فصل نوزدهم

برنامه‌نویسی وب

اهداف

- پروتکل CGI.
- سرآیندهای HTTP و HTT.
- عملکرد سرویس‌دهنده وب.
- سرویس‌دهنده Apache HTTP.
- تقاضای مستندات از سرویس‌دهنده وب.
- پیاده‌سازی اسکریپت‌های CGI.
- ارسال ورودی به اسکریپت‌های CGI توسط فرم‌های XHTML.

رئوس مطالب

۱-۱۹ مقدمه

۲-۱۹ انواع تقاضاهای HTTP

۱۹-۳ معماری چند گره‌ای
۱۹-٤ دسترسی به سرویس‌دهنده‌های وب
۱۹-٥ سرویس‌دهنده Apache HTTP
۱۹-٦ تقاضای مستندات XHTML
۱۹-۷ معرفی CGI
۱۹-۸ تراکنش ساده HTTP
۱۹-۹ اسکریپت‌های ساده CGI
۱۹-۱۰ ارسال ورودی به اسکریپت CGI
۱۹-۱۱ استفاده از فرم‌های XHTML برای ارسال ورودی
۱۹-۱۲ سرآیندهای دیگر
۱۹-۱۳ مبحث آموزشی: صفحه وب تعاملی
۱۹-۱٤ کوکی
۱۹-۱٥ فایل‌های طرف سرویس‌دهنده
۱۹-۱٦ مبحث آموزشی: کارت خرید

۱۹-۱ مقدمه

با ظهور و ورود *وب گسترده جهانی* (WWW). اینترنت محبوبیت باور نکردنی پیدا کرد. این حجم عظیم تقاضای کاربران برای بدست آوردن اطلاعات به سمت وب سایت‌ها هدایت گردید. آشکار شد که میزان تعامل مابین کاربر و وب سایت از اهمیت خاصی برخوردار است. قدرت وب نه تنها در ارائه مطالب به کاربران است، بلکه در واکنش و پاسخ دادن به تقاضاهای کاربران نیز می‌باشد و از اینرو محتویات وب حالت دینامیکی دارند.

در این فصل، در ارتباط با نرم‌افزار خاصی بنام *سرویس‌دهنده وب* (web server) صحبت خواهیم کرد که به تقاضای *سرویس‌گیرنده* (*client*)، مثلاً مرورگر وب، با فراهم آوردن منابع (مثلاً مستندات XHTML) و نمایش آنها برای سرویس‌گیرنده، ازخود واکنش نشان می‌دهد. برای مثال زمانیکه کاربر یک آدرس URL(Uniform Resource Locator) همانند www.detial.com را در یک مرورگر وب وارد می‌کند، کاربر مستند خاصی را از یک سرویس‌دهنده وب تقاضا می‌کند. سرویس‌دهنده وب مبادرت به نگاشت URL با فایلی بر روی سرویس‌دهنده (یا فایلی بر روی شبکه سرویس‌دهنده) کرده و مستند مورد تقاضا را به سرویس‌گیرنده برگشت می‌دهد. در مدت زمان این تعامل، سرویس‌دهنده وب و سرویس‌گیرنده از طریق پروتکل HyperText Transfer Protocol (HTTP)، که یک پروتکل مستقل از پلات‌فرم (platform) است با یکدیگر در ارتباط بوده و با استفاده از این پروتکل مبادرت به انتقال تقاضاها و فایلها بر روی اینترنت می‌کنند (یعنی مابین سرویس‌دهنده‌های وب و مرورگرهای وب).

سرویس‌دهنده وبی که در اینجا معرفی می‌کنیم سرویس‌دهنده Apache HTTP است. برای نشان دادن نتایج کار را از مرورگر وب Internet Explorer شرکت Microsoft استفاده کرده‌ایم.

۱۹-۲ انواع تقاضاهای HTTP

پروتکل HTTP در برگیرنده چندین نوع از تقاضا است که با نام متدهای تقاضا شناخته می‌شوند. هر کدامیک نحوه درخواست سرویس‌گیرنده از سرویس‌دهنده را مشخص می‌کنند. دو متد متداول در این زمینه عبارتند از get و post. این تقاضاها مبادرت به بازیابی و ارسال داده به سرویس‌گیرنده و از سرویس‌دهنده وب به سرویس‌گیرنده می‌کنند. فرم یک عنصر XHTML بوده و می‌تواند حاوی فیلدهای متنی، دکمه‌های رادیوئی، جعبه‌چک‌ها و سایر کامپونت‌های گرافیکی واسط کاربر (GUI) باشد که به کاربر امکان می‌دهد تا داده‌های خود را وارد یک صفحه وب نماید. البته فرم‌ها می‌توانند در برگیرنده فیلدهای پنهان هم باشند. از یک تقاضای get برای ارسال داده به سرویس‌دهنده استفاده می‌شود. همچنین از post برای ارسال داده به سرویس‌دهنده استفاده می‌شود. متد get ، داده را بعنوان بخشی از URL ارسال می‌کند، همانند `www.searchsomething.com/search?query=`*`userquery`* که حاوی ورودی کاربر است. برای مثال، اگر کاربر جستجوی بر روی "Massachusetts" انجام دهد، بخش انتهائی URL عبارت `?query=Massachusetts` خواهد بود. در حالیکه متد *get* رشته پرس‌وجو (*query string*) را با تعدادی از کارکترهای از پیش تعریف شده محدود می‌کند(رشته پرس‌وجو در این مثال `query=Massachusetts`است). این محدودیت از سرویس‌دهنده‌ای به سرویس‌دهنده دیگر، مختلف است. اگر طول رشته پرس‌وجو از این محدوده تجاوز نماید، بایستی از یک متد *post* استفاده کرد.

