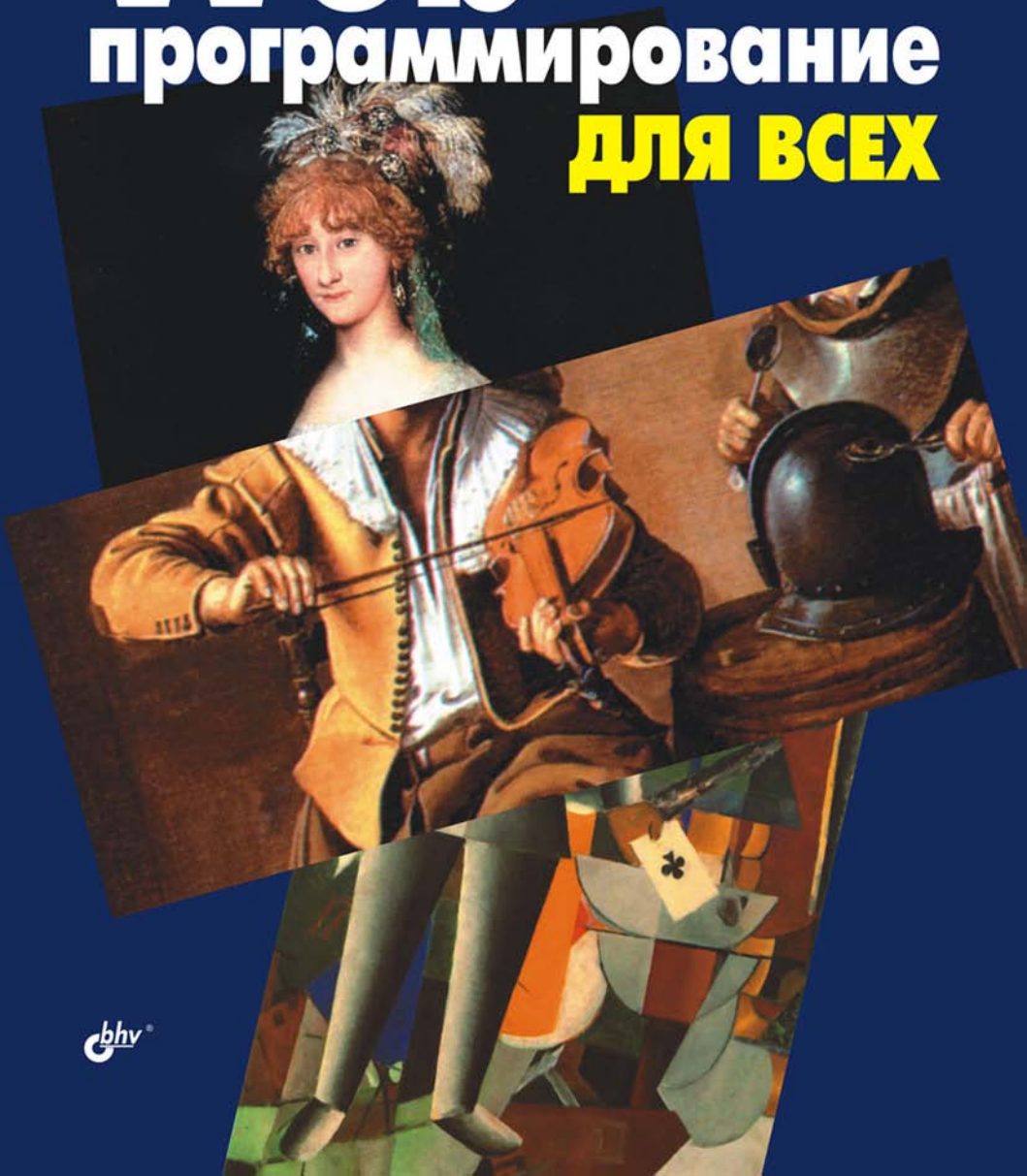
Вадим Дунаев

Web программирование для всех



bhv

Вадим Дунаев

Web программирование для всех

Санкт-Петербург

«БХВ-Петербург»

2008

УДК 681.3.068
ББК 32.973.26-018.1
Д83

Дунаев В. В.

Д83 Web-программирование для всех. — СПб.: БХВ-Петербург, 2008. — 560 с.: ил.

ISBN 978-5-9775-0197-2

В книге в доступной форме в виде диалогов между Простаком, Занудой и Профессором показано, что такое Web-программирование и в каких случаях его необходимо применять. Изложены основы языков JavaScript и PHP. Описано создание клиентских сценариев на JavaScript, начиная с создания новых окон браузера и заканчивая применением технологии AJAX. Рассмотрены серверные сценарии на языке PHP для работы с файлами и папками, взаимодействия с базами данных и многие другие. Материал сопровождается простыми практическими примерами, которые поддерживаются всеми современными браузерами, такими как Internet Explorer, Mozilla Firefox и Opera.

Для Web-разработчиков

УДК 681.3.06
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Леонид Кочин*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Виктория Пиотровская*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 30.04.08.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 45,15.
Тираж 2500 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.002108.02.07 от 28.02.2007 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0197-2

Оглавление

Введение **1**

Пролог **3**

Глава 1. Радости и горести программирования **7**

1.1. Когда Web-странице нужны программы? 7
1.2. С чего и как начать? 10
1.3. Почему программировать интересно? 23
1.4. Неприятности в программировании 24

Глава 2. О языках программирования вообще **31**

2.1. Переменные и типы данных 32
2.2. Массивы данных 39
2.3. Функции 43
2.4. Классы и объекты 49
2.5. Операторы и выражения 57
2.6. Специальные термины 59
2.7. Резюме и напутствие 61

Глава 3. Основы языка JavaScript **65**

3.1. Немного истории о версиях и стандартах 65
3.2. Общая характеристика языка 70
3.3. Как создавать и отлаживать сценарии на JavaScript 71
3.3.1. Вставка сценариев в HTML-документ 72
3.3.2. Подготовка, запуск и отладка сценариев 78
3.4. Ввод и вывод данных 81
3.4.1. Метод *alert — окно предупреждения* 82
3.4.2. Метод *confirm — окно подтверждения* 83
3.4.3. Метод *prompt — окно запроса* 84
3.4.4. Метод *document.write()* 85

