидный – централизованный сервер является уязвимым местом всей системы. Если сервер выключается на час, то в течение часа разработчики не могут взаимодействовать и никто не может сохранить новую версию своей работы. Если же повреждается диск с центральной базой данных и нет резервной копии, вы теряете абсолютно все – всю историю проекта, разве что за исключением нескольких рабочих версий, сохранившихся на рабочих машинах пользователей. Локальные системы контроля версий подвержены той же проблеме: если вся история проекта хранится в одном месте, вы рискуете потерять все.

Распределенные системы контроля версий

И в этой ситуации в игру вступают распределенные системы контроля версий (РСКВ). В таких системах, как Git, Mercurial, Bazaar или Darcs, клиенты не просто выгружают последние версии файлов, а полностью копируют весь

репозиторий. Поэтому, в случае когда «умирает» сервер, через который шла работа, любой клиентский репозиторий может быть скопирован обратно на сервер, чтобы восстановить базу данных. Каждый раз, когда клиент забирает свежую версию файлов, он создает себе копию всех данных (рисунок 3).

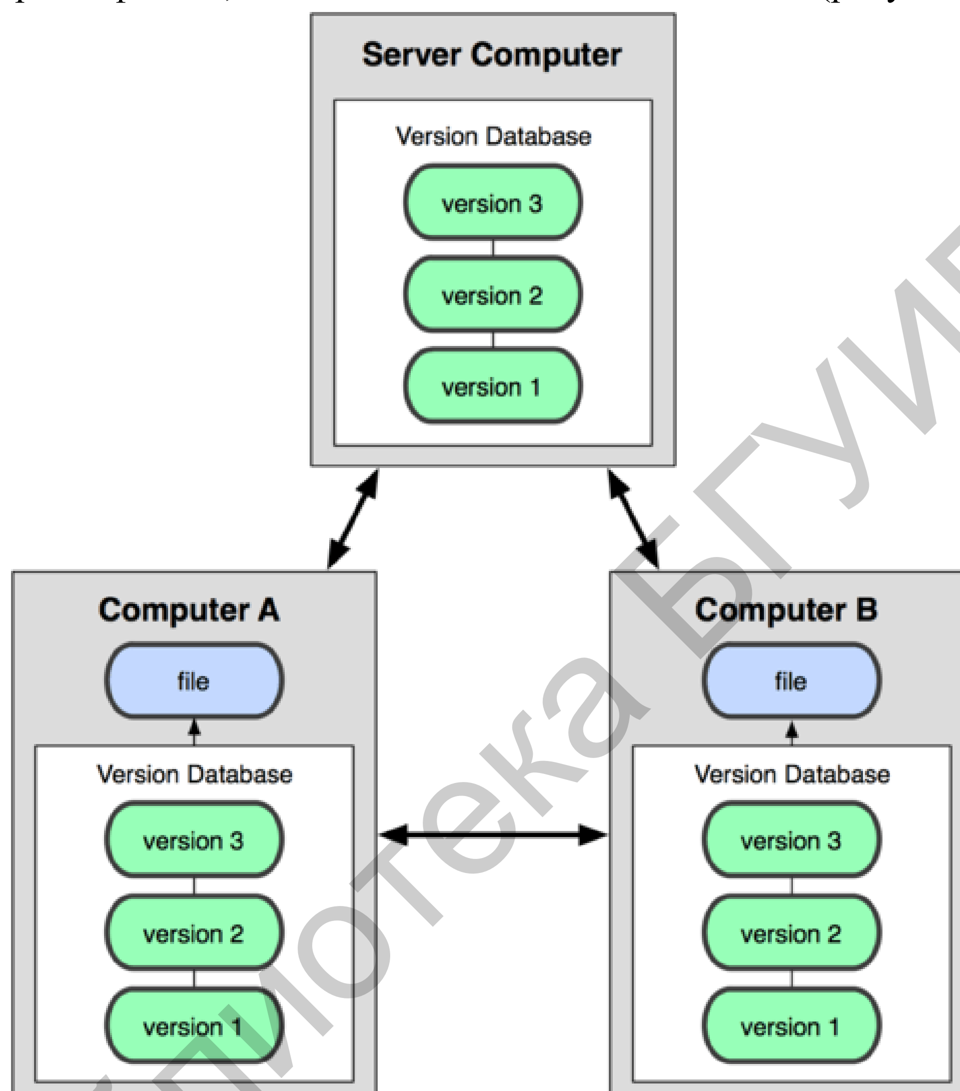

Рисунок 3 – Схема распределенной системы контроля версий

Кроме того, в большей части этих систем можно работать с несколькими удаленными репозиториями, таким образом, можно одновременно работать по-разному с разными группами людей в рамках одного проекта. Так, в одном проекте можно одновременно вести несколько типов рабочих процессов, что невозможно в централизованных системах.

Git

Git (произносится «гит») – распределенная система управления версиями файлов. Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux, первая версия выпущена 7 апреля 2005 г.

Библиотека БГУИР

Примерами проектов, использующих Git, являются ядро Linux, Android, Drupal, Cairo, GNU Core Utilities, Mesa, Wine, Chromium, Compiz Fusion, FlightGear, jQuery, PHP, NASM, MediaWiki, DokuWiki, Qt и некоторые дистрибутивы Linux.

Главное отличие Git от любых других СКВ (например, Subversion и ей подобных) – это то, как Git смотрит на свои данные. В принципе, большинство других систем хранит информацию как список изменений (патчей) для файлов. Эти системы (CVS, Subversion, Perforce, Bazaar и др.) относятся к хранимым данным как к набору файлов и изменений, сделанных для каждого из этих файлов во времени, как показано на рисунке 4.

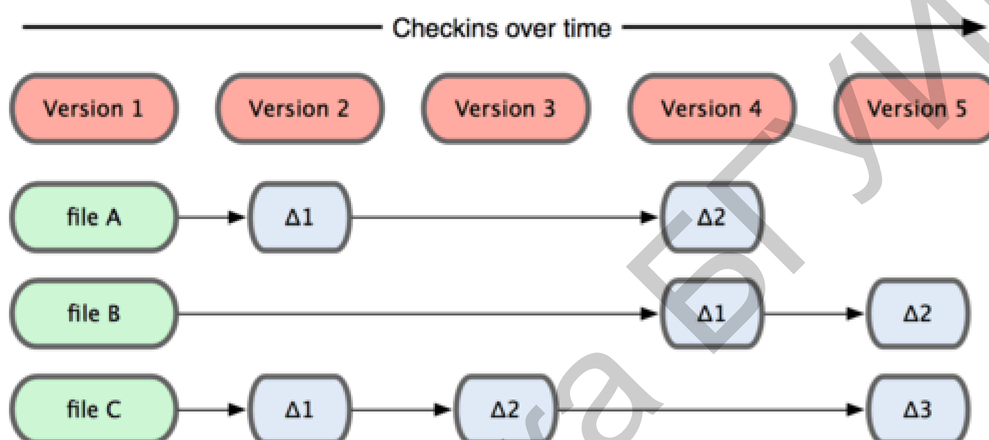

Рисунок 4 – Хранение данных как изменений к базовой версии для каждого файла

Git не хранит свои данные в таком виде. Вместо этого Git считает хранимые данные набором слепков небольшой файловой системы. Каждый раз, когда вы фиксируете текущую версию проекта, Git, по сути, сохраняет слепок того, как выглядят все файлы проекта на текущий момент. Ради эффективности, если файл не менялся, Git не сохраняет файл снова, а делает ссылку на ранее сохраненный файл. Подход Git к хранению данных показан на рисунке 5.

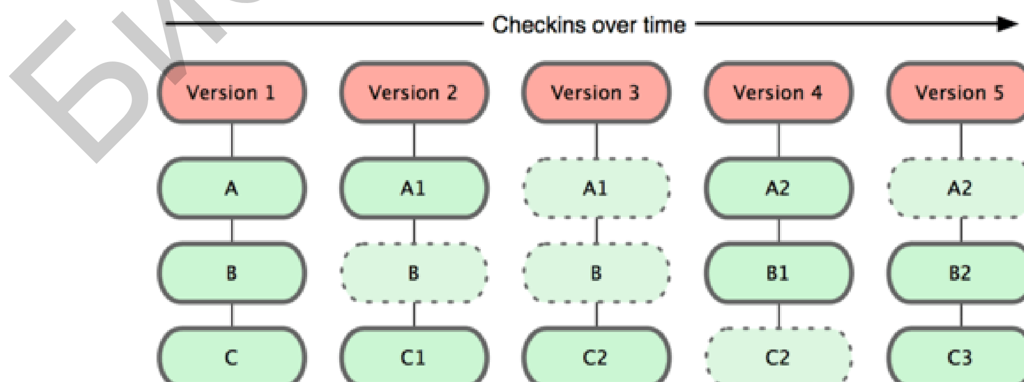

Рисунок 5 – Хранение данных как слепков состояний проекта во времени

Библиотека БГУИР

Это важное отличие Git от практически всех других систем контроля версий. Из-за него Git вынужден пересмотреть практически все аспекты контроля версий, которые другие системы переняли от своих предшественниц. Git больше похож на небольшую файловую систему с невероятно мощными инструментами, работающими поверх нее, чем на просто СКВ. Такое понимание данных дает ряд важных преимуществ.

Для совершения большинства операций в Git необходимы только локальные файлы и ресурсы, т. е. обычно информация с других компьютеров в сети не нужна. Если вы пользовались централизованными системами, где практически на каждую операцию накладывается сетевая задержка, вы, возможно, будете удивлены скоростью работы Git. Поскольку вся история проекта хранится локально у вас на диске, большинство операций кажутся практически мгновенными. К примеру, чтобы показать историю проекта, Git не нужно скачивать ее с сервера, он просто читает ее прямо из вашего локального репозитория. Поэтому историю вы увидите практически мгновенно. Если вам нужно просмотреть изменения между текущей версией файла и версией, сделанной месяц назад, Git может взять файл месячной давности и вычислить разницу на месте, вместо того чтобы запрашивать разницу у СКВ-сервера или качать с него старую версию файла и делать локальное сравнение.

Кроме того, работа локально означает, что много чего нельзя сделать без доступа к сети или VPN. Если вы в самолете или в поезде и хотите немного поработать, можно спокойно делать коммиты, а затем отправить их, как только станет доступна сеть. Если вы пришли домой, а VPN-клиент не работает, все равно можно продолжать работать. Во многих других системах это невозможно или же крайне неудобно. Например, используя Perforce, вы мало что можете сделать без соединения с сервером. Работая с Subversion и CVS, вы можете редактировать файлы, но сохранить изменения в вашу базу данных нельзя (потому что она отключена от репозитория).

Перед сохранением любого файла Git вычисляет контрольную сумму, и она становится индексом этого файла. Поэтому невозможно изменить содержимое файла или каталога так, чтобы Git не узнал об этом. Эта функциональность встроена в сам фундамент Git и является важной составляющей его философии. Если информация потеряется при передаче или повредится на диске, Git всегда это выявит.

Механизм, используемый Git для вычисления контрольных сумм, называется SHA-1 хешем. Это строка из 40 шестнадцатеричных символов (0–9 и a–f), вычисляемая в Git на основе содержимого файла или структуры каталога. SHA-1 хеш выглядит примерно так:

Библиотека БГУИР

24b9da6552252987aa493b52f8696cd6d3b00373

Работая с Git, вы будете встречать эти хеши повсюду, поскольку он их очень широко использует. Фактически в своей базе данных Git сохраняет все не по именам файлов, а по хешам их содержимого.

Практически все действия, которые вы совершаете в Git, только добавляют данные в базу. Очень сложно заставить систему удалить данные или сделать что-то неотменяемое. Можно, как и в любой другой СКВ, потерять данные, которые вы еще не сохранили, но как только они зафиксированы, их очень сложно потерять, особенно если вы регулярно отправляете изменения в другой репозиторий.

Состояния файлов

В Git файлы могут находиться в одном из трех состояний: зафиксированном, измененном и подготовленном. Зафиксированный файл – файл, который уже сохранен в вашей локальной базе. К измененным относятся файлы, которые поменялись, но еще не были зафиксированы. Подготовленные файлы – это измененные файлы, отмеченные для включения в следующий коммит.

Таким образом, в проектах, использующих Git, есть три части: каталог Git (Git directory), рабочий каталог (working directory) и область подготовленных файлов (staging area) (рисунок 6).