مهندسی نرم‌افزار

*داده ارسالی توسط متد **post** جزئی از **URL** نبوده و توسط کاربر قابل رویت نمی‌باشد. غالبا فرم‌های که باید برخی از فیلدهای آن مورد تایید قرار گیرند از متد **post** استفاده می‌کنند.فیلدهای حساسی همانند کلمه رمز از این روش برای ارسال داده استفاده می‌کنند.*

غالبا یک تقاضای HTTP داده را به سمت یک سرویس‌دهنده و از طریق یک *دستگیره* (*handler*) که پردازش داده‌ها را انجام می‌دهد، ارسال می‌کند.

در اکثر مواقع مرورگرها مبادرت به ذخیره کردن صفحات وب بر روی دیسک محلی(*cache*) برای بار شدن سریعتر صفحات می‌کنند تا از میزان داده‌های که نیاز است تا مرورگر آنها را برداشت نماید، کاسته شود. با این وجود، در حالت کلی مرورگرها مبادرت به ذخیره کردن پاسخ‌های داده شده به

تقاضاهای post نمی‌کنند، چراکه امکان دارد تقاضاهای post بعدی حاوی همان اطلاعات قبلی نباشند. از اینرو در اکثر مواقع مرورگرهای وب پاسخ تقاضاهای *get* را ذخیره می‌کنند. جدول شکل ۱-۱۹ لیستی از انواع تقاضاها را بجز get و post ارائه کرده است. این متدها استفاده زیادی ندارند.

نوع تقاضا	توضیح
delete	همانند تقاضای است که معمولاً برای حذف فایلی از سرویس‌دهنده بکار گرفته می‌شود. غالباً این تقاضا بدلایل امنیتی قابل انجام بر روی اکثر سرویس‌دهنده‌ها نیست.
head	معمولاً از این تقاضا زمانی استفاده می‌شود که سرویس‌گیرنده می‌خواهد فقط واکنش دهنده به سرآیندها باشد، همانند نوع محتویات و طول محتویات.
options	چنین تقاضای، اطلاعاتی به سرویس‌گیرنده برگشت می‌دهد که دلالت بر گزینه‌های پشتیبانی شده HTTP از سوی سرویس‌دهنده دارند، همانند نسخه HTTP (1.1 یا 1.0) و متدهای تقاضا که سرویس‌دهنده از آنها پشتیبانی می‌کند.
put	معمولاً از چنین تقاضای برای ذخیره فایل بر روی سرویس‌دهنده استفاده می‌شود. غالباً این تقاضا بدلایل امنیتی قابل انجام بر روی اکثر سرویس‌دهنده‌ها نیست.
trace	معمولاً از چنین تقاضای برای خطایابی استفاده می‌شود.

شکل ۱-۱۹ | دیگر نوع‌های تقاضای HTTP.

۳-۱۹ معماری چند گره‌ای

سرویس‌دهنده وب بخشی از *برنامه چندگره‌ای* (*multi-tier*) است که گاهی اوقات بعنوان برنامه*n گره‌ای* نیز شناخته می‌شود. عملکرد برنامه‌های چندگره‌ای به گره‌های جداگانه (یا گروه‌های منطقی) تقسیم می‌شود. اگرچه گره‌ها می‌توانند بر روی یک کامپیوتر قرار داشته باشند، اما غالباً گره‌های برنامه‌های مبتنی بر وب بر روی کامپیوترهای جداگانه قرار داده می‌شوند. شکل ۲-۱۹ ساختار اولیه یک برنامه سه گره‌ای را نشان می‌دهد.

گره اطلاعات (با نام *گره داده* یا *گره تحتانی* نیز شناخته می‌شود) وظیفه نگهداری داده‌های برنامه را برعهده دارد. عموماً این گره، داده‌ها را در یک *سیستم مدیریت پایگاه داده رابطه‌ای* (*RDBMS*) ذخیره می‌کند. برای مثال یک فروشگاه می‌تواند دارای پایگاه داده‌ای برای ذخیره‌سازی اطلاعاتی همانند نوع

کالا، قیمت و تعداد آنها باشد. همان پایگاه داده می‌تواند حاوی اطلاعات هر مشتری همانند نام کاربر، آدرس و شماره کارت اعتباری باشد. این گره می‌تواند متشکل از چندین پایگاه داده باشد که به کمک هم اطلاعات مورد نیاز برنامه را فراهم می‌آورند.