3.5. Типы данных ... 86
3.5.1. Примитивные типы данных ... 88
3.5.2. Составные типы данных ... 90
3.5.3. Автоматическое преобразование типов данных ... 92
Преобразование строк (*String*) ... 94
Преобразование чисел (*Number*) ... 95
Преобразование логических значений (*Boolean*) ... 95
Преобразование пустого значения (*null*) ... 95
Преобразование неопределенного значения (*undefined*) ... 95
3.5.4. Принудительное преобразование типов данных ... 96
3.6. Переменные и оператор присваивания ... 100
3.6.1. Имена переменных ... 100
3.6.2. Создание переменных ... 101
3.6.3. Операторы присваивания ... 110
3.6.4. Проверка типа переменной ... 112
3.7. Операторы ... 113
3.7.1. Комментарии ... 113
3.7.2. Арифметические операторы ... 114
3.7.3. Дополнительные операторы присваивания ... 117
3.7.4. Операторы сравнения ... 118
3.7.5. Логические операторы ... 121
3.7.6. Операторы условия ... 122
Оператор *if* ... 123
Оператор условия *?:* ... 126
Оператор *switch* ... 127
3.7.7. Операторы цикла ... 129
Оператор *for* ... 129
Оператор *while* ... 132
Оператор *do-while* ... 134
3.7.8. Об условиях в операторах условия и цикла ... 135
3.7.9. Побитовые операторы ... 135
3.7.10. Другие операторы ... 137
3.7.11. Приоритет операторов ... 137
3.8. Функции ... 139
3.8.1. Встроенные функции ... 140
3.8.2. Пользовательские функции ... 142
3.8.3. Объект *Function* ... 146
3.9. Строки ... 150
3.9.1. Кавычки и специальные символы ... 151
3.9.2. Объект *String* ... 153

3.9.3. Функции вставки и замены подстрок ... 161
3.9.4. Функции удаления ведущих и заключительных пробелов ... 163
3.10. Массивы ... 164
3.10.1. Создание массива ... 165
3.10.2. Многомерные массивы ... 167
3.10.3. Копирование массива ... 168
3.10.4. Объект *Array* ... 169
3.10.5. Функции обработки числовых массивов ... 175
3.11. Числа ... 176
3.11.1. Числа целые и с плавающей точкой ... 176
3.11.2. Объект *Number* ... 180
3.11.3. Объект *Math* ... 182
3.11.4. Функции для решения некоторых математических задач ... 184
Решение квадратного уравнения ... 185
Вычисление интеграла ... 186
Вычисление производной ... 189
Поиск экстремума ... 190
3.12. Дата и время ... 191
3.12.1. Создание объекта *Date* ... 192
3.12.2. Методы объекта *Date* ... 193
3.13. Объекты ... 206
3.13.1. Создание объекта ... 206
3.13.2. Свойства и методы объекта *Object* ... 212
3.13.3. Объектные операторы ... 214
3.14. Операторы обработки исключительных ситуаций ... 217

Глава 4. Клиентские сценарии на JavaScript ... 221

4.1. Об объектной модели браузера и документа ... 221
4.1.1. Общие сведения ... 221
4.1.2. Объект *window* ... 227
Свойства объекта *window* ... 227
Методы объекта *window* ... 228
4.1.3. Объект *document* ... 229
Свойства объекта *document* ... 229
Коллекции объекта *document* ... 231
Методы объекта *document* ... 231
4.1.4. Объект *location* ... 232
Свойства объекта *location* ... 232
Методы объекта *location* ... 232
14.1.5. Объект *history* ... 233
Свойства объекта *history* ... 233
Методы объекта *history* ... 233

4.1.6. Объект *navigator* ... 233
Свойства объекта *navigator* ... 234
Коллекции объекта *navigator* ... 234
Методы объекта *navigator* ... 234
4.1.7. Объект *screen* ... 235
4.2. Доступ к объектам браузера и документа ... 235
4.3. Обработка событий ... 243
4.3.1. Привязка обработчиков событий ... 243
4.3.2. Программный вызов обработчика события ... 247
4.3.3. Изменение поведения элементов по умолчанию ... 249
4.3.4. Прохождение событий ... 250
4.3.5. Информация о событии: объект *event* ... 253
4.4. Окна и фреймы ... 255
4.4.1. Создание новых окон браузера ... 255
4.4.2. Работа с фреймами ... 257
4.4.2. Работа с "плавающими" фреймами ... 263
4.5. Работа с каскадными таблицами стилей ... 263
4.6. Управление во времени ... 269
4.7. Работа с cookie ... 272
4.8. Работа с таблицами ... 278
4.9. Работа с формами ... 281
4.9.1. Проверка данных перед отправкой ... 281
4.9.2. Создание баннера ... 283
4.9.3. Переходы между полями по клавише <Enter> ... 284
4.10. Создание меню ... 286
4.10.1. Меню на основе раскрывающегося списка ... 286
4.10.2. Двухуровневое меню на основе таблиц ... 288
4.11. Перемещение элементов мышью ... 292
4.12. Динамическое изменение содержимого документа ... 297
4.12.1. Изменение свойств, ассоциированных с атрибутами элементов, и свойств стиля ... 298
4.12.2. Предварительная загрузка изображений ... 299
4.12.3. Использование изображения для парольной защиты страницы ... 302
4.12.4. Применение свойства *innerHTML* ... 304
4.12.5. Применение технологии AJAX ... 307
4.13. Распознавание типа браузера ... 317

Глава 5. Основы языка PHP ... 321

5.1. Предварительные сведения ... 322
5.1.1. Где писать сценарии на PHP ... 322
5.1.2. Сообщения об ошибках ... 324

5.1.3. Принудительный выход из сценария ... 324
5.1.4. Справка по PHP ... 325
5.2. Вывод и типы данных ... 325
5.3. Переменные и оператор присваивания ... 330
5.3.1. Имена переменных ... 330
5.3.2. Создание переменных ... 331
5.3.3. Отображение значений переменных ... 333
5.3.4. Переменные переменные ... 337
5.3.5. Область действия переменных ... 338
5.3.6. Проверка существования переменных и их типов ... 340
5.4. Константы ... 341
5.5. Операторы ... 342
5.5.1. Комментарии ... 342
5.5.2. Арифметические операторы ... 343
5.5.3. Строковый оператор ... 345
5.5.4. Дополнительные операторы присваивания ... 345
5.5.5. Операторы сравнения ... 346
5.5.6. Логические операторы ... 347
5.5.7. Побитовые операторы ... 349
5.5.8. Операторы условного перехода ... 349
Оператор *if* ... 349
Оператор *switch* ... 350
Оператор условия *?:* ... 351
5.5.9. Операторы цикла ... 352
Оператор *for* ... 352
Оператор *while* ... 355
Оператор *do-while* ... 356
5.6. Строки ... 357
5.6.1. Двойные и одинарные кавычки ... 357
5.6.2. Склейка строк ... 361
5.6.3. Преобразование строк ... 361
5.6.4. Форматирование строк ... 366
5.7. Числа ... 371
5.7.1. Математические функции ... 371
5.7.2. Математические константы ... 372
5.7.3. Представление чисел в различных системах счисления ... 373
5.7.4. Форматирование чисел ... 375
5.8. Дата и время ... 377
5.9. Массивы ... 381
5.9.1. Создание массива ... 381
5.9.2. Многомерные массивы ... 384