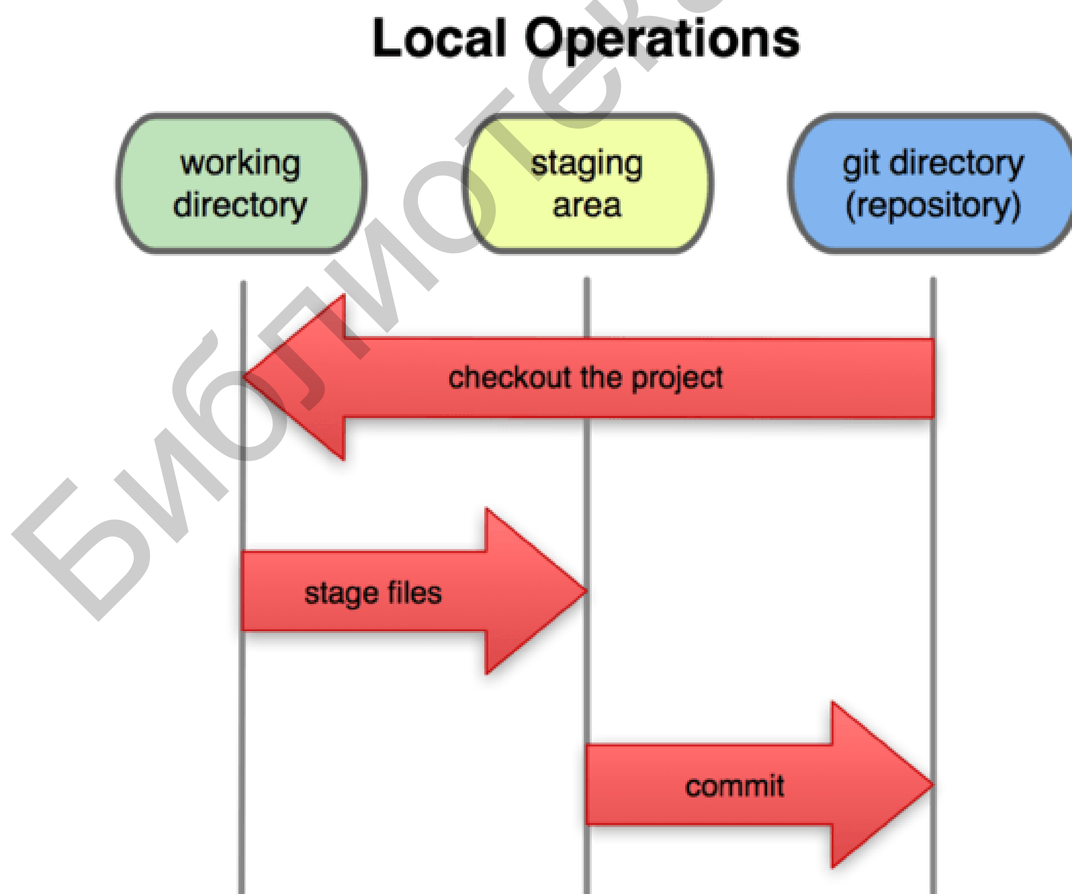

Рисунок 6 – Рабочий каталог, область подготовленных файлов, каталог Git

Библиотека БГУИР

Каталог Git – это место, где Git хранит метаданные и базу данных объектов вашего проекта. Это наиболее важная часть Git, и именно она копируется, когда вы клонируете репозиторий с другого компьютера.

Рабочий каталог – это извлеченная из базы копия определенной версии проекта. Эти файлы достаются из сжатой базы данных в каталоге Git и помещаются на диск для того, чтобы вы их просматривали и редактировали.

Область подготовленных файлов – это обычный файл, чаще всего хранящийся в каталоге Git, который содержит информацию о том, что должно войти в следующий коммит. Иногда его называют индексом (index), но в последнее время становится стандартом именно название область подготовленных файлов (staging area).

Стандартный рабочий процесс с использованием Git выглядит примерно так:

1) вносятся изменения в файлы в своем рабочем каталоге;
2) подготавливаются файлы, добавляются их слепки в область подготовленных файлов;
3) делается коммит, который берет подготовленные файлы из индекса и помещает их в каталог Git на постоянное хранение.

Если рабочая версия файла совпадает с версией в каталоге Git, файл считается зафиксированным. Если файл изменен, но добавлен в область подготовленных данных, он подготовлен. Если же файл изменился после выгрузки из БД, но не был подготовлен, то он считается измененным.

Номенклатура команд и описание самой системы контроля версий Git более подробно изложены на официальном сайте http://git-scm.com/book/ru/v1.

1.2 Варианты индивидуальных заданий

Работа в командной строке

1 Посмотреть справку по команде (не менее двух способов).
2 Создать пустой файл с указанным именем (не менее двух способов).
3 Создать файл с заданным атрибутом.
4 Создать папку.
5 Скопировать указанный файл в другую папку.
6 Скопировать файлы с указанным расширением в другую папку.
7 Переименовать файл.
8 Переместить файл.
9 Сравнить два файла (набора файлов) и вывести различия между ними.
10 Найти текстовую строку с заданным содержимым во всех файлах заданной директории с учетом поддиректорий.

Библиотека БГУИР

11 Вывести содержимое текстового файла на печать.
12 Удалить файл.
13 Удалить папку.
14 Удалить файлы, в которых есть строка с заданным содержимым в заданной директории.
15 Вывести графическую структуру каталогов диска или папки.
16 Вывести текущую дату.
17 Создать файл в текущей директории, в который записать сегодняшнюю дату.
18 Посчитать и вывести количество символов в указанном файле.
19 Посчитать и вывести количество встречающихся заданных символов в файле.
20 Посчитать и вывести количество встречающихся заданных символов во всех файлах указанной директории.
21 Удалить каталог вместе с файлами и подкаталогами.
22 Все файлы, созданные позже определенной даты, скопировать в заданную папку.
23 Удалить все файлы с заданным расширением из заданной папки, созданные позже заданной даты.
24 Создать папку, затем создать в ней столько текстовых файлов, сколько месяцев прошло с начала текущего года.
25 Создать ссылку на указанный исполняемый файл в заданной директории и запустить ее.
26 Создать файл сценария (.bat/.sh), выполняющий одно из вышеприведенных заданий.

Системы контроля версий

Для хранения удаленных репозиториев использовать бесплатные веб-сервисы, например github.com или bitbucket.org.

1 Создать локальный репозиторий в текущей папке.
2 Посмотреть статус текущего репозитория.
3 Пояснить, что такое ветка и какая ветка является обычно основной.
4 Добавить файл в контекст, который подлежит коммиту.
5 Создать коммит на основе текущего контекста для коммита и указать для него комментарий.
6 Создать коммит, включающий изменения всех наблюдаемых файлов, и указать для него комментарий.
7 Посмотреть протокол коммитов.

Библиотека БГУИР

8 Просмотреть информацию о текущих настройках.
9 Убрать файл из контекста.
10 Посмотреть изменения в файле по сравнению с последним коммитом.
11 Убрать изменения относительно последнего коммита из файла.
12 Просмотреть в графическом интерфейсе историю коммитов.
13 Добавить в контекст коммита все измененные и созданные файлы.
14 Изменить глобальные/локальные настройки.
15 Переписать имя пользователя.
16 Просмотреть существующие ветки.
17 Создать новую ветку.
18 Переключиться на другую ветку.
19 Создать новую ветку и сразу же переключиться на нее.
20 Удалить ветку.
21 Просмотреть в графическом интерфейсе историю коммитов по всем веткам.
22 Добавить (merge) изменения из указанной ветки в текущую.
23 Создать конфликт.
24 Посмотреть в каких файлах есть конфликты.
25 Устранить конфликты.
26 Переключиться на указанный коммит.
27 Сделать ребазирование (rebase) текущей ветки.
28 Устранить конфликты во время ребазирования.
29 Отменить ребазирование во время конфликтов.
30 Пропустить текущий конфликтный коммит и перейти к следующему.
31 Отправить изменения из локального репозитория для указанной ветки в удаленный (дистанционный).
32 Забрать изменения из репозитория, для которого были созданы удаленные ветки по умолчанию.
33 Забрать изменения удаленной ветки из репозитория основной ветки по умолчанию.
34 Создать копию репозитория.
35 Удалить и переименовать файл.
36 Создать, удалить, отправить файл в удаленный репозиторий, удалить файл в удаленном репозитории.
37 Отправить ветку в удаленный репозиторий, удалить ветку из удаленного репозитория.

Библиотека БГУИР

ЛАБОРАТОРНАЯ РАБОТА №2 РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ

Цель работы: изучить синтаксис регулярных выражений; создать образцы для поиска.

2.1 Теоретические сведения

В процессе выполнения лабораторной работы студенту необходимо познакомиться с регулярными выражениями и научиться их использовать для проверки корректности строки, поиска необходимой подстроки и замены подстрок в строке.

Рекомендации

Для написания и проверки регулярных выражений можно использовать онлайн-сервисы:

- http://gskinner.com/RegExr/ – **рекомендуемый вариант**;
- http://regexpr.ru/ – здесь регулярное выражение нужно обрамлять специальными символами (delimiters), например `~: , ~[0-9a-z]+~` , т. е. выражение должно начинаться и заканчиваться этим символом;
- любой другой способ (другие онлайн-сервисы, скрипты, настольные приложения и др.).

Так как существует множество реализаций движков регулярных выражений, то важно учитывать стандарт, которому тот или иной движок соответствует. В лабораторной работе мы рассматриваем PCRE (Perl-Compatible Regular Expressions) как наиболее распространенный в настоящее время. Так, например, регулярные выражения в JavaScript не поддерживают рекурсию, которая может пригодиться в третьем задании.

Синтаксис регулярных выражений

Регулярные выражения – это широко используемый способ описания шаблонов для поиска текста и проверки соответствия текста шаблону. Специальные метасимволы позволяют определять, например, что вы ищете подстроку в начале входной строки или определенное число повторений подстроки.

Регулярные выражения могут сэкономить многие часы ненужного кодирования, а в некоторых случаях будут и быстрее работать, чем вручную закодированные проверки.

Простое сравнение

Любой символ совпадает с самим собой, если он не относится к специальным метасимволам, описанным ниже.

Библиотека БГУИР

Последовательность символов совпадает с такой же последовательностью во входной строке, так что шаблон «temp» совпадет с подстрокой «temp» во входной строке.

Если необходимо, чтобы метасимволы или escape-последовательности воспринимались как обычные символы, их нужно предварять символом «\», например, метасимвол «^» обычно совпадает с началом строк, однако если записать его как «\^», то он будет совпадать с символом «^», «\\» совпадает с «\» и т. д.

Примеры:

foobar находит 'foobar'

\^FooBarPtr находит '^FooBarPtr'

Escape-последовательности

Любой символ может быть определен с помощью escape-последовательности, так же как это делается в языках C или Perl. Например:

\xnn – символ с шестнадцатеричным кодом nn;

\x{nnnn} – символ с шестнадцатеричным кодом nnnn (более одного байта можно задавать только в режиме Unicode);

\t – табуляция (HT/TAB), можно также \x09;

\n – новая строка (NL), можно также \x0a;

\r – возврат каретки (CR), можно также \x0d;

\f – перевод формата (FF), можно также \x0c;

\a – звонок (BEL), можно также \x07;

\e – escape (ESC), можно также \x1b.

Вообще, \xnn, где nn – это последовательность шестнадцатеричных цифр, означает символ с ASCII-кодом nn. Если необходимо определить двухбайтный (Unicode) символ, используется формат «\x{nnnn}», где «nnnn» – одна или более шестнадцатеричных цифр.

Примеры:

foo\x20bar находит 'foo bar' (обратите внимание на пробел посередине)

\tfoobar находит 'foobar', предшествуемый табуляцией

Перечни символов

Вы можете определить перечень, заключив символы в []. Перечень будет совпадать с любым одним символом, перечисленным в нем.

Если первый символ перечня (сразу после «[») – «^», то такой перечень совпадает с любым символом, не перечисленным в перечне.

Примеры:

foob[aeiou]r находит 'foobar', 'foober' и т. д., но не 'foobbr', 'foobcr' и т. д.

foob[^aeiou]r находит 'foobbr', 'foobcr' и т. д., но не 'foobar', 'foober' и т. д.

Библиотека БГУИР

Внутри перечня символ «-» может быть использован для определения диапазонов символов, например, a-z представляет все символы между «a» и «z» включительно.

Если необходимо включить в перечень сам символ «-», поместите его в начало или конец перечня или предварите «\». Если необходимо поместить в перечень сам символ «]», поместите его в самое начало или предварите «\».

Примеры:

[-az] «a», «z» и «-»

[az-] «a», «z» и «-»

[a\-z] «a», «z» и «-»

[a-z] все 26 малых латинских букв от «a» до «z»

[\n-\x0D] #10, #11, #12, #13

[\d-t] цифра, «-» или «t»

[]-a] символ из диапазона «]»...«a»

Метасимволы

Метасимволы – это специальные символы, являющиеся важнейшим понятием в регулярных выражениях. Существует несколько групп метасимволов:

1 Разделители строк:

^ – начало строки;

$ – конец строки;

\A – начало текста;

\Z – конец текста;

. – любой символ в строке.

Примеры:

^foobar находит «foobar», только если он в начале строки

foobar$ находит «foobar», только если он в конце строки

^foobar$ находит «foobar», только если это единственное слово в строке

foob.r находит «foobar», «foobbr», «foob1r» и т. д.

Метасимвол «^» по умолчанию совпадает только в начале входного текста, а метасимвол «$» – только в конце текста. Внутренние разделители строк, имеющиеся в тексте, не будут совпадать с «^» и «$».

Однако если необходимо работать с текстом как с многострочным, чтобы «^» совпадал после каждого разделителя строки внутри текста, а «$» – перед каждым разделителем, то можно включить модификатор «/m».

Метасимволы «\A» и «\Z» аналогичны «^» и «$», но на них не действует модификатор «/m», т. е. они всегда совпадают только с началом и концом всего входного текста.

Библиотека БГУИР

Метасимвол «.» по умолчанию совпадает с любым символом, однако если выключить модификатор /s, то «.» не будет совпадать с разделителями строк.

Метасимвол «^» совпадает с началом входного текста, а также, если включен модификатор /m, с точкой, непосредственно следующей после \x0D\x0A, \x0A или \x0D (если используется Unicode-версия выражений, то совпадает также с \x2028, или \x2029, или \x0B, или \x0C, или \x85). Обратите внимание, что он не совпадает в промежутке внутри последовательности \x0D\x0A.