گره میانی وظیفه پیاده‌سازی *موازنه منطقی* و *کنترل* تعامل صورت گرفته مابین برنامه سرویس‌گیرنده و داده‌های برنامه را برعهده دارد. گره میانی همانند یک میانجی مابین داده‌های موجود در گره اطلاعات و برنامه‌های سرویس‌گیرنده عمل می‌کند. گره میانی مبادرت به کنترل منطقی تقاضاهای سرویس‌گیرنده‌ها کرده و اطلاعات را از پایگاه داده بازیابی می‌کند. معمولاً برنامه‌های وب داده‌ها را بفرم مستندات HTML به سرویس‌گیرنده‌ها عرضه می‌کنند.

شکل ۲-۱۹ | معماری سه گره‌ای.

منطق موازنه در گره میانی، سبب اعمال *قوانین موازنه* می‌شود و کاری می‌کند که داده‌ها قبل از اینکه برنامه سرویس‌دهنده مبادرت به روز کردن پایگاه داده یا ارائه اطلاعات به کاربران نماید، قابل اطمینان و قابل عرضه شوند. قوانین موازنه نحوه دسترسی سرویس‌گیرنده‌ها به اطلاعات و پردازش داده‌ها را تعیین می‌کنند.

گره سرویس‌گیرنده یا *گره فوقانی*، برنامه واسط کاربر است که عموماً یک مرورگر وب می‌باشد و کاربران مستقیماً و از طریق واسط کاربر با برنامه در تعامل قرار می‌گیرند. گره سرویس‌گیرنده با گره میانی در ارتباط است تا تقاضای خود را مطرح و داده‌ها را از گره اطلاعات دریافت کند. سپس گره س‌رویس‌گیرنده داده‌های دریافتی از گره میانی را در اختیار کاربر قرار می‌دهد.

۱۹-٤ دسترسی به سرویس‌دهنده‌های وب

برای اینکه بتوان از مستندات مقیم بر روی سرویس‌دهنده‌های وب استفاده کرد، نیاز است تا با URL آنها آشنا بود. یک URL حاوی نام ماشین (به *نام میزبان* شناخته می‌شود) است که بر روی سرویس‌دهنده وب مقیم می‌باشد. کاربران می‌توانند، مستندات را از *سرویس‌دهنده‌های وب محلی* (مقیم بر روی ماشین یکی از کاربران) یا *سرویس‌دهنده‌های وب راه دور* (مقیم بر روی یکی از ماشین‌های موجود در شبکه) درخواست نمایند.

به دو روش می‌توان به سرویس‌دهنده‌های وب محلی دسترسی پیدا کرد: از طریق نام ماشین، یا از طریق localhost – نام میزبانی که اشاره به یک ماشین محلی دارد. در این فصل از localhost استفاده می‌کنیم. برای تعیین نام ماشین در ویندوز 2000، بر روی My Computer کلیک راست کرده و از

منوی ظاهر شده Properties را برای به نمایش در آمدن کادر تبادلی System Properties، انتخاب نمائید. در این کادر تبادلی، بر روی Network Identification کلیک کنید. فیلد Full Computer Name: در پنجره System Properties نام کامپیوتر را نشان می‌دهد. در ویندوز XP، منوی Start>Control Panel>Switch to Classic View> System را برای نمایش کادر تبادلی System Properties انتخاب کنید. در این کادر بر روی زبانه Computer Name کلیک نمائید.

نام دامنه نشاندهنده گروهی از میزبان‌ها در اینترنت است؛ که با نام یک میزبان (برای نمونه، **www**) و یک *دامنه سطح بالا* (TLD[1]) ترکیب می‌شوند، تا روش کاربر پسندی برای شناسائی یک سایت در اینترنت بدست آید. به هر کدامیک از این اسامی میزبان یک آدرس منحصر بفرد بنام *آدرس IP* تخصیص داده می‌شود. این آدرس‌ها بسیار شبیه آدرس یک خانه در یک شهر هستند. کامپیوترها با استفاده از آدرس‌های IP مبادرت به یافتن کامپیوترهای دیگر در اینترنت می‌کنند. یک *سرویس‌دهنده سیستم نام دامنه* یا *DNS*[2]، کامپیوتری است که نگهداری پایگاه داده اسامی میزبان‌ها و آدرس‌های IP متناظر با آنها، ترجمه اسامی میزبان‌ها به آدرس‌های IP، را بر عهده دارد. به این عمل ترجمه *DNS lookup* گفته می‌شود. برای مثال، برای دسترسی به وب سایت Deitel، نام میزبان (www.deitel.com) را در مرورگر وب تایپ می‌کنیم. سپس سرویس‌دهنده DNS مبادرت به ترجمه www.deitel.com به آدرس IP سرویس‌دهنده وب Deitel می‌کند (63.110.43.82). آدرس IP، localhost برابر 127.0.01 است.