5.9.3. Отображение массивов ... 386
5.9.4. Операции над массивами ... 387
Копирование массивов ... 387
Сортировка массивов ... 387
Перемещение по массиву ... 390
Запись значений элементов массива в переменные ... 393
Преобразование массива в текстовую строку ... 394
Преобразование текстовой строки в массив ... 394
Другие операции над массивами ... 395
5.10. Глобальные предопределенные переменные ... 397
5.11. Функции ... 399
5.11.1. Пользовательские функции ... 399
5.11.2. Переменные функции ... 405
5.11.3. Встроенные функции ... 406
5.11.4. Как узнать, есть ли такая функция ... 406
5.12. Классы и объекты ... 406
5.12.1. Определение класса ... 407
Свойства и методы ... 408
Конструктор ... 409
5.12.2. Применение объектов ... 411
5.12.3. Ограничение доступа к свойствам и методам ... 412
5.12.4. Клонирование и удаление объектов ... 414
5.12.5. Использование методов несозданных объектов ... 415
5.12.6. Обработка исключений ... 416
5.12.7. Пример класса формы ... 417
5.13. Выполнение PHP-кода в текстовых строках ... 419

Глава 6. Основы создания серверных сценариев на PHP ... 421

6.1. Получение данных из форм клиента ... 421
6.1.1. Получение данных из элементов форм ... 422
6.1.2. Передача файлов на сервер ... 430
6.2. Переходы между Web-страницами ... 433
6.2.1. Вывод ссылок ... 434
6.2.2. Применение форм ... 434
6.2.3. Переадресация с помощью функции *header()* ... 435
6.2.4. Добавление информации к URL-адресу ... 436
6.2.5. Применение cookie ... 437
6.2.6. Сеансы ... 439
Создание сеанса ... 439
Особенности сеансов ... 441
Пример организации сеанса ... 442
Защита страниц паролем ... 444

6.3. Работа с графикой ... 447
6.4. Работа с файлами ... 453
6.4.1. Открытие файла ... 454
6.4.2. Закрытие и удаление файлов ... 455
6.4.3. Чтение файла ... 456
Чтение файла в переменную ... 456
Чтение файла в массив ... 457
Чтение файла с удалением тегов HTML ... 457
6.4.4. Запись в файл ... 459
6.4.5. Работа с папками ... 459
6.4.6. Простой счетчик посещений страницы ... 461
6.4.7. Работа с таблицами в текстовых файлах ... 462
Чтение CSV-файла ... 463
Функции для работы с табличными данными ... 465
Сложный счетчик посещений страницы ... 470
Создание баннера ... 475
Создание гостевой книги ... 479
6.5. Работа с базами данных ... 486
6.5.1. Что такое база данных ... 486
6.5.2. Основные средства PHP для взаимодействия с базой данных ... 488
Подключение к базе данных ... 488
Передача запросов к базе данных ... 490
Обработка данных в сценарии ... 491
6.5.3. Создание гостевой книги на основе базы данных ... 492
Создание базы данных ... 493
Создание таблицы для хранения данных ... 493
Определение регистрационного имени и пароля пользователя ... 494
Определение прав пользователя ... 495
Сценарии для взаимодействия с посетителем ... 496
Сценарии для владельца гостевой книги ... 499
6.5.4. Применение SQLite ... 500
6.6. Другие возможности PHP ... 503

ПРИЛОЖЕНИЯ ... 505

Приложение 1. Краткий справочник по HTML и CSS ... 507

П1.1. Теги HTML ... 507
П1.2. Таблицы стилей ... 517
П1.2.1. Единицы измерения ... 517
П1.2.2. Параметры и свойства CSS ... 518
Свойства шрифта ... 518

Свойства цвета и фона ... 521
Свойства списков ... 524
Свойства текста ... 525
Блочные свойства ... 528
Основные свойства печати ... 536
Свойства фильтров ... 537
Свойство *cursor* ... 540

Приложение 2. Установка Web-сервера и PHP ... 541

П2.1. Установка Web-сервера ... 541
П2.2. Установка PHP ... 542
П2.2.1. Установка модуля PHP ... 542
П2.2.2. Настройка модуля PHP ... 543
П2.2.3. Установка расширений PHP ... 545
П2.2.4. Проверка работоспособности Web-сервера и обработчика PHP ... 546

Литература ... 547

Предметный указатель ... 548

Введение

Данная книга посвящена основам программирования клиентских и серверных сценариев для Web-сайтов на языках JavaScript и PHP соответственно. Она рассчитана на широкий круг читателей, впервые столкнувшихся с подобными задачами. Для понимания предлагаемого материала требуется лишь некоторое знакомство с языком HTML для разметки Web-страниц.

Общие темы программирования излагаются в форме диалогов в *главах 1* и *2*. В *главах 3* и *4* рассматриваются язык JavaScript и способы решения основных задач посредством клиентских сценариев. В *главах 5* и *6* рассказывается о языке PHP и создании серверных сценариев. В приложениях даны краткие справочные сведения об HTML и CSS, а также по установке Web-сервера и PHP.

Примеры сценариев на языке JavaScript разработаны таким образом, что выполняются всеми современными браузерами, такими как Internet Explorer 5+, Mozilla Firefox 2+ и Opera 8+.

Из-за ограниченности объема в книге рассматриваются далеко не все задачи создания сценариев и возможности языков JavaScript и PHP. Главная цель книги состоит в том, чтобы ввести читателя в увлекательный мир программирования для Web, и не только. Ваши отзывы и замечания я буду рад принять в своей гостевой книге по адресу **http://dunaevv1.narod.ru**.

Пролог

Простак. Привет, Зануда! Куда это ты спешишь с озабоченным видом?

Зануда. А вот решил обратиться к нашему знакомому Профессору, чтобы он помог мне стать программистом. Я давно ходил кругами около этой темы: разговаривал с друзьями, пробовал читать соответствующие книжки и даже писать кое-какие программки. Однако, признаюсь, кроме разочарования ничего из этого не вышло. Если хочешь, пошли вместе.

Простак. А зачем тебе это? Ты хочешь стать программером, чтобы поступить в какую-нибудь "крутую" программистскую фирму типа Microsoft и зарабатывать много денег?

Зануда. Нет, я пришел к идее заняться программированием, решая обычные задачи, связанные с созданием своего Web-сайта.

Простак. Не понимаю, как сейчас, при современном уровне развития софта можно было озаботиться программированием. Ты, похоже, просто не знаешь о существовании программных систем разработки пользовательских приложений, например, Dreamweaver, FrontPage, Flash, C++-Builder и множества других. Используя эти системы, ничего не придется программировать, достаточно просто нажимать кнопки, соответствующие твоим потребностям разработчика. Все, что получается в результате, появляется на экране компьютера почти в конечном виде, а программный код, заставляющий компьютер все это выполнять, создается и сохраняется автоматически. Ты, конечно, можешь из любопытства посмотреть на него, но зачем?