Метасимвол «$» совпадает с концом входного текста, а также, если включен модификатор /m, с точкой, непосредственно предшествующей \x0D\x0A, \x0A или \x0D (если используется Unicode-версия выражений, то совпадает также с \x2028, или \x2029, или \x0B, или \x0C, или \x85). Обратите внимание, что он не совпадает в промежутке внутри последовательности \x0D\x0A.

Метасимвол «.» совпадает с любым символом, но если выключен модификаторr /s, то «.» не совпадает с \x0D\x0A, и \x0A, и \x0D (если используется Unicode-версия выражений, то не совпадает также с \x2028, \x2029, \x0B, \x0C, \x85).

Обратите внимание, что «^.*$» (шаблон для пустой строки) не совпадает с пустой строкой вида \x0D\x0A, но совпадает с \x0A\x0D.

2 Стандартные перечни символов:

\w – буквенно-цифровой символ или «_»;

\W – не \w;

\d – цифровой символ;

\D – не \d;

\s – любой «пробельный» символ (по умолчанию – [\t\n\r\f]);

\S – не \s.

Стандартные перечни \w, \d и \s можно использовать и внутри перечней символов.

Примеры:

foob\dr находит «foob1r», «foob6r» и т. д., но не «foobar», «foobbr» и т. д.

foob[\w\s]r находит «foobar», «foob r», «foobbr» и т. д. но не «foob1r», «foob=r» и т. д.

3 Границы слов:

\b – совпадает на границе слова;

\B – совпадает не на границе слова.

Граница слова (\b) – это точка между двумя символами, один из которых удовлетворяет \w, а другой – \W (в любом порядке), при этом перед началом и после конца строки подразумевается \W.

Библиотека БГУИР

4 Повторения

После любого элемента регулярного выражения может следовать очень важный тип метасимвола – повторитель. Используя их, можно определить число допустимых повторений предшествующего символа, метасимвола или подвыражения:

* – нуль или более раз («жадный»), то же что {0,};

+ – один или более раз («жадный»), то же что {1,};

? – нуль или один раз («жадный»), то же что {0,1};

{n} – точно n раз («жадный»);

{n,} – не менее n раз («жадный»);

{n,m} – не менее n, но не более m раз («жадный»);

*? – нуль или более раз («нежадный»), то же что {0,}?;

+? – один или более раз («нежадный»), то же что {1,}?;

?? – нуль или один раз («нежадный»), то же что {0,1}?;

{n}? – точно n раз («нежадный»);

{n,}? – не менее n раз («нежадный»);

{n,m}? – не менее n, но не более m раз («нежадный»).

Таким образом, {n,m} задает минимум n повторов и максимум – m. Повторитель {n} эквивалентен {n,n} и задает точно n повторов. Повторитель {n,} задает минимум n повторов. Теоретически величина параметров n и m не ограничена, но рекомендуется не задавать большие значения, поскольку в некоторых ситуациях это может потребовать существенных затрат времени и ОЗУ при обработке такого повторителя в связи с рекурсивным характером работы.

Если фигурные скобки встречаются в «неправильном» месте, где они не могут быть восприняты как повторитель, то они воспринимаются просто как символы.

Примеры:

foob.*r находит «foobar», «foobalkjdflkj9r» и «foobr»

foob.+r находит «foobar», «foobalkjdflkj9r», но не «foobr»

foob.?r находит «foobar», «foobbr» и 'foobr', но не «foobalkj9r»

fooba{2}r находит «foobaar»

fooba{2,}r находит «foobaar», «foobaaar», «foobaaaar» и т. д.

fooba{2,3}r находит «foobaar» или «foobaaar», но не «foobaaaar»

Поясним использованное выше понятие «жадности». «Жадные» варианты повторителей пытаются захватить как можно большую часть входного текста, в то время как «нежадные» – как можно меньшую. Например, «b+» как и «b*», примененные к входной строке «abbbbc», найдут «bbbb», в то время как «b+?»

Библиотека БГУИР

найдет только «b», а «b*?» – вообще пустую строку; «b{2,3}?» найдет «bb», в то время как «b{2,3}» найдет «bbb».

Можно переключить все повторители в выражении в «нежадный» режим, воспользовавшись модификатором /g.

5 Варианты

Можно определить перечень вариантов, используя метасимвол «|» для их разделения, например, «fee|fie|foe» найдет «fee» или «fie» или «foe» (так же как «f(e|i|o)e»). В качестве первого варианта воспринимается все от предыдущего метасимвола «(» или «[» или от начала выражения до первого метасимвола «|», в качестве последнего – все от последнего «|'» до конца выражения или до ближайшего метасимвола «)». Обычно, чтобы не запутаться, набор вариантов всегда заключают в скобки, даже если без этого можно было бы обойтись.

Варианты пробуются начиная с первого, и попытки завершаются, сразу же как удастся подобрать такой, при котором совпадет вся последующая часть выражения. Это означает, что варианты не обязательно обеспечат «жадное» поведение. Например, если применить выражение «foo|foot» ко входной строке «barefoot», то будет найдено «foo», т. к. это первый вариант, который позволил совпасть всему выражению.

Обратите внимание, что метасимвол «|» воспринимается как обычный символ внутри перечней символов, например, [fee|fie|foe] означает ровно то же самое, что и [feio|].

Пример:

foo(bar|foo) находит «foobar» или 'foofoo'

6 Подвыражения

Метасимволы (...) могут также использоваться для задания подвыражений по завершении поиска выражения.

Подвыражения нумеруются слева направо в порядке появления открывающих скобок.

Первое подвыражение имеет номер «1» (выражение в целом – «0», к нему можно обращаться в Substitute как «$0», так и «$&»).

Примеры:

(foobar){8,10} находит строку, содержащую 8, 9 или 10 копий «foobar»

foob([0-9]|a+)r находит «foob0r», «foob1r» , «foobar», «foobaar», «foobaar» и т. д.

7 Обратные ссылки

Метасимволы от \1 до \9 воспринимаются как обратные ссылки. \<n> совпадает с ранее найденным подвыражением #<n>.

Примеры:

(.)\1+ находит «aaaa» и «cc».

Библиотека БГУИР

(.+)\1+ также находит «abab» и «123123»

(["']?)(\d+)\1 находит "13" (в двойных кавычках), или '4' (в одиночных кавычках), или 77 (без кавычек) и т. д.

2.2 Варианты индивидуальных заданий

1 Задать регулярными выражениями следующие множества слов:

  1.1 Слова в алфавите {a, b}, такие, что на третьем месте от начала слова стоит буква a, а на пятом месте с конца – буква b.

  1.2 Слова в алфавите {a, b}, в которых число букв четно.

  1.3 Слова в алфавите {a, b}, не содержащие подстроки ab.

  1.4 Слова в алфавите {a, b}, не содержащие подстроки aab.

  1.5 Слова в алфавите {a, b, c}, в которых нет двух соседних букв b.

  1.6 Слова в алфавите {a, b, c}, содержащие подслово вида bxa, где x – произвольная буква алфавита.

  1.7 Слова в алфавите {a, b, c}, в которых за буквой a обязательно следует буква c.

2 Проверить, что строка содержит синтаксически корректный IP-адрес (IPv4).

3 Проверить, что строка является синтаксически корректным математическим выражением. Математическое выражение может содержать следующие символы:

  3.1 Числа.

  3.2 Односимвольные переменные (a–z, A–Z).

  3.3 Знаки математических операций (+, -, *, /).

  3.4 Скобки (круглые).

  Выражение должно быть синтаксически корректным с математической точки зрения.

  Корректные примеры: 17*4+(x-54/(2+4))=y, 2+2, 18-41*c.

  Некорректные примеры: +45, 17+4*, (34+1, 45-3), (4+5)).

Библиотека БГУИР

ЛАБОРАТОРНАЯ РАБОТА №3 РАБОТА С ТЕКСТОВЫМ РЕДАКТОРОМ

Цель работы: создать, отредактировать, отформатировать текстовые документы с помощью системы верстки документов TeX.

3.1 Теоретические сведения

В процессе выполнения лабораторной работы студенту необходимо познакомиться с системой верстки текстов TeX, языком верстки TeX, издательской системой LaTeX.

Для верстки документа использовать онлайн-редактор https://www.sharelatex.com или любые другие LaTeX-редакторы.

Необходимо на практике уметь выполнять различные виды форматирования текста при помощи языка LaTeX в редакторе.

Для защиты лабораторной работы необходимо сверстать одну страницу научной статьи с использованием системы верстки текстов TeX. Текст статьи необходимо взять на сайте конференции OSTIS (http://conf.ostis.net) в соответствии с выданным преподавателем вариантом задания (доклады представлены в форме статей в формате .pdf).

TeX – система компьютерной верстки, разработанная американским профессором информатики Дональдом Кнутом в целях создания компьютерной типографии. Название произносится как «тех» (от греч. τέχνη – искусство, мастерство). В нее входят средства для секционирования документов, для работы с перекрестными ссылками. Многие считают TeX лучшим способом для набора сложных математических формул. В частности, благодаря этим возможностям, TeX популярен в академических кругах, особенно среди математиков и физиков.

Достоинства:

- наивысшее типографское качество при печати документа;
- набор сложных математических формул;
- работает на всех существующих компьютерных платформах;
- гибкие средства для работы с логической структурой текста.

Недостатки:

- не является системой типа WYSIWYG;
- очень сложно писать неструктурированные и неорганизованные документы.

Необходимость использования TeX обосновывается следующим:

Библиотека БГУИР

- TeX является языком международного обмена по математике и физике (большинство научных издательств принимают тексты в печать только в этом формате);
- TeX является средством обмена в рамках Internet/Intranet (система хранения и доступа к статьям, отчетам и дугим документам в формате HTML).

Наиболее актуальную информацию по TeX можно найти по адресу http://www.latex-project.org/.

Основные понятия

Пример самого минимального LaTeX-файла, составленного по всем правилам:

\documentstyle{article}
\begin{document}
Hello, world!
\end{document}

Основные правила при наборе текста:

- исходный текст не должен содержать переносов (TeX сделает их сам);
- символ «return» – это то же самое, что пробел;
- два пробела рядом считаются за один пробел;
- абзацы разделяются одной или несколькими пустыми строками.

Специальные символы:

% – комментарии;
{ – начало группы;
} – конец группы;
$ – ввод математики;
_ – нижние индексы математики;
^ – верхние индексы математики;
~ – неразрывный пробел;
\ – сигнальный символ (команд);
– параметры в определениях команд;
& – табулятор.

Спецсимволы можно вывести на печать, поставив перед ними символ \.

Команды:

а) командное слово – **имякоманды** (из букв [A–Z, a–z]);

b) командный символ – * **символ-не-буква**.

Команды могут иметь аргументы: {**обязательные**}, [**необязательные**].

Пример: *\documentstyle[12pt,twocolumn]{book}*

Команды могут иметь варианты (со звездочкой).

Библиотека БГУИР

Пример: *section* – начать раздел;

*section** – начать раздел не нумеруя.

Параметры

TeX в каждый момент обработки исходного текста учитывает значения различных параметров: величину абзацного отступа, ширину и высоту страницы и многое другое.

Параметры – частный случай команд, т. е. это команды, которые оперируют числами.

Пример: *parindent=2cm* – задать абзацный отступ.

Группы

Группы – это еще одно важнейшее понятие TeX, как впрочем, и любого другого языка программирования, где оно известно как «блочная структура». Внутри задаются локальные переменные и определения, которые не видны извне.

Пример:

a) {\bf} – слово будет набрано полужирным;

b) *centerline{Эта информация должна быть {\it в центре}}*.

Окружения (environment)

Эта конструкция объединяет в себе принципы предыдущих:

begin{Имя окружения} – начальные действия

/-----

/

| Тело

/

/-----

end{Имя окружения} – конечные действия

Пример:

begin{center}

Все строки этого абзаца

будут центрированы

и переносов слов не будет

end{center}

Оформление документа

documentclass[a4paper,12pt]{article}

usepackage{russian} – пакет поддержки русского языка

Russian – переключение на переносы русских слов и т. д. (обратно: *English*)

Пример:

begin{document}

Библиотека БГУИР

```
% Работая с LaTeX разумно делать заголовки и нумерацию
% разделов не вручную, а с помощью специальных команд.
\author{...}
\title{...} % заглавие документа

\maketitle{...}
\tableofcontents % создает файл *.toc с информацией о содержании

\section{...} % или если необходимо более мелкое деление
 % subsection-subsubsection-paragraph-subparagraph
% Идет соответсвующая нумерация, хотя все зависит от стиля и от того,
% есть ли звездочка при команде

\begin{enumerate} % Кроме команд, разбивающих документ на части,
 \item первый пункт% очень полезным для улучшения зрительного
 \item второй пункт% текста является создание перечней:
 \begin{...} % В LaTeX описаны три вида перечней (окружения):
 \item ... % itemize – простые ( «горошина» - * . )
 \item ... % enumerate – нумерованные ( 1 a I (римские) A)
 \end{...} % description – описательные (конкретные названия)
\end{...}

\label{mylabel} % В LaTeX можно ссылаться практически на любой
 . % элемент текста: на страницу, раздел документа
 . % номер рисунка, номер элемента в нумерованном
 . % перечне

\end{document}
```

Набор текста

Тире

Большинство знаков препинания (точка, запятая, двоеточие) набирается очевидным образом, но некоторые требуют специального набора:

- дефис или знак переноса;

-- черточка или en-тире;

--- тире или em-тире;

$-$ знак минус.