۱۹-۵ سرویس‌دهنده Apache HTTP

سرویس‌دهنده Apache HTTP توسط Apache Software Foundation پشتیبانی می‌شود و در حال حاضر یکی از محبوبترین سرویس دهنده‌های وب است چرا که از پایداری، هزینه، کارایی و قابلیت حمل مناسبی برخوردار است. این نرم‌افزار یک نرم‌افزار open-source است (به این معنی که کد منبع آن مجاناً و بدون محدودیت در اختیار همه قرار دارد) که بر روی پلاتفرم‌های UNIX، Linux و Windows اجرا می‌شود.

برای برداشت کردن سرویس‌دهنده Apache HTTP از سایت httpd.appache.org بازدید کنید. برای کسب دستورالعمل‌های نصب Apache می‌توانید به وب سایت www.deitel.com یا httpd.apache.org مراجعه کنید. اگر سرویس‌دهنده Apache HTTP بعنوان یک سرویس نصب شده باشد، آماده اجرا پس از نصب است. در غیر اینصورت، می‌توانید سرویس‌دهنده را با انتخاب منوی Start، سپس All Programs > Apache HTTP Server 2.0.52 > Control Apache Server > Start فعال کنید. برای متوقف کردن

1- Top-Level Domain
2- Domain Name System

سرویس‌دهنده Apache مراحل فوق را انجام داده و در پایان گزینه Stop را انتخاب نمائید. برای کاربران Linux، دستورالعمل‌های stop/start سرویس‌دهنده Apache HTTP و اجرای مثال‌ها در وب سایت www.deital.com آورده شده است.

۱۹-۶ تقاضای مستندات XHTML

این بخش شما را با نحوه تقاضای یک مستند XHTML از سرویس‌دهنده Apache HTTP آشنا می‌کند. در ساختار شاخه سرویس‌دهنده Apache HTTP، بایستی مستندات XHTML در شاخه htdocs ذخیره شوند. بر روی سیستم عامل ویندوز، شاخه htdocs در مسیر C:\ProgramsFiles\Apache Group\Apache2 قرار دارد. در سیستم عامل لینوکس، شاخه htdocs در شاخه /user/local/httpd مقیم است. مستند (فایل) test.html را از شاخه مثال‌های فصل ۱۹ که بر روی CD کتاب قرار دارد، کپی کرده و در شاخه htdocs قرار دهید. برای تقاضای مستند، مرورگر وب (همانند Internet Explorer یا Netscape) را اجرا کرده و URL را در فیلد Address وارد سازید (یعنی http://localhost/test.html). شکل ۳-۱۹ نتیجه تقاضای test.html را نشان می‌دهد. [نکته: در Apache، شاخه ریشه سرویس‌دهنده اشاره به شاخه پیش‌فرض، htdocs، دارد از اینرو نام شاخه را قبل از نام فایل (یعنی test.html) در فیلد Address وارد نکرده‌ایم.]

شکل ۳-۱۹ | تقاضای test.html از Apache.

۱۹-۷ معرفی CGI

واسط CGI یا Common Gateway Interface یک پروتکل استاندارد است که به برنامه‌ها امکان می‌دهد تا با سرویس‌دهنده‌های وب و (بطور غیرمستقیم) با سرویس‌گیرنده‌ها (مثلاً مرورگرهای وب) در تعامل قرار گیرند. معمولاً به این برنامه‌ها، برنامه‌های CGI یا اسکریپت‌های CGI می‌گویند. غالباً از CGI برای تولید محتویات دینامیکی وب با استفاده از ورودی سرویس‌گیرنده، پایگاه داده‌ها و سایر سرویس‌های اطلاعاتی، استفاده می‌شود. یک صفحه وب در صورتی دینامیکی است که محتویات آن بصورت برنامه‌نویسی شده در زمان تقاضای برنامه تولید شود، برخلاف محتویات استاتیکی وب که محتویات آن بصورت برنامه‌نویسی شده در زمان تقاضا تولید نمی‌شود.

برای مثل، می‌توانیم از یک صفحه وب استاتیکی استفاده کرده و از کاربر بخواهیم کد پستی را وارد کند، سپس کاربر را به طرف یک اسکریپت CGI هدایت کنیم که برحسب ورودی کاربر یک صفحه وب دینامیک تولید نماید. در این فصل، به معرفی اصول CGI و استفاده از C++ در نوشتن اسکریپت‌های CGI خواهیم پرداخت. CGI اختصاص به یک سیستم عامل خاص و ویژه یا یک زبان برنامه‌نویسی ندارد. CGI به نحوی طراحی شده که می‌تواند با هر زبان برنامه‌نویسی همانند C، C++، Perl، Python یا VisualBasic بکار گرفته شود.