Зануда. Я знаю, что настоящее взаимодействие с компьютером может осуществить только программист. Чтобы подкрепить это высказывание, попробую воспользоваться следующей аналогией. Так, ты можешь изучать окружающий мир по книгам, туристическим буклетам и рассказам

очевидцев, но вся эта опосредованная информация не способна пробудить в тебе те чувства, которые возникают при непосредственном соприкосновении с явлением, объектом или проблемой, а значит, ты не сможешь мобилизовать свою потенциальную энергию, чтобы решить возникшие задачи. Что бы там мне не рассказывали, я чувствую себя неуверенно или дискомфортно, пока не выясню досконально лично сам, что к чему.

Ты ведь знаешь, что я люблю математику, хотя и не все в ней понимаю достаточно глубоко. Мне кажется, что программирование чем-то похоже на математическое творчество и исследование. Именно эта связь, не вполне ясная для меня, разбудила во мне интерес к программированию. Чтобы программировать, нужно, каждый скажет, знать языки программирования. Я не знаю ни одного из них. Пробовал читать справочники и учебники, но ничего путного из этого не получилось. Понятия плохо запоминаются, а с их применением вообще "труба". Казалось бы, мне все понятно, но мои первые, даже очень примитивные программки, просто не работают. Ситуация похожа на ту, когда я знаю несколько сотен слов иностранного языка, но не могу составить ни одного предложения, которое бы понял иностранец.

Простак. Я тоже не знаю языков, но не горюю об этом. Заметь, что я даже сделал пару сайтов, заказанных мне коммерческими фирмами, и при этом знания языков программирования мне совершенно не понадобились. Я просто воспользовался одним из известных пакетов для визуальной разработки сайтов. Согласен, что иногда приходится применять чужие программки с открытым кодом. Однако в этих случаях можно не разбираться ни в их деталях, ни, тем более, в языках, на которых они написаны. Если ты где-то нашел какую-то чужую программу, и она работает, то все в порядке. В противном случае, если ее исходный код доступен, ты можешь экспериментально, по интуиции, кое-что чуть-чуть изменить, чтобы твой компьютер ее "заглотил" и выполнил без выдачи непонятных для тебя сообщений об ошибках. Почти все так и делают. Если что-то остается непонятным, можно поискать советы на форумах в Интернете. Согласись, в этом есть что-то романтичное или, лучше сказать, "заводное".

Зануда. Итак, как выяснилось, мы оба не знаем языков программирования, но различие между нами в том, что я хочу понять программирование, а ты — нет. Знания, как говорят, плечи не оттягивают.

Простак. Но еще говорят, что от многих знаний много печали. Ладно, я тоже не прочь немного расширить границы своей эрудиции. Поэтому могу составить тебе компанию. Однако предупреждаю, если беседы с Профессором окажутся слишком занудными, то я ретируюсь.

З а н у д а. Хорошо. Насколько я знаю, программирование — очень широкая область деятельности, и его технологии существенно зависят от того, какого рода приложения (программы) требуется создавать. Чтобы не распыляться, предлагаю ограничиться созданием приложений для Интернета, точнее для Всемирной паутины, т. е. для Web. Другими словами, давай сначала познакомимся с программированием Web-приложений. Эта тема актуальна и для новичков, и для более опытных разработчиков. Я хотел бы понять, действительно ли можно взяться за это дело с нулевой предварительной подготовкой и довольно быстро достичь более или менее хороших результатов, или же требуются долгие годы и недюжинные усилия, чтобы стать профессиональным разработчиком Web-приложений. Попутно мне хотелось бы узнать, насколько полученные знания могут пригодиться для разработки программ, непосредственно не связанных с Web.

Глава 1

Радости и горести программирования

Профессор. Уважаемые Зануда и Простак! Что ж, я готов поговорить с вами о программировании, как вообще, так и применительно к задачам создания Web-приложений, в частности. А начну я, пожалуй, с нескольких замечаний о том, почему желательно уметь хотя бы немного программировать, на какие жертвы ради этого предстоит пойти, и какова будет награда. Затем мы рассмотрим, как лучше и быстрее этому научиться.

1.1. Когда Web-странице нужны программы?

Профессор. Вы, должно быть, уже знаете, что Web-страницу или даже многостраничный сайт можно сделать, вообще не прибегая к программированию, т. е. составлению вручную текстов программ на специальных языках. Другими словами, для создания Web-страницы в настоящее время нет прямой необходимости заниматься написанием кодов на таких языках, как HTML, JavaScript и др.

Простак. Да, широко известно, что для этой цели существует не одна система визуальной разработки сайтов, например, Dreamweaver, FrontPage, CofeeCap и др. Об этом я неоднократно твердил Зануде, но он хотел чего-то большего.

Зануда. Однако подобные системы-конструкторы (системы визуальной разработки) автоматически генерируют программные коды на языке HTML, соответствующие нашим манипуляциям мышью и клавиатурой во время "рисования" Web-страницы на экране монитора. Мы можем просмотреть и при желании вручную откорректировать эти коды (если мы знаем язык HTML и JavaScript), а затем увидеть результат на экране. Тем не менее, я слышал, что вполне можно обойтись без этих специальных средств, а просто написать коды HTML и других языков в обычном текстовом редакторе, сохранить их в файле с расширением htm или html, а затем открыть этот файл

в Web-браузере, например, Microsoft Internet Explorer, Opera, Mozilla Firefox и др. Если все правильно, то в результате в окне браузера отобразится соответствующая Web-страница. Впрочем, любой современный браузер предоставляет возможность взглянуть на код загруженной в него страницы. Таким образом, можно избежать использования систем-конструкторов, а делать все вручную, сохраняя полный контроль над своим проектом.

Простак. Конечно, можно и отказаться от технологических оснасток, но зачем? Если что-то можно сделать более наглядным, простым и быстро достижимым с помощью специальных инструментов, то по-чему бы ими не воспользоваться?

Профессор. Вы оба правы. Профессионала от любителей отличает то, что он в работе всегда пользуется наиболее подходящим средством, а в своем арсенале держит только хорошие инструменты. Что-то он делает вручную, "колдуя" над кодами, а что-то — с помощью инструментов визуальной разработки. Тем не менее, мы собрались здесь для обсуждения не того, как и с помощью каких оснасток следует разрабатывать сайты вообще, а как при этом создавать программы, т. е. встраиваемые в HTML-документы сценарии (скрипты). Уточню, говоря о встраиваемых в HTML-код программах, я имею в виду программные коды, написанные не на HTML, а на языках программирования, подобных тем, на которых пишутся настоящие приложения.