Библиотека БГУИР

Кавычки

Французские «елочки» – *<<елочки>>*

Немецкие „лапки или 99–66“ – *,,лапки или 99--66‘‘*

Английские “лапки или 66–99” – *‘‘лапки или 66--99’’*

Неверно: *„нигде так не принято”*

Неверно: *”и так тоже никто не делает“*

Неверно: *«а это вообще не кавычки»*

Многоточие

Многоточие в тексте и формулах набирают не тремя точками, а командой \dots:

cdots – центрированное (обычно в математике между знаками «+», «-» и т. д.);

vdots – вертикальное (обычно в математике);

ddots – диагональное (в матрицах и т. д.).

Верно: *подумал\dots и сказал*

Верно: *$i=1,2,\dots,n$.*

Неверно: *подумал... и сказал*

Неверно: *$i=1,2,...,n$.*

Прочие знаки:

S – знак номера параграфа;

dag – кинжал или обелиск;

copyright – знак авторского права;

pounds – знак фунта стерлингов.

В тексте можно использовать любой дополнительный символ, заключив его в $...$, например:

Я $\heartsuit$ тебя

Это слово будет \underline{подчеркнуто}

Два слова будут \fbox{в рамке}

Диакритические знаки – дополнительные знаки, размещающиеся над или под буквой (для разных европейских языков):

\′ – постановка ударения;

\″ – для русской буквы «ё».

Пробелы:

enskip – пробел в 0,5 em;

quad – пробел в 1 em;

qquad – пробел в 2 em;

hspace{длина} – конкретный промежуток;

, – тонкий пробел (шпация);

: – средний пробел;

; – толстый пробел.

Библиотека БГУИР

Шрифты

Команды смены семейства, насыщенности и начертания шрифтов действуют на свой аргумент и могут комбинироваться в различных сочетаниях.

Виды шрифта:

bf (boldface) – полужирный;

it (italic) – курсив;

sl (slanted) – наклонный;

sf (sans serif) – рубленый;

sc (Small Caps) – капитель;

tt (typewriter) – имитация пишущей машинки;

rm (roman) – прямой светлый.

Размеры:

tiny – самый маленький;

scriptsize – очень маленький (как индексы);

footnotesize – маленький (как сноски);

small – мелкий;

normalsize – нормальный;

large – большой;

Large – очень большой;

LARGE – совсем большой;

huge – громадный;

Huge – самый громадный.

Реальный размер шрифтов зависит от выбранного стиля.

Формирование абзацев

Указать перенос в слове:

a) \\- – «одноразовый (на это слово)» способ, например *на\\-ем\\-ник*;

b) *hyphenation* – «глобальный (на весь документ)» способ, например *hyphenation{вклю-чен об-ласть}*.

Запретить перенос в слове:

a) *mbox* (make box) – «одноразовый» способ (например, *параметр \\mbox{filename}* задает имя файла);
b) \\- – «одноразовый, хулиганский» способ позволяет запретить перенос в слове, поставив команду в его конце (например, *это слово корова\\-перенесено не будет*);
c) *hyphenation{...,корова,...}* – «глобальный» способ, при котором необходимо указать слово без дефисов.

Указать разрыв между словами:

Библиотека БГУИР

a) \\ – принудительный разрыв строки без ее растяжения (например, *Эта строка\\ была разорвана);*

b) *\linebreak* – принудительный разрыв строки и ее дальнейшее выравнивание по правому краю (например, э*та строка была \linebreak разорвана*).

Запретить разрыв между словами можно символами неразрывного пробела ~.

Закончить абзац (эквивалентные варианты):

a) пустая строка;

b) команда \par.

Задать необходимый промежуток между абзацами можно следующими командами:

a) *\smallskip* – маленький вертикальный пробел;

b) *\medskip* – вертикальный пробел побольше;

c) *\bigskip* – еще больше (точные размеры зависят от стиля и кегля);

d) *\vspace*{*...*} – промежуток конкретного размера.

Можно задать промежуток не фиксированной, а переменной длины:

\vspace{x plus y minus z}, здесь x, y, z – длины, *plus, minus* – ключевые слова (без backslash).

Интерлиньяж – интервал между строками (устанавливается автоматически для каждого шрифта). Иногда бывает необходимо пропорционально изменить все интервалы. Это возможно осуществить с помощью команды *\renewcommand{\baselinestretch}{1.01}*, которая увеличит интерлиньяж на 1 %.

Формирование страниц

Принудительный разрыв страницы:

a) *\newpage* – дополняется снизу пустым пространством;

b) *\clearpage* – дополняется снизу пустым пространством и еще допечатывает плавающие иллюстрации;

c) *\pagebreak* – с «растягиванием» страницы;

d) *\newpage \mbox{} \newpage* – пустая страница.

Запрет разрыва страницы:

a) *\nopagebreak* – локальный, т. е. между конкретными абзацами;

b) *\samepage* – глобальный, т. е. разрывы станут возможны только между абзацами, а не внутри и не между текстом и формулой.

Набор в две колонки:

a) *[twocolumn]* – стилевая опция для всего документа;

b) *\twocolumn[широкий текст]* – сначала выполнить команду \clearpage, затем (если надо) широкий текст, далее начать формировать две колонки (\onecolumn – обратное переключение).

Библиотека БГУИР

Чтобы сделать две колонки в конкретном месте необходимо:

- отменить действие команды *clearpage;*
- использовать окружение *minipage;*
- использовать команду *parbox.*

Выравнивание текста

Выравнивание по левому краю:

begin{flushleft}
Текст \\ выравнивается\\ по левому краю
end{flushleft}

Выравнивание по центру:

begin{center}
Текст\\ выравнивается\\ по центру
end{center}

Выравнивание по правому краю:

begin{flushright}
Текст\\ выравнивается\\ по правому краю
end{flushright}

Таблицы

begin{tabular}{преамбула}
|------
| преамбула это нечто, описывающее колонки:
| *c* – центрированы;
| *l* – выровнены по левому краю;
| *r* – выровнены по правому краю;
| | – задает вертикальную линию;
| *p{ширина колонки}* – для абзацев текста;
| сама таблица формируется с использованием следующих команд:
| \\ – разделяет строки таблицы;
| & – разделяет колонки внутри строки;
| *hline* – задает горизонтальную линию на всю ширину;
| *cline*{первая-последняя} – задание линии по колонкам;
| *multicolumn*{N}{P}{T} – задает «широкую» графу, где:
| *N* – количество «охватываемых» колонок;
| *P* – «преамбула» для данной графы;
| *T* – текст, записываемый в данную графу;
|------
end{tabular}

Библиотека БГУИР

Пример:

```
\begin{tabular}{|l|lr|}
\hline
\multicolumn{3}{|c|}{Западные сладости}\\
\hline
Название & Количество & Цена \\
\hline
Сникерс & штука &9000 \\
\cline{2-3}
& десяток &85000 \\
\hline
Марс & штука &5000 \\
\hline
\end{tabular}
```

Результат отображения таблицы показан на рисунке 7.

Западные сладости		
Название	Количество	Цена
Сникерс	штука	9000
	десяток	85000
Марс	штука	5000

Рисунок 7 – Таблица

Иллюстрации

Для введения в текст иллюстраций вначале надо задать пустое место, не пропадающее при разрыве страницы:

begin{table}[tbph] – плавающая таблица;

begin{figure}[tbph] – плавающая картинка.

|------

| *«tbph»* – пожелание по поводу размещения.

| Это одна (или несколько букв, если допускается любая из указанных):

| *t* – разместить в верхней части страницы;

| *b* – в нижней части страницы;

| *p* – на отдельной «иллюстрационной» странице;

| *h* – разместить именно в этом месте;

| По умолчанию задано [tbp]. Все свойства для «таблиц» и «картинок»

| совпадают, за исключением подписи.

|-----

end{table} или *end{figure}*

Библиотека БГУИР

Пример:

\begin{figure}
\vspace{4cm}
\caption{Белый квадрат}
\label{my_label} |
\end{figure}

Формулы

Основным способом задания «внутренних» формул, т. е. внутри текста, является набор *$...$*.

«Выключенные» формулы, т. е. выделенные в отдельную строку, набираются следующим образом:

a) *$$...$$* – стандартный способ TeX;

b) *{equation}* – способ LaTeX с автоматической нумерацией (стилевые опции: *[leqno]* – нумерация слева, *[flegn]* – формулы слева).

Основные принципы набора формул:

- пробелы игнорируются (TeX их сделает сам);
- пустые строки не разрешаются;
- математическая формула является группой;
- каждая буква рассматривается как имя переменной и набирается шрифтом «математический курсив», поэтому обычный текст включается командой *\mbox*.

Греческие буквы задаются командами по их английским названиям (начинаются с большой буквы для прописных букв), например \psi или \Psi.

Пример:

\alpha \iota \sigma
\beta \kappa \varsigma

Бинарные операции:

+ – плюс;
- – минус;
* – умножение;
\times – умножение «крестиком»;
\div – деление (минус между точками «÷»).

Бинарные отношения:

< – меньше;
> – больше;
= – равно;
\le – меньше либо равно;
\ge – больше либо равно;
\ne – не равно;

Библиотека БГУИР

sim – подобно (одна волна «~»);
approx – приближенно (две волны «≈»);
equiv – эквивалентно (тройное равенство «≡»).
Стрелки различных видов:
to – тонкая стрелочка вправо;
Rightarrow – двойная стрелочка вправо;
gets – тонкая стрелочка влево;
Leftarrow – двойная стрелочка влево.
Тригонометрические и логарифмические функции:
sin *tan* *exp*;
cos *arctan* *dim*;
arcsin *log* *lg*;
arccos *ln.*
Дополнительные операции:
sum – сумма;
prod – произведение;
lim – предел;
inf – инфимум;
max – максимум;
int – интеграл;
min – минимум;
oint – контурный интеграл.
Скобки различных видов:
() – круглые скобки;
[] – квадратные скобки;
\{ \} – фигурные скобки;
| – знак модуля;
\langle – \rangle – угловые скобки.

Автоматическое задание размера по высоте фрагмента формулы осуществляется командами *left(... \right)* .

Эти команды могут появляться только парами, однако скобку можно сделать невидимой, задав вместо нее точку: *left*.

Прочие знаки:
partial – частная производная;
prime или ' – штрих-производная;
forall – «для всех»;
exists – «существование»;
Box – квадрат;

Библиотека БГУИР

Diamond – ромб;

sharp – музыкальный диез;

flat – музыкальный бемоль.

Переносы в формулах:

a) «выключенные» формулы никогда не переносятся автоматически, но можно это сделать, используя некоторые трюки, например окружение *{array}*:

```
$$ \begin{array}{l}
e^x=1+x+\frac{x^2}{2!}\\
\qquad +\frac{x^3}{3!}+\cdots
$$ \end{array}
```

b) «внутренние» формулы переносятся автоматически после знаков бинарных отношений и операций, но часть не может разорваться, ограниченная фигурными скобками.

Буквы в формулах набираются:

a) математическим курсивом *mit* (по умолчанию) – эта команда используется редко;

b) полужирным прямым шрифтом (если командой в преамбуле задать *boldmath*).

Обычный текст в формулах набирается командой *mbox{текст}*, а иначе ТеХ расставит свои маленькие «математические» пробелы.

Примеры:

Неправильный набор команд (вся правая часть сольется в одно слово):

```
$$ 2x=x+x {\rm для всех} x $$
```

Правильный набор команд:

```
$$ 2x=x+x \qquad \mbox{для всех }x. $$
```

В формуле используется шрифт, который был текущим перед ее началом.

Надстрочные знаки в формулах:

a) *overline{...}* – горизонтальная черта над любым фрагментом формулы;

b) *overrightarrow{...}* – стрелка (вектор) над любым фрагментом формулы;

c) «узкие» значки:

hat – шляпка;

tilde – волна;

bar – черточка;

vec – вектор (например, *vec a*);

dot – точка;

ddot – две точки;

d) «широкие» значки:

widehat{...};

Библиотека БГУИР

\widetilde{...} (например, *тождество $ \widehat{f*g}=\hat f\cdot\hat g & означает...*).