CGI در سال ۱۹۹۳ توسط NCSA (National Centr for Supercomputing Applications) به منظور استفاده با سرویس‌دهنده وب HTTPd توسعه یافته است. برخلاف پروتکل‌های وب و زبان‌های که دارای ساختار رسمی هستند، توصیف اولیه CGI که توسط NCSA نوشته شده اثبات کرده است که CGI بعنوان یک استاندارد غیررسمی در جهان پذیرفته شده است. پشتیبانی از CGI بسرعت در میان سرویس‌دهنده‌های وب از جمله Apache پذیرفته شده است.

۸-۱۹ تراکنش ساده HTTP

قبل از بررسی نحوه عملکرد CGI، درک اولیه‌ای از شبکه و وب گسترده جهانی (www[1]) ضروری است. در این بخش، به بررسی عملکرد داخلی *پروتکل انتقال فوق متن* (HTTP[2]) و اتفاقاتی که در پس پرده نمایش یک صفحه وب در مرورگر رخ می‌دهد، می‌پردازیم. HTTP مشخص کننده مجموعه‌ای از *متدها* و *سرآیندها* (*headers*) است که به سرویس‌گیرنده‌ها و سرویس‌دهنده‌ها امکان می‌دهند تا با یکدیگر به تعامل پرداخته و اطلاعات را به یک شکل واحد و به روش قابل اطمینانی مبادله نمایند.

در ساده‌ترین فرم، یک صفحه وب چیزی بیش از یک مستند HTML نیست. این مستند یک فایل متنی ساده حاوی *نشانه‌ها* یا *دنباله‌ها* است که به مرورگر وب نشان می‌دهند چگونه اطلاعات موجود در مستند را به نمایش در آورد. برای مثال، به نشانه HTML زیر توجه کنید:

```
<title> My Web Page </title>
```

این نشانه، به مرورگر اعلان می‌کند که عبارت قرار گرفته مابین دو دنباله شروع و پایانی (**<title>** و **</title>**) عنوان صفحه وب است. همچنین مستندات HTML می‌توانند حاوی داده‌های *فوق متن* (با نام *فوق لینک* نیز شناخته می‌شوند) باشند، که لینک‌هایی به صفحات مختلف یا بخش‌های دیگری در همان صفحه ایجاد می‌کنند. هنگامی که کاربر بر روی یک فوق لینک کلیک می‌کند، صفحهٔ درخواستی وب (یا بخش متفاوتی از همان صفحه وب) به بدورن پنجره مرورگر کاربر بار می‌شود.

هر مستندی که برای نمایش بر روی وب در دسترس می‌باشد، دارای یک URL[3] است که در واقع آدرسی است که موقعیت یا مکان منبع را نشان می‌دهد. URL حاوی اطلاعاتی است که مرورگر را به منبع مستند که کاربر مایل به دسترسی به آن است، هدایت می‌کند. کامپیوترهایی که نرم‌افزار سرویس‌دهنده وب بر روی آنها اجرا می‌شود چنین منابعی را فراهم می‌آورند. اجازه دهید تا به بررسی اجزای URL زیر بپردازیم:

```
http://www.deitel.com/books/downlods.htm
```

1- World Wide Web
2- HyperText Transfer Protocol
3- Uniform Resource Locator

http:// نشان می‌دهد که منبع از پروتکل HTTP استفاده می‌کند. بخش میانی، **www.deitel.com** نام کامل میزبان سرویس‌دهنده است. *نام میزبان*، نام کامپیوتری است که منابع بر روی آن قرار دارند. معمولاً به این کامپیوترها، *میزبان* گفته می‌شود، چرا که منزلگاه و نگهدارنده منابع هستند. نام میزبان **www.deitel.com** به یک *آدرس IP* ترجمه می‌شود (**63.110.43.82**)، که نشاندهنده هویت سرویس‌دهنده می‌باشد، این هویت همانند یک شماره تلفن است که منحصراً متعلق به یک خط تلفن می‌باشد. معمولاً ترجمه نام میزبان به یک آدرس IP توسط یک *سرویس‌دهنده نام دامنه* (DNS[1]) صورت می‌گیرد. DNS به کامپیوتری اطلاق می‌شود که پایگاه داده‌ای (بانک اطلاعاتی) از اسامی میزبان‌ها و آدرس‌های IP آنها را در خود نگهداری می‌کند. به این فرآیند ترجمه *DNS lookup* گفته می‌شود.