Как известно, "несущую конструкцию" Web-страницы обычно создают посредством языка HTML. С помощью HTML в Web-страницу (Web-документ) вставляют обычные тексты, гиперссылки, таблицы, графические изображения и другие элементы, например, звук, видео, Flash-ролики, кнопки, поля ввода данных и т. п. Вместе с тем, в HTML-коде возможны программные вставки, написанные на совсем другом языке программирования, например, JavaScript. Эти вставки, называемые еще сценариями или скриптами (scripts), предназначены главным образом для обеспечения интерактивности Web-страниц — обработки событий, вызванных манипуляциями мышью и клавиатурой, и данных, введенных пользователем. Ведь пользователь может ввести, "тарабаня" по клавиатуре, какие угодно глупости. Кроме того, с помощью сценариев можно изменить часть или все содержимое страницы, идентифицировать браузер пользователя с целью адаптации к нему загружаемого документа, прочитать и записать cookie, чтобы сохранить на пользовательском компьютере какие-то сведения, а также многое другое. Такие задачи, как создание гостевой книги, счетчика количества посещений Web-страницы, форума и электронного магазина, вообще невозможно решить без сценариев. Более того, эти функции выполняют не клиентские, а серверные сценарии.

Так что если вы замахнулись на решение клиент-серверных задач, то вам просто необходимо заняться не только языками, но и технологиями Web-программирования. Если HTML — язык разметки гипертекстовых документов — довольно прост, то язык сценариев несколько сложнее и требует для освоения особого подхода. Нередко HTML вообще не относят к языкам программирования, но это лишь особое мнение программистов, которые привыкли к большему могуществу, чем простое форматирование текста или верстка отображаемого в браузере документа, содержащего помимо обычного текста еще и другие мультимедиавставки.

З а н у д а. А почему HTML — это не язык программирования? Ведь и в нем есть команды, пусть даже и называемые тегами или инструкциями, а также параметры команд — атрибуты. Такие категории присущи всем языкам программирования. Разумеется, не все, что хочешь, можно сделать с помощью HTML, но и другие языки также имеют какие-то ограничения.

П р о ф е с с о р. Наверное из-за того, что программы, написанные на HTML, имеют очень простую (линейную) структуру. Интерпретатор HTML, т. е. Web-браузер, выполняет загружаемый HTML-код последовательно, команду за командой, как они написаны в исходном коде. Иногда, возможно, браузер делает это несколько иначе, но автор HTML-программы все равно может считать, что ее выполнение происходит именно так, и в подавляющем большинстве случаев будет действительно прав. В языке HTML нет ни переменных для хранения в оперативной памяти компьютера промежуточных и окончательных значений, ни команд управления вычислительным процессом, ни многого другого, что свойственно большинству настоящих языков программирования, хотя в нем много команд и параметров. Я бы сказал, что язык HTML лексически (словарно) богат, а средствами выражения (синтаксическими структурами) беден. Но нет худа без добра. Именно незатейливость HTML-программ в структурном отношении и обеспечила всеобщую доступность программирования на языке HTML. Представьте себе очень примитивного человека. Его словарный запас не более 200 слов. Однако мы, обладая запасом более 2000 слов, и он, весьма стесненный в средствах выражения, все же сохраняем способность общения друг с другом. Это происходит из-за того, что мы можем построить свои тексты в примитивно-линейном порядке, сократив объем удерживаемого контекста до минимума (избегая ссылок на ранее сказанное), а также не используя логические конструкции вида "если … то" и т. п. Другими словами, мы сможем кое-как общаться на основе весьма примитивного языка. Но если нам потребуется более изощренная коммуникация, лучше приспособленная для выражения нетривиальных мыслей, то понадобится и более сложный язык, более гибкий не столько лексически, сколько грамматически. Поэт использует обыкновенную

лексическую базу и почти стандартную грамматику языка, а поэтического эффекта достигает игрой лексических подстановок (метафорами) в стандартные синтаксические конструкции. Если этого мало, то он модифицирует и грамматические шаблоны, достигая мелодической и ритмической выразительности своей рифмованной речи. Я не поэт и не литературовед. Сказанное мною предназначено лишь для того, чтобы посредством аналогии акцентировать ваше внимание на некоторых важных языковых аспектах программирования.

Не следует забывать, что HTML (Hyper Text Markup Language — язык разметки гипертекстовых документов) был задуман как язык форматирования текста со ссылками на другие документы. Вслед за этим стало очевидным, что гипертекстовый документ должен иметь возможность включать в себя не только тексты, но и внешние ресурсы другого сорта, такие как графические изображения, видео, звук и т. д. Идея гипертекстов была предложена Ванневаром Бушем в 1945 году, т. е. на заре компьютерной техники, но для широкого внедрения этой концепции в жизнь потребовалось более 40 лет. Сначала всего этого было более чем достаточно, а простота HTML делала его доступным широким массам, желающим что-либо опубликовать во Всемирной сети. Позднее стало очевидным, что кроме разметки публикуемого текстового документа и вставки в него чего-то инородного, требуется делать еще кое-что, по крайней мере, организовывать каким-то особым образом взаимодействие с сервером. Прежде чем отправить на сервер какие-то данные, введенные пользователем в поля формы, настоятельно требовалось их проверить. Действительно, если пользователь ничего не ввел и нажал на кнопку **Отправить**, то зачем передавать "пустые" данные? Эти первоочередные задачи и потребовали подключения к HTML-коду программ обработки данных, способных не только их визуально представлять, но и анализировать, а при необходимости и преобразовывать.

1.2. С чего и как начать?

Простак. Насколько я правильно Вас понял, уважаемый Профессор, настоящие программы пишутся на более сложных, чем HTML, языках. Не соприкасаясь вплотную с программированием, я это и так понимал, а потому всячески сторонился такой деятельности. Я считал, что программирование, будучи сродни математике и другим точным наукам, чуждо мне. Оно кажется мне сухим и весьма занудным делом, в котором требуется точное соблюдение предопределенных правил, заучивание специальных слов и особенностей их употребления, а также понимание многих нюансов со-

ставления программ Я же люблю быстро получать ощутимые результаты, а не копаться во всяких там "крючках".

З а н у д а. А я не знаю, получится из меня специалист или нет, но, тем не менее, хотел бы поближе познакомиться с тем, чем программисты занимаются, в чем суть их проблем и как они пытаются их разрешить. Если я не ошибаюсь, то к освоению программирования следует приступить, начав с изучения какого-нибудь конкретного языка. При этом правила языка лучше осваивать на примерах и выполняя контрольные задания. Верно?

П р о ф е с с о р. Чтобы заниматься практическим программированием нужно, разумеется, изучить язык, на котором вы будете писать свои программы. Кроме этого, необходимо знать еще и среду программирования — операционную систему, под управлением которой будет выполняться ваша программа, имеющиеся и подключаемые при необходимости уже готовые библиотеки функций, а также возможности конкретной исполнительной системы более низкого уровня, такой, например, как интернет-браузер. Исполнительная система предоставляет вам доступ ко многим объектам: своим собственным элементам, загруженным документам, к конструкциям собственно языка и, возможно, операционной системы. Все это довольно сложное для понимания "хозяйство", однако его постепенно осваивают даже люди изначально малоподготовленные, но искренне желающие стать программистами.

Бытует мнение, что для понимания программирования непременно требуется освоение конкретного языка. Однако я хочу сказать, что суть умений программиста, как и обычного писателя, составляет отнюдь не доскональное знание языка. Язык лишь необходимое формообразующее средство, сам по себе он не обеспечивает автору достаточные условия создания хорошего произведения — конечного продукта его творчества.