Элементарные мелочи:

a) степени и индексы набираются знаками ^ и _ соответственно, например *R^i_{jkl}* или *$R_j{}^i{}_{kl}$*. Во втором случае, оформив индексы к пустой формуле, добились, чтобы они располагались не один под другим;

b) дроби, обозначаемые косой чертой, набираются непосредственно, например *$x+1/x\ge 2$ верно для всех $x>0$*;

c) запятая в десятичной дроби записывается в фигурных скобках, иначе после нее будет поставлен дополнительный пробел, например *$\pi\approx 3{,}14$;*

d) корень набирается командой *\sqrt[показатель]{подкоренное выражение}*, например *$\sqrt[3]{x^3}=x$;*

e) команда *\log_{основание}{аргумент}* используется для того, чтобы основание задавать как нижний индекс;

f) «пределы» у знака суммы (по умолчанию):

- в «выключной» формуле печатаются над и под знаком;
- во «внутренней» формуле печатаются, как и индексы, например *$$ \sum_{i=1}^n n^2 $$;*

g) «пределы» у знака интеграла (по умолчанию) печатаются, как индексы, «сбоку», например *$$ \int_0^1 x^2 dx $$ – от 0 до 1 от x^2*;

- *\limits* обязательно располагать над и под знаком интеграла, например *$$ \int\limits_0^1 x^2 dx $$*;
- *\nolimits* обязательно располагать сбоку от знака интеграла;

Фрагменты, размещаемые друг над другом:

a) *\frac{числитель}{знаменатель}* – запись для дроби обыкновенной (одну букву или цифру можно не брать в скобки), например *$$ \frac12+\frac x 2=\frac{1+x}2 $$;*

b) горизонтальная фигурная скобка:

- *\overbrace{фрагмент формулы}^надпись* – над формулой;
- *\underbrace{фрагмент формулы}_подпись* – под формулой;

c) расположение типа «над-под»:

- *\atop* – общий случай – *{верхняя часть формулы \atop нижняя часть формулы}*, например *$\left\{ij \atop k\right\}$*, здесь {} выполняют две функции;
- *\choose* – биноминальные коэффициенты, например *$n \choose k$*;

Библиотека БГУИР

d) расположение типа «вровень-над» : *\stackrel{будет над строкой}{будет в строке},* например *$A\strackrel{f}{\longrightarrow}B$*, выглядит: *Af⟶B*.

Матрицы

```
\begin{array}{преамбула}
|---------
| преамбула – это ряд букв (по букве на столбец),
| описывающих столбцы:
| c – центрированы;
| l – выровнены по левому краю;
| r – выровнены по правому краю;
| сама матрица формируется с использованием:
| \\ – разделяет строки матрицы;
| & – разделяет элементы столбцов внутри строки;
|---------
\end{array}
```

Пример записи простой квадратной матрицы из *n* элементов (скобки надо записать отдельно):

```
$$
\left(\begin{array}{cccc}
a_{11} & a_{12} & \ldots & a_{1n} \\
a_{21} & a_{22} & \ldots & a_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
a_{n1} & a_{n2} & \ldots & a_{nn}
\end{array}\right)
$$
```

Системы уравнений

Системы можно записывать, используя *{array}*:

```
$$
\left\{
\begin{array}{rcl}
x^2+y^2 & = & 7 \\
x+y & = & 3 \\
\end{array}
\right.
$$
```

Пример резюме:

```
\documentclass[11pt,a4paper]{moderncv}
```


Библиотека БГУИР

```
\moderncvtheme[purple]{classic}
\usepackage[utf8]{inputenc}
\usepackage[scale=0.8]{geometry}

\usepackage[unicode]{hyperref}
\definecolor{linkcolour}{rgb}{0,0.2,0.6}
\hypersetup{colorlinks,breaklinks,urlcolor=linkcolour, linkcolor=linkcolour}

\firstname{Petr}
\familyname{Petrov}
\address{}{Minsk, Belarus}
\mobile{+375 29 123 45 67}
\email{petya.petrov@gmail.com}
\homepage{https://github.com/Petroff}

\makeatletter
\renewcommand*{\bibliographyitemlabel}{\@biblabel{\arabic{enumiv}}}
\makeatother

\begin{document}
\maketitle
\section{Brief description}
\cvline
 {}
 {My main direction - iOS and RubyOnRails development.\newline{}
 This symbiosis allows me to create interacting system of a mobile client and a web service.\newline{}
 I really enjoy OpenSource software and do my best to develop and contribute OpenSource projects which I'm interested in.}

\section{Experience}
\cventry
 {Aug 2013 - Feb 2014}
 {C++/Qt developer}
{WiseTroll (Sole Proprietorship). Minsk, Belarus}
 {}{}
{Assessment of requirements for software development and time estimation.
\newline{}Software development.}
```


Библиотека БГУИР

```
\section{Development skills}
\subsection{Programming Languages}
\cvline
 {Expert}{Objective-C}
\cvline
 {Intermediate}{C, C++, Ruby, Bash}
\cvline
 {Basic}{Perl, LaTeX}
\subsection{Frameworks and tools}
\cvline{iOS}{CocoaPods, Cedar, Calabash-iOS, RestKit, CoreData}
\cvline{RubyOnRails}{Cucumber, RSpec, Rabl, Devise, CanCan, Twitter Bootstrap, Grape, Slim}

\section{Server administration skills}
\cvline
 {Redmine, GitLabHQ}{Deployment, support and migration to another servers (Debian, Ubuntu)}
\cvline
 {Jenkins}{Deployment and support on MacOS X}
\cvline
 {Nginx, Apache}{Setup and configuration on rpm- and deb-based linux systems}

\section{Computer skills}
\cvline
{IDE}{Xcode, Vim, QtCreator}
\cvline
{OS}{Linux (RedHat, Debian), MacOS X}
\cvline
{VCS}{Git, Mercurial}
\cvline
{Development technics}{BDD, TDD, XP, Continuous Integration}

\section{Education}
\cventry
{Sep 2010 - Jun 2015}
{Software Development}
{BSUIR, Minsk, Belarus}
```

Библиотека БГУИР

{}{}
{}
end{document}

Подробное описание формата LaTeX можно найти по адресу http://scibooks.narod.ru/ladocs/index.html.

3.2 Варианты индивидуальных заданий

Оформить, используя формат LaTeX, одну из статей по выбору преподавателя, опубликованных в сборнике материалов какой-либо конференции, например в сборнике материалов международной научно-технической конференции OSTIS-2015 (http://conf.ostis.net/images/8/8b/OSTIS-2015.compressed.pdf).

Библиотека БГУИР

ЛАБОРАТОРНАЯ РАБОТА №4 РАЗРАБОТКА ФРАГМЕНТА ОНТОЛОГИИ ПРЕДМЕТНОЙ ОБЛАСТИ

Цель работы: создать и отредактировать онтологию при помощи программного средства Protégé.

4.1 Теоретические сведения

В процессе выполнения лабораторной работы студенту необходимо изучить средство создания онтологий Protégé-Frames, получить навыки разработки онтологий предметной области.

Для этого необходимо решить следующие задачи:

1) выбрать предметную область для реализации онтологии;
2) проанализировать предметную область, определить назначение онтологии, объем информации, который она должна содержать;
3) изучить средство создания онтологий Protégé и реализовать в нем онтологию для выбранной предметной области;
4) наполнить базу знаний онтологии конкретными фактами и проверить работоспособность выбранных запросов;
5) продемонстрировать полученные результаты и подготовить отчет о проделанной работе.

Онтологией называют схему, состоящую из классов, связанных между собой посредством различных отношений и правил. Это своеобразная форма представления некоторой области знаний в формальном виде. В настоящее время онтологии широко используются в программировании, обучении, различного рода исследованиях.

Создание простой онтологии в Protégé

Допустим, что необходимо разработать онтологию, выполняющую задачу классификации. Для примера возьмем простейшую классификацию попугаев.

В рамках предметной области можно выделить несколько основных классов:

- *попугай* – главный класс, содержащий три класса-наследника – *мелкие попугаи*, *крупные попугаи* и *средние попугаи*;
- *владелец* – класс, содержащий информацию о человеке-владельце;
- *регион* – класс, содержащий информацию о месте обитания попугая.

Далее можно переходить к созданию проекта.

Первым шагом запускаем программу Protégé и создаем новый проект. В окне настроек выбираем Protégé Files (рисунок 8).

Библиотека БГУИР

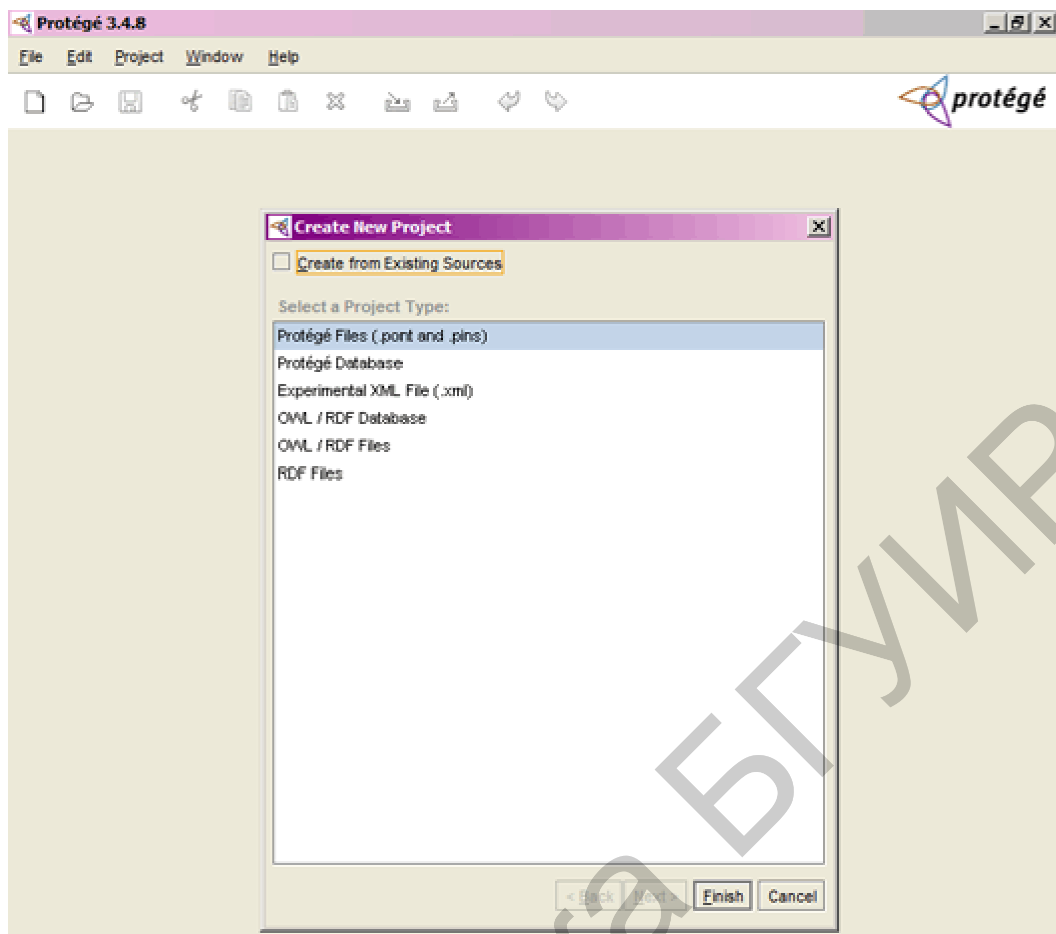

Рисунок 8 – Создание проекта

На экране появляется рабочее окно, в котором нам и предстоит работать.

Первым делом при создании онтологии необходимо создать классы. Все спроектированные нами классы будут отображаться в окне Class Browser. Для создания нового можно щелкнуть на значке Create Class или правой кнопкой мыши в поле браузера классов с указанием действия создания класса (рисунок 9).

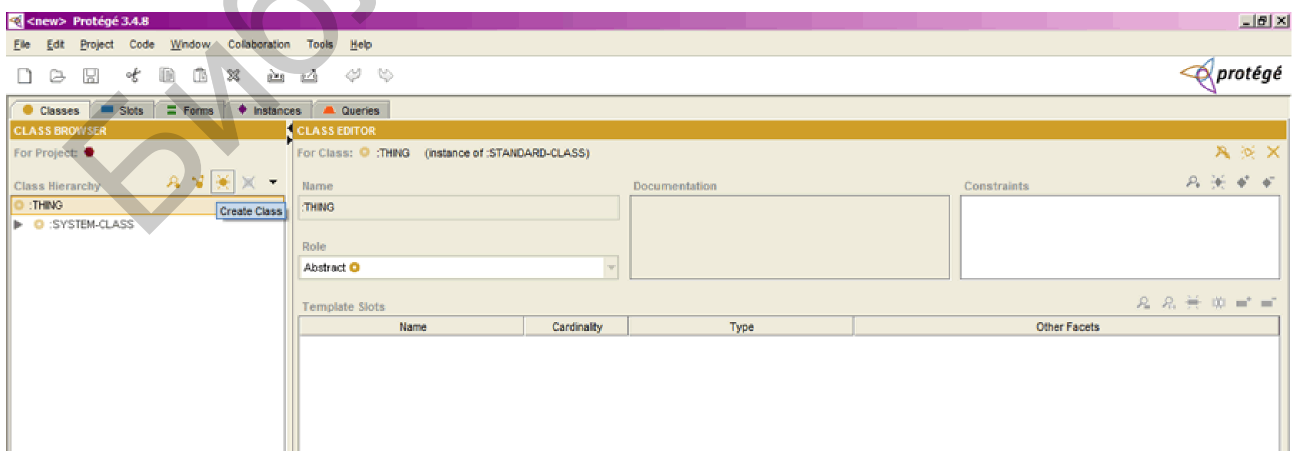

Рисунок 9 – Создание класса

Библиотека БГУИР

После создания класса можно переименовать его в поле Name.

Для создания подкласса щелкаем по классу-родителю, нажимаем правую кнопку мыши и выбираем команду Create Subclass. При желании класс можно сделать абстрактным, выбрав соответствующий пункт в выпадающем списке Role (рисунок 10).