مابقی URL نشاندهنده *نام منبع* درخواستی یعنی **/books/downloads.htm** (یک مستند HTML) است. این بخش از URL هم نام منبع (**downloads.htm**) و هم مسیر یا مکان آنرا (**/books**) بر روی سرویس‌دهنده وب مشخص کرده است. مسیر بکار رفته می‌تواند نشاندهنده یک شاخه واقعی بر روی سیستم فایل سرویس‌دهنده وب باشد. با این وجود، به دلایل امنیتی، غالباً مسیر مشخص شده، نشاندهنده یک شاخه مجازی است. در چنین سیستم‌هایی، سرویس‌دهنده مبادرت به تبدیل شاخه مجازی به یک مکان واقعی بر روی سرویس‌دهنده (یا کامپیوتر دیگری بر روی شبکه سرویس‌دهنده) می‌کند، از اینرو مکان واقعی منابع پنهان خواهد ماند. علاوه بر این، برخی از منابع بصورت دینامیکی ایجاد می‌شوند و در هیچ کجای کامپیوتر سرویس‌دهنده قرار ندارند.

به هنگام وارد کردن یک URL، مرورگر مبادرت به انجام یک تراکنش ساده HTTP برای دریافت و نمایش صفحه وب مورد تقاضا می‌کند. در شکل ۴-۱۹ جزئیات تراکنش صورت گرفته، دیده می‌شود. این تراکنش متشکل از تأثیر متقابل مابین مرورگر وب (طرف سرویس‌گیرنده یا *Client*) و برنامه کاربردی سرویس‌دهنده وب (طرف سرویس‌دهنده یا *server*) می‌باشد. در شکل ۴-۱۹، مرورگر وب یک تقاضای HTTP به سرویس‌دهنده ارسال کرده است. تقاضا (در ساده‌ترین فرم) عبارت است از

```
GET /books/downloads.htm HTTP/1.1
Host: wwww.deitel.com
```

کلمه ***GET*** یک *متد HTTP* است که نشان می‌دهد سرویس‌گیرنده مایل به بدست آوردن منبعی از سرویس‌دهنده است. مابقی تقاضا، تدارک بیننده نام مسیر منبع (یک مستند HTML) و نام پروتکل و شماره نسخه آن می‌باشد (**HTTP/1.1**).

1- Domain Name Server

هر سرویس‌دهنده‌ای که قادر به درک HTTP (نسخه 1.1) باشد می‌تواند این تقاضا را بررسی کرده و بطور مناسب به آن پاسخ دهد. در شکل ۴-۱۹ نتیجه درخواست دیده می‌شود. ابتدا سرویس‌دهنده با ارسال یک عبارت متنی که نشاندهنده نسخه HTTP و بدنبال آن یک کد عددی و کلمه تعیین وضعیت تراکنش است به سرویس‌گیرنده پاسخ می‌دهد. برای مثال

```
HTTP/1.1 200 OK
```

نشاندهنده موفقیت آمیز بودن عمل است در حالیکه

```
HTTP/1.1 404 Not found
```

به سرویس‌گیرنده اعلان می‌کند که سرویس‌دهنده وب نتوانسته است منبع درخواستی را پیدا کند.

شکل ٤-١٩ | تعامل سرویس‌گیرنده با سرویس‌دهنده. گام 1: تقاضای GET،
GET/books/downloads.htm/HTTP/1.1

شکل ٤-١٩ | تعامل سرویس‌گیرنده با سرویس‌دهنده. گام 2: پاسخ HTTP، HTTP/1.1 200 OK.

سپس سرویس‌دهنده یک یا چند سرآیند HTTP ارسال می‌کند که حاوی اطلاعات بیشتری در مورد داده‌ای است که ارسال خواهد شد. در این مورد، سرویس‌دهنده یک مستند متنی HTML ارسال می‌کند، از اینرو سرآیند HTTP برای این مثال بصورت زیر خواهد بود:

```
Content-type: text/html
```

اطلاعات تدارک دیده شده در این سرآیند مشخص کننده نوع محتویات *MIME*[1] است که سرویس‌دهنده آنها را به مرورگر انتقال می‌دهد. MIME یکی از استانداردهای اینترنت است که روش قالبندی داده‌های مشخصی را تعیین می‌کند تا برنامه‌ها بتوانند داده‌ها را بطرز صحیحی تفسیر نمایند. برای مثال، **text/plain** از نوع MIME است که نشان می‌دهد اطلاعات ارسالی متنی است که می‌تواند مستقیماً و بدون هیچ گونه تفسیری به نمایش درآید. به همین ترتیب نوع **image/gif** نشان می‌دهد که محتوی یک تصویر GIF است. زمانیکه مرورگر این نوع از MIME را دریافت کند، مبادرت به نمایش تصویر خواهد کرد.

1- Multipurpose Internet Mail Extensions

یک خط خالی که پس از سرآیند یا سرآیندها قرار می‌گیرد به سرویس‌گیرنده اعلان می‌کند که سرویس‌دهنده به ارسال سرآیندهای HTTP خاتمه داده است. سپس سرویس‌دهنده محتویات مستند HTML تقاضا شده (**downloads.htm**) را ارسال می‌کند. پس از کامل شدن انتقال منبع، سرویس‌دهنده به اتصال برقرار شده، خاتمه می‌دهد. در این نقطه، مرورگر طرف سرویس‌گیرنده شروع به تجزیه HTML دریافتی و راندو (یا نمایش) آن می‌کند.