П р о с т а к. Вот уж никак не ожидал услышать, что для программирования изучение языка не является необходимым. А как же я буду писать программы? Ведь я могу такое написать, что не только компьютер, но и Вы, Профессор, не поймете.

П р о ф е с с о р. Попробуем разобраться в этом кажущемся парадоксе. Обратите внимание, что дети примерно к пяти годам почти в совершенстве пользуются родным языком в устной речи, не имея при этом никакого понятия о его теории: орфографии, морфологии, синтаксисе и т. д. Дошкольники учатся говорить в процессе коммуникации, на практических примерах, а не по учебникам. Представьте себе, что бы получилось, если обучение языку годовалого ребенка начиналось с разъяснения того, что является именем существительным, а что — глаголом. Дети учатся примитивному словоупотреблению, начиная с называния окружающих предметов и постепенно

переходя к несложным предложениям, отображающим отношения между видимыми и уже названными предметами. Эти навыки формируются у детей удивительно быстро и как будто без нашего, взрослого сознательного вмешательства. Во всяком случае, наше взрослое вмешательство кажется нам лишь корректирующим, а не креативным, т. е. принудительно формирующим. Эффективность данного процесса особенно высока, если родители, бабушки и дедушки не используют в своей речи "птичий язык", т. е. сленг, обильно приправленный междометиями и словами-паразитами. Настоящие проблемы у детей возникают при изучении уже практически освоенного устного языка в школе, где требуется не только знать правила правописания, но и следовать им в письменной речи. И сложность не только в переходе от устной речи к письменной. Трудность в правильном использовании доставшегося нам от предков родного языка вообще, независимо от целей его применения и среды. Когда я слышу фразу, начинающуюся со слова "короче", то подозреваю, что краткости изложения мысли мне не стоит ждать, несмотря на декларированную претензию говорящего быть кратким. Желающие высказаться сжато (нередко обозначающие это стремление вводным словом "короче"), в действительности выражают лишь свое желание выразить суть, но не могут (по разным причинам), а потому за подобным "ключевым" словом следует весьма длительное и невразумительное словоблудие. А в чем причина? Трудно дать исчерпывающий ответ, но на некоторые вопиющие, грубые ошибки я легко могу указать. Прежде всего, речь должна следовать за мыслью, а не наоборот. Если у вас это не всегда получается, то просто не спешите говорить (не тараторьте), заполняя требуемые процессом мышления паузы чем попало. Не забывайте при этом, что высказанное утверждение имеет значительно более далеко идущие последствия, чем мы обычно это имеем в виду, если действительно об этом задумываемся. Недаром говорят, что слово — не воробей, вылетит — не поймаешь. Мысли материализуются, даже если вы их не записали ни в обычном письме, ни в программном коде. Помните, что "сначала было слово" и только потом все остальное — реализации и видимые представления. Программист, пишущий свою речь на искусственном языке для компьютера, должен быть точен в выражении своих мыслей. Возможные ошибки не только искажают мысль, но и прерывают само общение.

Создатель художественных произведений стремится представить свою мысль в виде образов, атакующих эмоциональную сферу, т. е. неточных и расплывчатых изначально (по своему определению). Его задача заключается не в том, чтобы научить кого-то и чему-то конкретно, а в том, чтобы возбудить в реципиенте некое чувство, которое поможет ему что-то осознать или даже сотворить. Пусть, как думает автор, читатель или зритель произведения сам сотворит свой мир на заданную тему. Программист, однако, не может огра-

ничиться только этими целями и технологиями, он должен донести свою идею и сопутствующие ей образы сначала до "холодного" компьютера, а через него и "одухотворенному" конечному пользователю.

Общаясь друг с другом, мы часто даем указания или команды, высказываем какие-то требования и пожелания. Особенно мы расположены к поучениям в виде советов, хотя их выдача — большой грех, даже при искреннем желании помочь ближнему. Это обусловлено тем, что рекомендации "благожелателей" ограничивают, так или иначе, ваше личное движение по собственному пути, т. е. вашу творческую активность, которую вы просто обязаны проявить в отпущенный вам срок. А советчики хотя и готовы помочь, но помните, что "благими намерениями вымощена дорога в ад". Поэтому советы следует давать лишь тем, кто их просит (не следует отказывать в милосердии тем, кто стучит в вашу дверь). Вместе с тем, преподавательская деятельность в различных формах (занятия в учебных заведениях, учебники и всяческие пособия) вполне оправдана в человеческом общежитейском плане тем, что разумно передать накопленный жизненный опыт, по крайней мере, следующему поколению. Однако преподаватель не должен ничего навязывать своим ученикам в качестве истины в последней инстанции. Доступные нам здесь и сейчас знания эфемерны и неоднозначны, а подлинная, вечная и незыблемая истина все время впереди нас, светит нам подобно маяку, указывающему лишь направление движения, вселяя в нас надежду когда-нибудь пристать к берегу. Но так ли нам важен вожделенный берег? А что будет потом, после достижения цели? Это сакраментальный вопрос, на который мало кто отваживался ответить серьезно, а полученные ответы в итоге оказались неудовлетворительными. Я для себя понял лишь, что продвижение к чему-то, что называют истиной, важнее ее самой, быть может недоступной изначально в нашем мире. Этот процесс и составляет, по моему мнению, содержание и смысл жизни.

Итак, мы должны быть готовы к преодолению всех возможных препятствий на пути к истинному знанию, пусть даже без надежды когда-нибудь пристать к берегу. Никто из смертных не обладает и не может постичь истину абсолютно, но отказ от стремления к ней, подобно самоубийству, лишает содержания и смысла саму жизнь. Жизнь — плавание, а не отстой у причала. С другой стороны, ученик не должен оправдывать свои неудачи тем, что кто-то его так-то и тому-то научил, в результате чего он заблудился, напортачил или просто получил двойку на экзамене. Учение — дело не столько преподавателя, сколько самого ученика, который должен сам искать и выбирать, чему, как и у кого учиться. Я в этом убежден, что бы там ни говорила официальная пропаганда и площадная молва об устройстве системы нашего образования.

Простак. Но, уважаемый Профессор, учителя в школах обязаны дать базовые знания своим ученикам по основным областям знаний: родному языку, истории, географии, математике, физике и т. д. Они должны приобщить детей к сложившейся культуре своего народа, познакомить с другими цивилизациями и, наконец, сообщить основные сведения о достижениях современной общечеловеческой науки и технологии. Ребенок, согласитесь, не может самостоятельно понять и сформулировать, из какой области знаний ему нужно получить какую-то конкретную информацию. Кроме того, ни ребенок, ни его родители не могут подчас определить, хороши ли учителя. Все это так сложно.