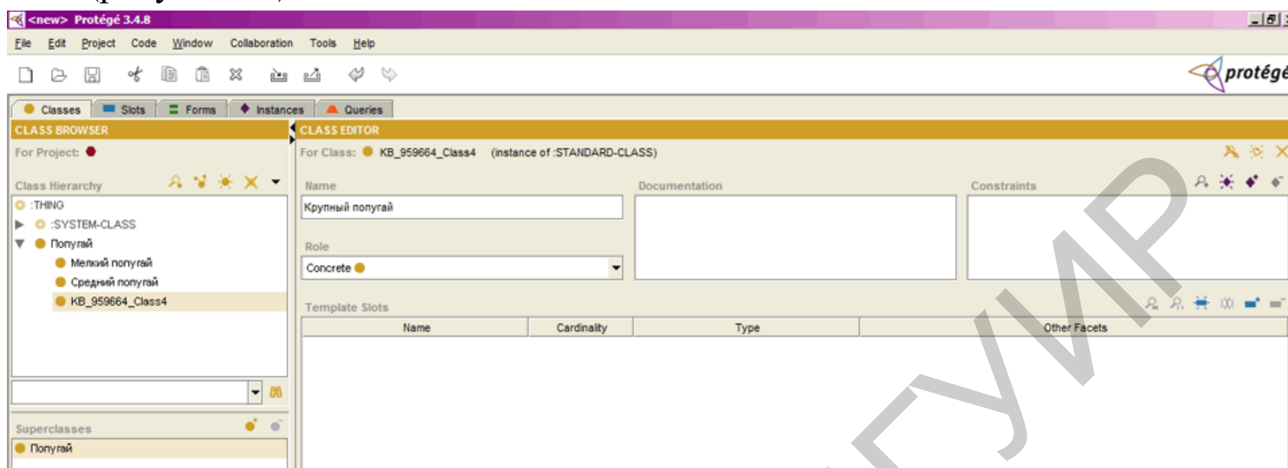

Рисунок 10 – Создание подкласса

После создания классов необходимо прописать в них поля – свойства. К примеру, у класса Регион будет свойство «Имя», которое будет содержать название региона, в котором проживают попугаи. Для добавления свойства в класс необходимо щелкнуть правой кнопкой мыши в окне Template Slots и указать команду Create Slot (рисунок 11).

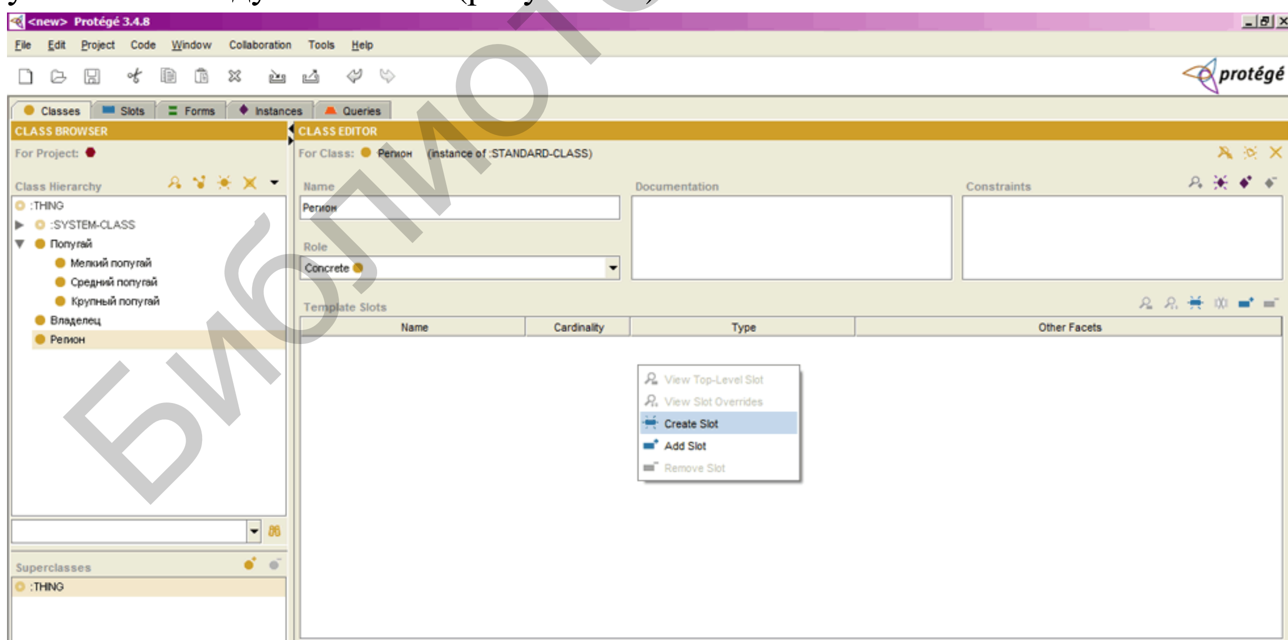

Рисунок 11 – Создание свойств

Библиотека БГУИР

При создании слота ему можно задать название, тип, значение по умолчанию, временное значение, описание и т. п. Стоит отметить, что в качестве типа слота может выступать объект другого класса. Таким способом в программе Protégé устанавливается взаимосвязь между двумя классами.

Если ранее какой-либо слот, например «Имя», уже создавался, то его можно просто добавить в класс (при условии, что он подходит), нажав кнопку в виде прямоугольника с плюсом в правом верхнем углу окна Template Slots (рисунок 12).

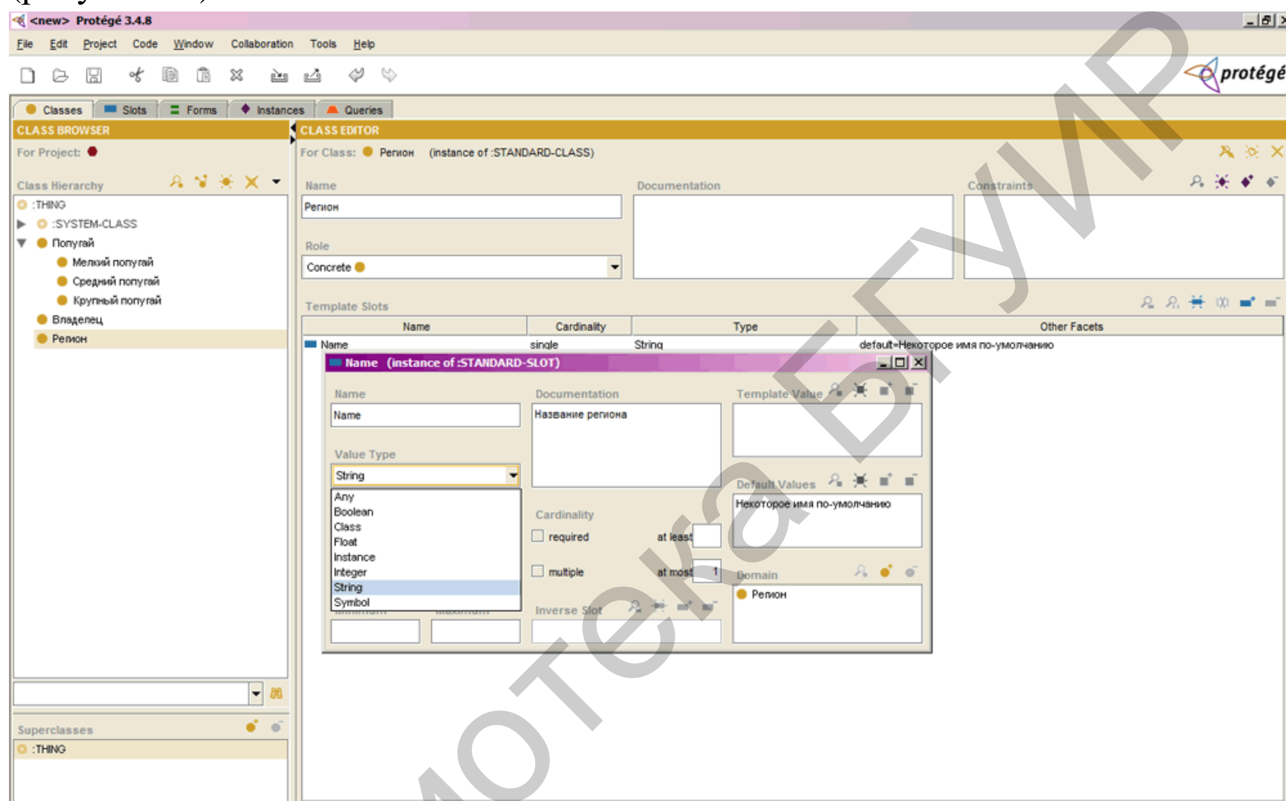

Рисунок 12 – Добавление слотов

После создания слотов и классов можно приступать к созданию экземпляров или Instances. Для этого сверху щелкните на одноименной вкладке и посередине увидите окно под названием Instance Browser.

Для того чтобы создать экземпляр какого-либо класса, щелкните в окне Class Browser на нужном классе, а затем в окне Instance Browser нажмите значек добавления сущности Create Instance, после нажатия которой в окне Instance Browser появится строчка с вновь созданной сущностью, а справа в окне Instance Editor – поля, соответствующие слотам класса, который необходимо заполнить (рисунок 13).

Библиотека БГУИР

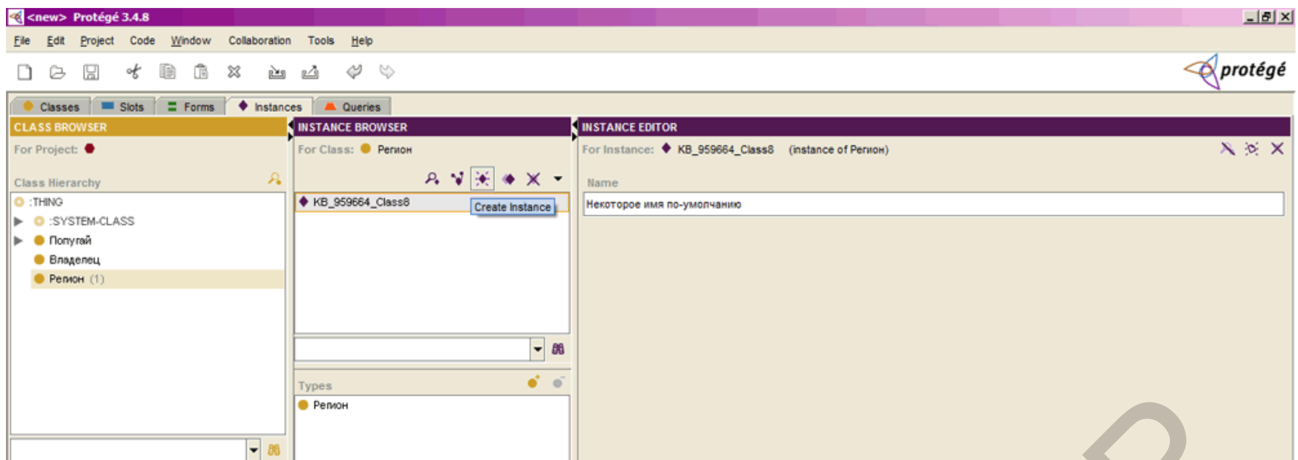

Рисунок 13 – Создание экземпляра

Чтобы в окне Instance Browser отображать объекты по какому-либо признаку, щелкните на треугольнике справа на панели и выберите свойство, которое будет отображаться в браузере. В данном примере укажем свойство «Имя» (рисунок 14).

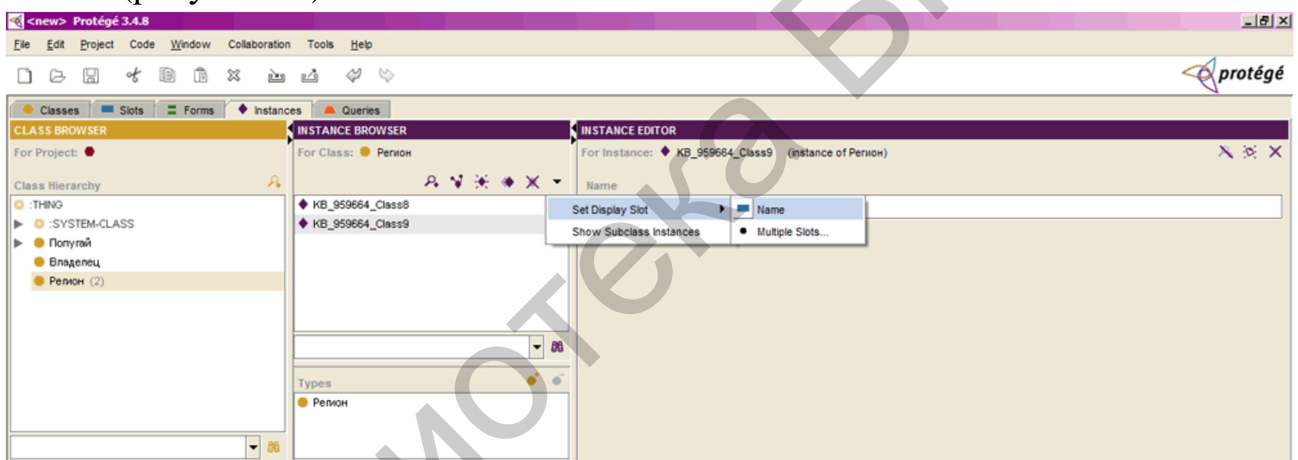

Рисунок 14 – Выборка по признаку

После создания сущностей можно приступать к формированию запросов. Формы для них находятся во вкладке «Запросы».