۹-۱۹ اسکریپت‌های ساده CGI

مادامیکه یک فایل XHTML بر روی سرویس‌دهنده بدون تغییر باقی بماند، URL مرتبط با آن، همان محتویات را در هر بار دسترسی فایل در مرورگر سرویس‌گیرنده به نمایش در خواهد آورد. برای اعمال تغییر در محتویات یک فایل XHTML (مثلاً افزودن لینک‌های جدید)، بایستی فردی فایل را بصورت دستی بر روی سرویس‌دهنده و احتمالاً با استفاده از یک برنامه ویرایشگر متن یا نرم‌افزار طراحی صفحه وب تغییر دهد. این نحوه تغییر برای مولفان صفحه وب که می‌خواهند صفحه وب دینامیکی جالبی ایجاد کند، مشکل است. داشتن فردی که مدام صفحه وب را تغییر دهد، کاری خسته کننده است. برای مثال، اگر بخواهید صفحه وب شما همیشه تاریخ جاری یا شرایط آب و هوا را به نمایش درآورد، باید این صفحه مرتباً به روز شود.

اولین اسکریپت CGI

اولین اسکریپت CGI ما در شکل ۵-۱۹ آورده شده است. توجه کنید که برنامه بیشتر از عبارات **cout** تشکیل شده است (خطوط 26-19). تا بدین جا، همیشه خروجی **cout** بر روی صفحه نمایش ظاهر می‌شد. با این وجود، به لحاظ تکنیکی، مکان یا هدف پیش‌فرض برای **cout**، خروجی استاندارد است. زمانیکه یک برنامه ++C بعنوان یک اسکریپت CGI اجرا می‌شود، خروجی استاندارد، توسط سرویس‌دهنده وب به سمت مرورگر وب سرویس‌گیرنده هدایت می‌شود. برای اجرای برنامه بعنوان یک اسکریپت CGI، فایل اجرائی کامپایل شده ++C را در شاخه **cgi-bin** سرویس‌دهنده وب قرار داده‌ایم. برای برآورده کردن اهداف این فصل، پسوند فایل اجرای را از **exe.** با **cgi.** تغییر داده‌ایم. با فرض اینکه سرویس‌دهنده وب بر روی کامپیوتر محلی شما قرار دارد، می‌توانید اسکریپت را با تایپ

```
http://localhost/cgi-bin/localtime.cgi
```

در فیلد Address یا Location مرورگر خود، به اجرا در آورید. اگر این اسکریپت را از یک سرویس‌دهنده وب راه دور تقاضا کنید، نیاز دارید تا localhost را با نام ماشین سرویس‌دهنده یا آدرس IP آن جایگزین سازید.

```
1  // Fig. 19.5: localtime.cpp
2  // Displays the current date and time in a Web browser.
3  #include <iostream>
4  using std::cout;
```

```
5
6  #include <ctime>//definitions of time_t, time, localtime and asctime
7  using std::time_t;
8  using std::time;
9  using std::localtime;
10 using std::asctime;
11
12 int main()
13 {
14    time_t currentTime; // variable for storing time
15
16    cout << "Content-Type: text/html\n\n"; // output HTTP header
17
18    // output XML declaration and DOCTYPE
19    cout << "<?xml version = \"1.0\"?>"
20       << "<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.1//EN\" "
21       << "\"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd\">";
22
23    time( &currentTime ); // store time in currentTime
24
25    // output html element and some of its contents
26    cout << "<html xmlns = \"http://www.w3.org/1999/xhtml\">"
27       << "<head><title>Current date and time</title></head>"
28       << "<body><p>" << asctime( localtime( &currentTime ) )
29       << "</p></body></html>";
30    return 0;
31 } // end main
```

شکل ۵-۱۹ | اولین اسکریپت CGI.

نظریه خروجی استاندارد شبیه به ورودی استاندارد است، که آنرا مرتبط با **cin** می‌دانیم. همانطور که ورودی استاندارد اشاره به منبع استاندارد ورودی به برنامه (معمولاً صفحه‌کلید) دارد، خروجی استاندارد اشاره به مقصد استاندارد خروجی از برنامه (معمولاً صفحه نمایش) دارد. امکان هدایت (یا لوله‌کشی) خروجی استاندارد به مقصد دیگر وجود دارد. از اینرو، اسکریپت CGI ما، زمانیکه یک سرآیند HTTP (خط 16) یا عناصر XHTML (خطوط 21-19 و 29-26) را خارج می‌سازیم، خروجی به سرویس‌دهنده وب ارسال می‌شود. سرویس‌دهنده آن خروجی را به سرویس‌دهنده از طریق HTTP ارسال می‌کند که، سرآیند و عناصر را تفسیر می‌کند.