Профессор. Общий рецепт прост. Если вам не нравится учитель, перейдите к другому. Смешно требовать от учителя научить чему-то, особенно в том случае, если ученик не хочет учиться.

Зануда. Большинство детей, насколько я знаю, не очень-то любят учиться. Но с высшей точки зрения (государства или нации) мы же должны как-то позаботиться об уровне образования следующего поколения. Ведь мы не можем в этом важном деле пустить все на самотек или положиться на природу или Провидение. Бог не любит, если мы опускаем руки.

Простак. Как же не требовать от учителя, если мы отдаем своих детей в школу, чтобы их там научили всему, что надо. Я хотел бы, чтобы мой ребенок закончил школу в том состоянии, которое позволило бы ему продолжить образование в университете, например. Если не быть требовательным к учителям, то они просто спишут все свои огрехи на нерадивость учеников, а их успехи, если такое произойдет, непременно отнесут к своим заслугам.

Профессор. Не будем излишне драматизировать ситуацию. Дети обладают различными от природы способностями и интересами. От преподавателя можно ожидать в лучшем случае не более того, что он сам знает и умеет. Разумеется, школьные учителя отличаются от специалистов-предметников (пусть даже нобелевских лауреатов) тем, что они владеют методикой обучения, т. е. технологией освоения знаний. Все это прекрасно, но если учитель не имел собственного опыта научных исследований, то никакая изощренная методика изложения знаний не поможет их передать. Без исследований нет и откровенных, т. е. настоящих знаний. Ученики рано или поздно распознают, что их учитель просто цитирует не им написанный учебник, подчас интерпретируя изложенные в нем истины весьма убого, скучно и некреативно, а в предмете ориентируется лишь формально. Детей трудно обмануть, поскольку их вводит в жизнь в первую очередь природа, а только после этого — человеческих рук культура. Детская способность отличить фальшь

от правды значительно сильнее, чем у нас, взрослых, наученных жизнью терпимости и беспринципности.

Вы говорите, что учитель должен дать основные знания по фундаментальным наукам и искусствам, которые были сформированы человечеством в течение нескольких веков, если не тысячелетий? Вы полагаете, что это возможно? Преподаватель может лишь поведать своим ученикам, как лично он, а также многочисленные предшествующие поколения пробирались сквозь тернии к истине, как она (истина) трансформировалась с годами и под влиянием ее творцов — героев-исследователей, — как удавалось в этом процессе отделить субъективное (личное и эфемерное) от объективного (не зависящего от испытателя и перманентно воспроизводящегося), и наконец, каково текущее состояние изучаемой предметной области.

Аналогичные проблемы возникают и при попытках изучения искусственных языков, созданных человеком для определенных целей и ориентированных на работу с компьютерами, т. е. при изучении языков программирования.

П р о с т а к. Уважаемый Профессор, Вы намекаете на то, что, следуя великой и мудрой природе, не нужно загружать учеников громадой знаний о языках, а лучше рассмотреть хорошие практические примеры. При этом наиболее трудные для понимания места Вы, как мастер, можете прокомментировать, предостеречь от грубых ошибок и прочих глупостей, обычно свойственным новичкам. Я знаю, что большинство как начинающих, так и опытных разработчиков программ больше всего ценят не общие разглагольствования, а профессиональные советы, способствующие решению вполне конкретных задач. Мне кажется, что нам с Вашей, разумеется, помощью следует поставить ряд актуальных задач, а Вы затем дадите развернутые рекомендации для их решения.

П р о ф е с с о р. Предлагаемый Вами подход — лишь один из возможных вариантов. Изучение типичных примеров, а также конкретных приемов и рецептов — хороший способ подготовки исполнителей низшего звена. Таков нормальный путь обучения ремеслам, но не наукам. Освоение моделей различных областей мира, технологий их изучения и проектирования, способов применения теории к жизни — задача более высоких ступеней образования. Ремесло живо лишь в течение довольно короткого периода, пока конъюнктурно востребовано и использует конкретные и доступные в данный момент инструменты и технологические приемы. Более высокий уровень образования востребован значительно дольше, именно он постигается в высшей школе, хотя нередко в число ее дисциплин (я думаю, из меркантильных соображений) вводят прикладные курсы, не имеющие под собой никакого научного фундамента. Изучение, например, способов замены электрических

предохранителей практически очень важно, но не является задачей курса электротехники в высшем учебном заведении. Главное в том, что получившие более высокое образование имеют бесценную способность наращивать свои первоначальные знания. Иначе говоря, они знают, как учиться, изучать уже известное и исследовать еще непознанное.

Правильный подход к обучению во многом определяется степенью предварительной подготовленности учеников, а также целями, которые они перед собой ставят. Для выбора эффективного способа изучения программирования мне, как вашему преподавателю, очень важно знать, имел ли ученик предварительное знакомство с предметом изучения или нет. Это необходимо для построения оптимального плана обучения не только программированию, но и вообще всем наукам и ремеслам. Можно выделить три уровня знакомства с программированием: нулевой, предварительный и профессиональный.

Само собой разумеется, нулевой уровень соответствует полному отсутствию каких бы то ни было знаний, которые применимы на практике или, на худой конец, в "светских" разговорах о компьютерах. Человек с данным уровнем подготовленности либо шарахается от программирования, как черт от ладана, либо, ведомый природными любопытством и целеустремленностью, затратив некоторые усилия, переходит на следующую ступень.

Предварительный уровень предполагает некоторое, обычно весьма поверхностное, знакомство, по крайней мере, с основными терминами и понятиями, такими как переменная, оператор, функция, объект, тип данных и т. п., которое все же остается неточным и неполным настолько, что не позволяет уверенно решать даже относительно простые задачи, не говоря уж об эффективности принятых решений. Представители данного уровня стремятся найти и вставить в свои конструкции уже готовые коды программ. При этом они пытаются их как-то изменить, приспособив к своим конкретным задачам. Иногда это удается, но чаще вызывает неудовлетворенность из-за множества сообщений об ошибках. В поисках помощи такие люди мечутся по просторам Интернета "без руля и ветрил", обретая дурные привычки или окончательно отвращаясь от программистских амбиций. Подобным ученикам для дальнейшего продвижения в области программирования нужен хороший учебник или руководитель.