Чтобы создать необходимый запрос, нужно выбрать класс, в котором будет производиться поиск, свойство, по которому будет производится поиск, а также указать признак. Под признаком может пониматься как строчка, так и условие is, is not, contains, begins with и т. д.

Если запрос необходимо сделать составным, т. е. содержащим два и более условий, в окне следует нажать кнопку more и ввести данные в соответствующие поля (рисунок 15).

Библиотека БГУИР

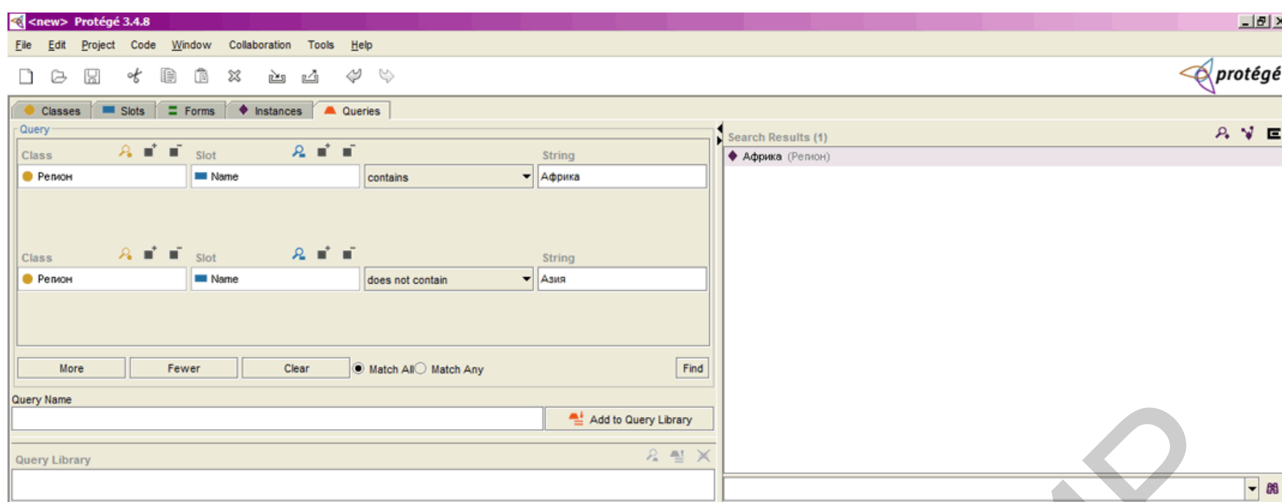

Рисунок 15 – Составной запрос

Созданный запрос можно сохранить и оставить в библиотеке, введя внизу в поле имя и нажав кнопку Add to Query Library.

4.2 Варианты индивидуальных заданий

1 Разработать онтологию предметной области системы единиц СИ.
2 Разработать онтологию предметной области музыкальных стилей.
3 Разработать онтологию предметной области биологии (классификация).
4 Разработать онтологию предметной области созвездий.
5 Разработать онтологию предметной области туманностей.
6 Разработать онтологию предметной области писателей Республики Беларусь.
7 Разработать онтологию предметной области структуры университета.
8 Разработать онтологию предметной области химических элементов.
9 Разработать онтологию предметной области кинофильмов Республики Беларусь.
10 Разработать онтологию предметной области радиоэлементов.
11 Разработать онтологию предметной области галактик.
12 Разработать онтологию предметной области административного деления Республики Беларусь.
13 Разработать онтологию предметной области мировых религий.
14 Разработать онтологию предметной области государственного устройства Республики Беларусь.
15 Разработать онтологию предметной области автомобильных дорог Республики Беларусь.
16 Разработать онтологию предметной области правил дорожного движения.

Библиотека БГУИР

17 Разработать онтологию предметной области управления предприятием.
18 Разработать онтологию предметной области числовых моделей.
19 Разработать онтологию предметной области русского алфавита.
20 Разработать онтологию предметной области английского алфавита.
21 Разработать онтологию предметной области белорусского алфавита.
22 Разработать онтологию предметной области мировых валют.
23 Разработать онтологию предметной области форматов файлов.
24 Разработать онтологию предметной области международной классификации болезней (МКБ 10).
25 Разработать онтологию предметной области художников.
26 Разработать онтологию предметной области рода Радзивиллов.
27 Разработать онтологию предметной области видов спорта.
28 Разработать онтологию предметной области жанров литературы.
29 Разработать онтологию предметной области операторов языка sql.

Библиотека БГУИР

ЛАБОРАТОРНАЯ РАБОТА №5 СТРУКТУРЫ ДАННЫХ

Цель работы: изучить нетривиальные структуры данных в традиционных компьютерах и принципы работы с ними.

5.1 Теоретические сведения

В процессе выполнения данной лабораторной работы, которая является первой во втором семестре изучения дисциплины, студентам необходимо изучить принципы работы и назначение некоторой структуры данных согласно варианту. Для этого необходимо решить следующие задачи:

- ознакомиться с описанием выбранной структуры данных в указанном источнике;
- разработать библиотеку для работы со структурой данных, указанной в индивидуальном задании, на любом императивном языке программирования (Pascal, C\C++, Java, C#, Python и др.);
- разработать тестовую программу, которая демонстрирует работоспособность реализованной библиотеки;
- разработать систему тестов, которые продемонстрировали бы работоспособность реализованной библиотеки. Система тестов должна отвечать требованиям полноты, адекватности и непротиворечивости. Система тестов должна учитывать не только корректную работу на правильных данных, но и предусматривать корректное завершение программы в случае некорректных данных.

По желанию студента допускается использование принципов объектно-ориентированного программирования. К разрабатываемой библиотеке выдвигаются следующие требования:

- возможность создавать одновременно несколько независимых экземпляров структуры данных. В случае реализации библиотеки как набора функций каждая функция должна принимать в качестве входных параметров конкретные экземпляры структур для обработки;
- запрещается использовать глобальные переменные;
- все параметры, необходимые для выполнения какого-либо действия со структурой данных, должны передаваться в функцию или метод из основного приложения. Запрещается запрашивать какие-либо данные непосредственно у пользователя или читать их из файла;
- для представления структуры данных должен использоваться тип данных «запись» (struct, record) или класс, внутри которого используются более

Библиотека БГУИР

простые структуры данных на усмотрение студента. Запрещается явно передавать в функцию экземпляр структуры в виде, например, набора массивов.

Перечень предлагаемых структур данных с источниками информации:

1 Множество [1, с. 19–23].
2 Одномерный массив [2, с. 87–94].
3 Двухмерный массив [2, с. 87–94].
4 Стек [3, с. 260–262].
5 Очередь [3, с. 262–263].
6 Дек [4, с. 5–6].
7 Очередь с приоритетом [3, с. 190–193].
8 Однонаправленный список [2, с. 94–99].
9 Двунаправленный список [3, с. 264–268].
10 Хэш-таблица [3, с. 282–298].
11 Система непересекающихся множеств [3, с. 582–592].
12 Ориентированный граф [3, с. 609–621].
13 Неориентированный граф [3, с. 609–612, 622–630].
14 Бинарное дерево поиска [3, с. 317–327].
15 Красно-черное дерево [3, с. 336–350].
16 AVL-дерево [4, с. 7–8].
17 B-дерево [3, с. 515–533].
18 N-арное дерево [2, с. 219–225].
19 Квадродерево [5, 78–79].
20 Дерево Хаффмана [5, 79–81].
21 Дерево отрезков [4, с. 12].
22 Дерево сумм [4, с. 13].
23 Дерево максимумов [4, с. 9–12].
24 Дерево Фенвика [6, http://e-maxx.ru/algo/fenwick_tree].
25 Декартово дерево [6, http://e-maxx.ru/algo/treap].
26 Таблица Юнга [5, 333–335].
27 Бор [6, http://e-maxx.ru/algo/aho_corasick].
28 Суффиксный массив [6, http://e-maxx.ru/algo/suffix_array].

Библиотека БГУИР

5.2 Варианты индивидуальных заданий

Вариант выбирается в соответствии с номером студента в рамках группы. Если студентов в группе больше, чем вариантов в списке, то варианты снова повторяются, начиная с единицы.

1. Реализовать структуру данных: N-дерево. Вставка узла в дерево. Удаление узла из дерева.
2. Реализовать структуру данных: множество. Добавление элемента во множество. Удаление элемента из множества. Поиск элемента во множестве. Объединение двух множеств. Пересечение двух множеств.
3. Реализовать структуру данных: одномерный массив. Сортировка массива. Вставка элемента в отсортированный массив. Поиск элемента в отсортированном массиве. Объединение двух отсортированных массивов. Пересечение двух отсортированных массивов.
4. Реализовать структуру данных: бинарное дерево поиска. Поиск узла в дереве. Вставка узла в дерево. Удаление узла из дерева. Обходы дерева. Построение дерева из массива.
5. Реализовать структуру данных: дерево отрезков. Поиск количества вхождений какого-то числа на отрезке массива. Изменение всех чисел на отрезке массива на какое-то значение.
6. Реализовать структуру данных: B-дерево. Поиск. Вставка. Удаление.
7. Реализовать структуру данных: суффиксный массив. Построение суффиксного массива из строки. Нахождение наименьшего циклического сдвига строки. Наибольший общий префикс двух подстрок.
8. Реализовать структуру данных: декартово дерево. Вставка. Поиск. Удаление. Построение дерева из массива значений. Объединение двух деревьев. Пересечение двух деревьев.
9. Реализовать структуру данных: AVL-дерево. Вставка. Удаление. Поиск. Поиск минимума, максимума, ближайшего большего и ближайшего меньшего.
10. Реализовать структуру данных: бор. Вставка строки в бор. Удаление строки из бора. Поиск строки в боре.
11. Реализовать структуру данных: дерево максимумов. Поиск максимума на отрезке массива. Изменение всех чисел на отрезке массива на какое-то значение.
12. Реализовать структуру данных: таблица Юнга. Вставка элемента в таблицу. Удаление элемента из таблицы.

Библиотека БГУИР

13 Реализовать структуру данных: красно-черное дерево. Вставка. Удаление. Поиск. Поиск минимума, максимума, ближайшего большего и ближайшего меньшего.
14 Реализовать структуру данных: дерево Фенвика. Поиск суммы чисел на отрезке массива. Обновление числа в массиве.
15 Реализовать структуру данных: квадродерево. Перевод матрицы в дерево. Перевод дерева в матрицу.
16 Реализовать структуру данных: хэш-таблица. Вставка элемента в таблицу. Поиск элемента в таблице. Удаление элемента из таблицы.
17 Реализовать структуру данных: ориентированный граф. Вставка вершины. Удаление вершины. Вставка дуги. Удаление дуги. Построение дерева обхода в ширину.
18 Реализовать структуру данных: дек. Добавление элемента в начало или конец дека. Удаление элемента из начала или конца дека.
19 Реализовать структуру данных: двухмерный массив. Сортировка массива. Вставка элемента в отсортированный массив. Поиск элемента в отсортированном массиве.
20 Реализовать структуру данных: объединение двух отсортированных массивов. Пересечение двух отсортированных массивов.
21 Реализовать структуру данных: дерево сумм. Поиск суммы чисел на отрезке массива. Изменение всех чисел на отрезке массива на какое-то значение.
22 Реализовать структуру данных: система непересекающихся множеств. Создание множества. Объединение двух множеств. Определение множества, которому принадлежит указанный элемент.
23 Реализовать структуру данных: очередь. Вставка элемента в очередь. Взятие элемента из очереди.
24 Реализовать структуру данных: стек. Вставка элемента в стек. Взятие элемента из стека.
25 Реализовать структуру данных: двунаправленный список. Вставка элемента в список. Удаление элемента из списка. Сортировка списка. Поиск элемента в списке. Объединение двух списков. Пересечение двух списков.
26 Реализовать структуру данных: очередь с приоритетом. Вставка элемента в очередь. Взятие элемента из очереди.
27 Реализовать структуру данных: неориентированный граф. Вставка вершины. Удаление вершины. Вставка дуги. Удаление дуги. Построение дерева обхода в глубину.

Библиотека БГУИР

28 Реализовать структуру данных: однонаправленный список. Вставка элемента в список. Удаление элемента из списка. Сортировка списка. Поиск элемента в списке. Объединение двух списков. Пересечение двух списков.
29 Реализовать структуру данных: дерево Хаффмана. Зашифровка текста с помощью дерева.

Библиотека БГУИР

ЛАБОРАТОРНАЯ РАБОТА №6 ПРИМЕНЕНИЕ ТЕОРИИ ГРАФОВ

Цель работы: научиться применять теорию графов в решении задач, сводить различные прикладные задачи к теоретико-графовым.

6.1 Теоретические сведения

Подробные сведения о теории графов можно получить в процессе выполнения практических занятий по курсу «Традиционные и интеллектуальные информационные технологии», а также из предлагаемой литературы.

В процессе выполнения данной лабораторной работы необходимо разработать программу на любом императивном языке программирования, которая решает задачу, предусмотренную вариантом индивидуального задания. Задача считается решенной, если для всех наборов входных данных она получает правильные наборы выходных данных и укладывается в ограничение по памяти и по времени.

Тестирование работоспособности программы осуществляется в автоматическом режиме на наборе тестов, предлагаемых вместе с условием задачи. Для запуска автоматического тестирования необходимо запустить файл сценария AllTest.bat (в некоторых вариантах – TestAll.bat) и в качестве параметра указать имя разработанной программы (в некоторых вариантах расширение *.exe должно быть опущено). При этом формат входных и выходных данных должен в точности соответствовать описанному в тексте задачи.

После успешного прохождения программой тестов необходимо защитить лабораторную работу в устной форме. Для этого необходимо:

1) сформулировать индивидуальное задание в терминах теории графов;
2) дать описание алгоритма решения задачи (подробное пошаговое описание действий или блок-схема);
3) дать описание структур данных, которые были использованы при реализации алгоритма;
4) проанализировать время работы алгоритма (O-оценка времени работы алгоритма, обоснование);
5) проанализировать память, потребляемую реализованной программой (O-оценка, обоснование).

Библиотека БГУИР

6.2 Варианты индивидуальных заданий

Вариант выбирается в соответствии с номером студента в рамках группы. Если студентов в группе больше, чем вариантов в списке, то варианты снова повторяются, начиная с единицы. Полный текст каждой задачи в электронном виде можно найти на сервере информационных ресурсов кафедры.

1 Задача «Roads».
2 Задача «Безопасные пути».
3 Задача «Водородный поезд».
4 Задача «Гамильтонов цикл».
5 Задача «Главные дороги».
6 Задача «Домино».
7 Задача «Жадный гном».
8 Задача «Игра».
9 Задача «Коровьи танцы».
10 Задача «Космическая экспедиция».
11 Задача «Кратчайший путь».
12 Задача «Крестики-нолики 2007».
13 Задача «Кто приносит пирожные».
14 Задача «Кубики».
15 Задача «Куфический дирхем».
16 Задача «Лабиринт знаний».
17 Задача «Максимальный поток».
18 Задача «Метро».
19 Задача «Метро Санкт-Петербурга».
20 Задача «Минимальное остовное дерево».
21 Задача «Мосты и точки сочленения».
22 Задача «Новогодняя вечеринка».
23 Задача «Переливания».
24 Задача «Построение».
25 Задача «Симпатичные таблицы».
26 Задача «Флойд-существование».
27 Задача «Цикл».
28 Задача «Эйлеров цикл».

Библиотека БГУИР

ЛАБОРАТОРНАЯ РАБОТА №7 ОПЕРАЦИИ НАД МНОЖЕСТВАМИ

Цель работы: представить множества в памяти традиционных компьютеров, научиться программированию операций над множествами.

7.1 Теоретические сведения

Множество – простейшая информационная конструкция и математическая структура, позволяющая рассматривать какие-то объекты как целое, связывая их. Объекты, связываемые некоторым множеством, называются элементами этого множества. Если объект связан некоторым множеством, то говорят, что существует вхождение объекта в это множество, а объект принадлежит этому множеству. Допускается неограниченное количество вхождений одного объекта в какое-либо множество. Допускаются множества, каждое из которых имеет только одно вхождение какого-либо единственного элемента этого множества, а также допускается множество, не имеющее вхождений, – пустое множество. Среди множеств выделяют ориентированные и неориентированные множества. Множество может быть задано с помощью механизма, процедуры. Если некоторая процедура дает ответ для любого объекта является он или нет элементом некоторого множества, то такая процедура называется разрешающей процедурой для такого множества. Если же некоторая процедура позволяет получить любой новый элемент некоторого множества, отличный от известных или выданных ранее элементов этого множества, то такая процедура называется порождающей процедурой для такого множества. Неориентированное множество, имеющее малое количество вхождений, может быть представлено в тексте в следующем виде:

$$S = \{a, b, a, a, c\}.$$

Элементы a, b и c принадлежат множеству с именем S, причем множество S имеет три вхождения элемента a ($S/a/ = 3$) и по одному вхождению элементов b и c ($S/b/ = S/c/ = 1$). Неориентированное множество A называют подмножеством неориентированного множества B тогда и только тогда, когда для любого элемента x, который принадлежит множеству A, истинно $A/x/ \leq B/x/$. Если некоторый элемент x не принадлежит множеству S, то истинно $S/x/ = 0$. Если множество A является подмножеством множества B, то это записывают так:

$$A \subseteq B.$$

Неориентированные множества A и B равны тогда и только тогда, когда $B \subseteq A$ и $A \subseteq B$.

Все вхождения, которые имеет любое ориентированное множество, упорядочены. Два ориентированных множества равны тогда и только тогда, когда все их элементы входят в одинаковом порядке. Ориентированное множество может быть представлено в тексте в следующем виде:

$$<a, b, a, c>.$$

Множеством *с кратными* вхождениями элементов называют множество S тогда и только тогда, когда существует x, такой, что истинно $S/x/ > 1$.

Множеством *без кратных* вхождений элементов называют множество S тогда и только тогда, когда для любого x истинно $S/x/ < 2$.

Пересечением неориентированных множеств A и B *с учетом* кратных вхождений элементов будем называть неориентированное множество S тогда и только тогда, когда для любого x истинно $S/x/ = \min\{A/x/, B/x/\}$.

Объединением неориентированных множеств A и B *с учетом* кратных вхождений элементов будем называть неориентированное множество S тогда и только тогда, когда для любого x истинно $S/x/ = \max\{A/x/, B/x/\}$.

Разностью неориентированных множеств A и B *с учетом* кратных вхождений элементов будем называть неориентированное множество S тогда и только тогда, когда для любого x истинно $S/x/ = \max\{A/x/-B/x/, 0\}$.

Симметрической разностью неориентированных множеств A и B *с учетом* кратных вхождений элементов будем называть неориентированное множество S тогда и только тогда, когда для любого x истинно $S/x/ = \max\{A/x/-B/x/, B/x/-A/x/\}$.

Суммой неориентированных множеств A и B элементов называют неориентированное множество S тогда и только тогда, когда для любого x истинно $S/x/ = A/x/+B/x/$.

Пересечением неориентированных множеств A и B *без учета* кратных вхождений элементов будем называть неориентированное множество S тогда и только тогда, когда для любого x истинно $S/x/ = \min\{A/x/, B/x/, 1\}$.

Объединением неориентированных множеств A и B *без учета* кратных вхождений элементов будем называть неориентированное множество S тогда и только тогда, когда для любого x истинно $S/x/ = \min\{\max\{A/x/, B/x/\}, 1\}$.

Разностью неориентированных множеств A и B *без учета* кратных вхождений элементов будем называть неориентированное множество S тогда и только тогда, когда для любого x истинно $S/x/ = \max\{\min\{A/x/, 1\}-\min\{B/x/, 1\}, 0\}$.

Симметрической разностью неориентированных множеств A и B *без учета* кратных вхождений элементов будем называть неориентированное множество S тогда и только тогда, когда для любого x истинно $S/x/ = \max\{\min\{A/x/, 1\}-\min\{B/x/, 1\}, \min\{B/x/, 1\}-\min\{A/x/, 1\}\}$.

Библиотека БГУИР

Булеаном неориентированного множества *A*, которое является множеством без кратных вхождений элементов, называют неориентированное множество *S* тогда и только тогда, когда для любого *x* истинно $S/x/ < 2$ и $((S/x/=1) \Leftrightarrow (x \subseteq A))$.

Декартовым произведением неориентированных множеств *A* и *B* называют неориентированное множество *S* тогда и только тогда, когда для любого *z* истинно: если $S/z/ > 0$, то $z = \langle x, y \rangle$; $S/z/ = A/x/*B/y/$ и наоборот.

Формат данных:

[<имя_множества>=]<множество>

<множество>::=

<ориентированное_множество>|<неориентированное_множество>

<неориентированное_множество>::={[<элемент>{,<элемент>}]}

<ориентированное_множество>::=<<элемент>,<элемент>{,<элемент>}>

<элемент>::=<имя>|<множество>

<имя_множества>::=<имя>

<имя>::=<цифра>|<буква> {<буква>|<подчеркивание>|<цифра>}

<подчеркивание>::=_

<цифра>::=0|…|9

<буква >::=A|…|z

Пример:

A={o,{},A}

B={o,<1,2>,A2,c3,B,b3_A,{},{o,{},A}}

C={o,A2,b3,A,<1,2>,{}}

Пересечением множеств A, B, C будет множество *{o,{},{o,{},A}}.*

Все корректные входные и выходные данные программы должны строго соответствовать четко заданному формату данных. Программа должна давать верный результат для любых корректных входных данных. Для любых некорректных входных данных программа должна сообщать о некорректности входных данных и не должна терять управления в результате ошибки исполнения. Все входные данные должны читаться программой из файла.

Исходный текст программы должен содержать имя, фамилию и отчество разработчика. Там же должны быть указаны дата разработки, описание функциональности и назначения программы. Все функции в исходном тексте программы должны быть прокомментированы. Для каждой функции должны быть указаны и пояснены: назначение, входные данные, выходные данные (в том числе и используемые глобальные переменные). Все глобальные переменные и некоторые локальные переменные должны быть прокомментированы.

Библиотека БГУИР

7.2 Варианты индивидуальных заданий

Вариант выбирается в соответствии с номером студента в рамках группы. Если студентов в группе больше, чем вариантов в списке, то варианты снова повторяются, начиная с единицы.

1 Реализовать программу, формирующую множество, равное объединению произвольного количества исходных множеств (без учета кратных вхождений элементов).

2 Реализовать программу, формирующую множество, равное пересечению произвольного количества исходных множеств (без учета кратных вхождений элементов).

3 Реализовать программу, формирующую множество, равное симметрической разности произвольного количества исходных множеств (без учета кратных вхождений элементов).

4 Реализовать программу, формирующую множество, равное разности двух исходных множеств (без учета кратных вхождений элементов).

5 Реализовать программу, формирующую множество, равное булеану исходного множества.

6 Реализовать программу, формирующую множество, равное декартовому произведению произвольного количества исходных множеств.

7 Реализовать программу, формирующую без повторений всевозможные ориентированные множества из элементов исходного неориентированного множества. Количество элементов в сформированных множествах должно быть равно исходному натуральному n.

8 Реализовать программу, формирующую без повторений всевозможные неориентированные множества из элементов исходного неориентированного множества. Количество элементов в сформированных множествах должно быть равно исходному натуральному n.

9 Реализовать программу, определяющую, является ли одно либо оба из двух исходных множеств подмножеством или элементом другого.

10 Реализовать программу, формирующую множество, равное объединению произвольного количества исходных множеств (с учетом кратных вхождений элементов).

11 Реализовать программу, формирующую множество, равное пересечению произвольного количества исходных множеств (с учетом кратных вхождений элементов).

Библиотека БГУИР

12 Реализовать программу, формирующую множество, равное симметрической разности произвольного количества исходных множеств (с учетом кратных вхождений элементов).
13 Реализовать программу, формирующую множество, равное разности двух исходных множеств (с учетом кратных вхождений элементов).

Библиотека БГУИР

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Новиков, Ф. А. Дискретная математика для программистов / Ф. А. Новиков. – СПб. : Питер, 2000.

2 Седжвик, Р. Фундаментальные алгоритмы на С++. Анализ. Структуры данных. Сортировка. Поиск / Р. Седжвик ; пер. с англ. – М. : ДиаСофт, 2001.

3 Кормен, Т. Алгоритмы: построение и анализ / Т. Кормен [и др.] ; пер. с англ. – 2-е изд. – М. : Издательский дом «Вильямс», 2005.

4 Густокашин, М. С. Лекции по олимпиадному программированию / М. С. Густокашин. – М. : Дистантное обучение, 2007.

5 Долинский, М. С. Решение сложных и олимпиадных задач по программированию : учеб. пособие / М. С. Долинский. – СПб. : Питер, 2006.

6 e-maxx [Электронный ресурс]. – 2014. – Режим доступа : http://e-maxx.ru/.

7 Кузнецов, О. П. Дискретная математика для инженера / О. П. Кузнецов, Г. М. Адельсон-Вельский. – М. : Энергоиздат, 1988.

8 Представление и обработка знаний в графодинамических ассоциативных машинах / В. В. Голенков [и др.] ; под ред. В. В. Голенкова. – Минск : БГУИР, 2001.

9 Голенков, В. В. Семантическая технология проектирования интеллектуальных решателей задач на основе агентно-ориентированного подхода / В. В. Голенков, Д. В. Шункевич, И. Т. Давыденко // Программные системы и вычислительные методы. – 2013. – №1.

10 Базы знаний интеллектуальных систем : учебник / Т. А. Гаврилова [и др.]. – СПб. : Питер, 2001.

11 Окулов, С. М. Программирование в алгоритмах / С. М. Окулов. – М. : Бином, 2004.

Библиотека БГУИР

Св. план 2015, поз. 26

Учебное издание

Голенков Владимир Васильевич
Гулякина Наталья Анатольевна
Давыденко Ирина Тимофеевна
Шункевич Даниил Вячеславович

ТРАДИЦИОННЫЕ И ИНТЕЛЛЕКТУАЛЬНЫЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

ПОСОБИЕ

Редактор *Е. С. Чайковская*
Корректор *Е. И. Герман*
Компьютерная правка, оригинал-макет *Е. Г. Бабичева*

Подписано в печать 09.09.2016. Формат 60х84 1/16. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л. 3,95. Уч.-изд. л. 4,0. Тираж 100 экз. Заказ 152.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий №1/238 от 24.03.2014,
№2/113 от 07.04.2014, №3/615 от 07.04.2014.
ЛП №02330/264 от 14.04.2014.
220013, Минск, П. Бровки, 6

Библиотека БГУИР