Профессиональный уровень может иметь несколько градаций. Однако для него характерно владение, по меньшей мере, одним языком и одной технологией программирования, а также опыт участия в реализации хотя бы одного конкретного проекта. Профессионал довольно легко переходит от одного языка к другому и приспосабливается в случае необходимости к соответствующей этому языку технологии. Владея одним языком программирования,

он уже приблизительно знает, с чем ему придется встретиться при изучении другого. Если он уже овладел одной технологией программирования, то при изучении другой он понимает, чем она хороша или плоха при решении задач интересующего его класса. Профессионала обычно отличает то, что он знает, с какой стороны подойти к проекту даже незнакомой ему предметной области, как его провести через все перипетии реализации и чем закончить. И это знание является, пожалуй, важнейшим среди всей информации, которой он владеет. Тем не менее, профессионал все время читает новые публикации не только на узкую специальную тему, но и о сопредельных областях. Он всегда "держит руку на пульсе" и обеспокоен тем, чтобы не упустить из виду современные тенденции развития его науки и ремесла. Профессионалам обычно хватает справочников или кратких руководств по новому языку, чтобы с минимальными усилиями перейти к практической деятельности. Программисты обычно держат в тайне освоенные приемы работы и полученные знания. Чаще всего они не могут поведать о своем мастерстве не только из-за предосторожности разбазарить свое "ноу-хау", но и просто органически: они не знают с чего начать и чем закончить в своем изложении. Возможно поэтому всем нам, обычным смертным, программисты представляются в облике "гуру", слова и звуки которых следует ловить, боясь что-либо пропустить, а затем внимательно, не считаясь с трудозатратами, пытаться как-то интерпретировать с пользой для решения своих меркантильных задачек. Чаще всего гуру — это просто очень опытный программист, сталкивавшийся в своей жизни и с разными задачами, и с различными приемами их решения. Реже встречаются специалисты, которые не только могут решать кем-то поставленные задачи, но и ставить (формулировать) их с учетом известных технологических средств.

На заре компьютерной эры программисты в большинстве своем были математиками. До середины 1980-х годов их так и называли. И недаром, поскольку они умели представить научные, инженерные и другие задачи в форме алгоритмов их решения, а затем и в виде программ для компьютера. Тех программистов, кто не обладал алгоритмическим способом мышления и не мог трансформировать постановку задачи в программу для ее решения, называли кодировщиками. Кодировщики досконально знали язык программирования или системы машинных команд, но не владели способами решения пользовательских задач. А алгоритмы решения прикладных задач в большинстве случаев следовало искать не в области программирования, а в математике. Нередко возникали вполне осмысленные и практически востребованные ситуации, которые, как казалось на первый взгляд, можно было легко решить с помощью компьютера. Малообразованный программист с радостью хватался за выгодный заказ и с энтузиазмом начинал писать соответствую-

щую программу. Затем возникали проблемы, связанные не только с ограниченными ресурсами компьютера. Программист долго мучился в поисках выхода из создавшейся затруднительной ситуации и нередко находил его путем изменения постановки самой задачи. Данный процесс мог иметь не одну безуспешную итерацию. А дело порой оказывалось в том, что задача в первоначальной своей постановке оказывалась алгоритмически неразрешимой. Иначе говоря, было принципиально невозможно построить алгоритм (а значит и программу), который приводил бы к решению. В таких случаях говорят, что задача алгоритмически неразрешима. А как распознать, разрешима ли алгоритмически данная задача? Для этого необходимо выполнить некоторое исследование, которое чаще всего лежит в области математики, а не программирования.

З а н у д а. Если я Вас правильно понял, существуют задачи, которые нельзя решить в принципе, даже с использованием суперкомпьютеров. Было бы неплохо, если бы Вы привели какой-нибудь пример для иллюстрации этого парадоксального обстоятельства.

П р о ф е с с о р. В качестве примера рассмотрим практическую задачу обмена валют, так или иначе решаемую с помощью программного обеспечения компьютера в обменном пункте. Валюты, как хорошо известно, имеют различные наборы денежных купюр. Например, существуют купюры (бумажные деньги) достоинством 5, 10, 50, 100, 500, 1000 и 5000 рублей, но нет купюр достоинством 3 и 4 рубля. Другие валюты могут иметь отличающиеся наборы купюр. Кроме того, в пункте обмена в данный момент может не оказаться некоторых из принципиально существующих купюр, а количество имеющихся очень ограничено. Клиент желает обменять по курсу некоторую сумму, например, в рублях на деньги в другой валюте. Очевидно, что возможна ситуация, когда предъявленная сумма не может быть полностью переведена в другую валюту из-за ограниченного набора купюр в обменном пункте. Чтобы все же выполнить операцию обмена, клиент либо должен согласиться получить сдачу в рублях, либо добавить некоторую сумму в рублях к предъявленной для обмена. Естественно желать, чтобы эта дополнительная сумма сдачи или добавки была как можно меньше. Можно потребовать, чтобы эта сумма была минимальной. Именно в такой постановке задача может заинтересовать математиков.

З а н у д а. Очевидно, что сначала нужно с помощью коэффициента, выражающего переводной курс валют, вычислить сумму, которую обменный пункт должен, в первом приближении, отдать клиенту.

П р о с т а к. За вычетом комиссионных, разумеется.

З а н у д а. Да, но сейчас это не столь важно, потому не будем учитывать комиссионные. Далее следует подобрать купюры так, чтобы сложившаяся сумма оказалась как можно ближе к вычисленной по курсу.

П р о с т а к. Лучше сказать, чтобы разница между вычисленным значением и суммой номиналов купюр была минимальной. Поскольку, вообще говоря, одну и ту же сумму можно составить с помощью различных комбинаций купюр, то придется рассматривать множество вариантов, чтобы выбрать наилучший в смысле минимальности сдачи (или добавки).

П р о ф е с с о р. Должен заметить, что ни один обменный пункт не решает задачу в такой постановке. Я имею в виду требование минимальности. Дело в том, что, как показали специальные математические исследования, длина пути поиска такого решения (т. е. число перебираемых вариантов) слишком велика.

П р о с т а к. В обменные пункты следует просто поставить более производительные компьютеры. Наконец, то, что нам недоступно сейчас, станет обычным завтра.

П р о ф е с с о р. К сожалению, загвоздка имеет принципиальный, а не технологический характер. Дело в том, что рассматриваемая задача с требованием минимальности расхождения вычисленной и составленной из купюр сумм имеет, как говорят специалисты, экспоненциальную сложность. Это значит, что сложность (например, число шагов или время вычисления) зависит экспоненциально от объема исходных данных. Так, если n — объем исходных данных, то сложность экспоненциально сложных задач определяется как e^n (где $e \approx 2{,}72$ — основание натуральных логарифмов), т. е. очень быстро возрастает с увеличением n. Например, при $n = 50$ значение e^n превосходит количество атомов в видимой части вселенной. Только перечисление их с помощью самого быстродействующего компьютера потребует практически неприемлемо большого времени. Относительно рассматриваемой задачи доказано, что она разрешима и, более того, найден алгоритм ее решения. Вместе с тем, доказано, что эта задача имеет экспоненциальную сложность и, следовательно, соответствующий алгоритм в интересных с точки зрения практики случаях нереализуем. При очень малых объемах данных его вполне можно реализовать и достаточно быстро получить результаты, но для пользователя это может быть не интересно. Таким образом, существуют теоретически разрешимые задачи с практически нереализуемыми алгоритмами.

П р о с т а к. Сказанное Вами, Профессор, звучит парадоксально. Но ведь формулировка задачи для пункта обмена валют выглядит вполне разумно с точки зрения потенциального пользователя, не говоря уж о ее