نوشتن یک برنامه C++ که زمان و تاریخ جاری را چاپ کند کار سختی نیست. در واقع اینکار مستلزم نوشتن چند خط کد است (خطوط 23,14 و 28). خط 14 مبادرت به اعلان متغیر **currentTime** بعنوان متغیری از نوع **time_t** کرده است. تابع **time** در خط 23 زمان جاری را بدست آورده و مقدار آنرا در مکان مشخص شده توسط پارامتر ذخیره می‌کند (در این مورد **currentTime**). تابع کتابخانه‌ای **localtime** (خط 28)، زمانیکه متغیر **time_t** را می‌پذیرد (یعنی **currentTime**) یک اشاره‌گر به شی که حاوی زمان محلی است برگشت می‌دهد. تابع **asctime** در خط 28 که یک اشاره‌گر به شی که حاوی زمان است دریافت می‌کند، رشته‌ای بصورت زیر برگشت می‌دهد

```
Wed Oct 31 13:10:37 2007
```

اگر بخواهیم زمان جاری را به پنجره مرورگر یک سرویس‌گیرنده ارسال کنیم چه باید کرد؟ CGI این امکان را با هدایت خروجی یک برنامه به خود سرویس‌دهنده وب فراهم می‌آورد که آن خروجی هم به مرورگر سرویس‌گیرنده ارسال می‌شود.

نحوه هدایت خروجی توسط سرویس‌دهنده وب

در شکل ۶-۱۹ این فرآیند بدقت بررسی شده است. در گام اول، سرویس‌گیرنده تقاضای منبعی بنام **localtime.cgi** از سرویس‌دهنده می‌کند، همانند تقاضای صورت گرفته برای **downloads.html** در مثال قبلی (شکل ۴-۱۹). اگر سرویس‌دهنده برای کار با اسکریپت CGI پیکربندی نشده باشد، می‌تواند محتویات فایل اجرای C++ را به سرویس‌دهنده برگشت دهد، مثل اینکه مستند دیگری وجود ندارد. با این همه، براساس پیکربندی سرویس‌دهنده وب، سرویس‌دهنده **localtime.cgi** (پیاده‌سازی شده توسط C++) را اجرا کرده و خروجی اسکریپت CGI را به مرورگر وب ارسال می‌کند.

به هر حال اگر سرویس‌دهنده وب بدرستی پیکربندی شده باشد، انواع منابع مختلف را که بایستی به روش‌های متفاوتی با آنها کار شود را تشخیص خواهد داد. برای مثال، زمانیکه منبع یک اسکریپت CGI باشد، بایستی اسکریپت توسط سرویس‌دهنده قبل از ارسال آن، اجرا شده باشد. یک اسکریپت CGI به یکی از دو روش معین می‌شوند: خواه دارای یک پسوند نام فایل ویژه است (همانند **cgi.** یا **exe.**) یا در شاخه خاص قرار دارد (اغلب **cgi-bin**).

علاوه بر این، بایستی مدیر سرویس‌دهنده به صراحت مجوزهای لازم را در اختیار سرویس‌گیرنده‌های راه دور قرار دهد تا آنها بتوانند به اسکریپت‌های CGI دسترسی پیدا کرده و به اجرا در آورند.

در گام دوم از شکل ۶-۱۹، سرویس‌دهنده تشخیص داده است که منبع یک اسکریپت CGI است و اسکریپت را اجرا می‌کند. در گام سوم، خروجی توسط سه عبارت **cout** تولید شده (خطوط 16، 21-19 و 29-26 از شکل ۵-۱۹) و به خروجی استاندارد ارسال شده و به سرویس‌دهنده وب برگشت داده می‌شود. سرانجام در گام چهارم، سرویس‌دهنده وب پیغامی به خروجی اضافه می‌کند که دلالت بر وضعیت تراکنش HTTP دارد (همانند HTTP/1.1 200 Ok برای موفقیت) و کل خروجی را از برنامه CGI به سرویس‌گیرنده ارسال می‌کند.

شکل ٦-١٩ | گام اول: تقاضای GET/cgi-bin/localtime.cgi HTTP/1.1.get.

شکل ٦-١٩ | گام دوم: سرویس‌دهنده وب اسکریپت CGI را راه‌اندازی می‌کند.

شکل ٦-١٩ | گام سوم: خروجی اسکریپت به سرویس‌دهنده وب ارسال می‌شود.

سپس مرورگر طرف سرویس‌گیرنده مبادرت به پردازش مستند XHTML کرده و نتیجه را به نمایش در می‌آورد. توجه داشته باشید که مرورگر از آنچه که در سرویس‌دهنده اتفاق می‌افتد بی‌خبر است. به عبارتی

دیگر، تا آنجا که به مرورگر مربوط می‌شود، وی تقاضای منبعی را مطرح کرده و پاسخی را دریافت می‌کند. مرورگر خروجی اسکریپت را دریافت و تفسیر می‌کند.

شکل ٦-١٩ | گام چهارم: پاسخ HTTP، HTTP/1.1 200 OK.

در واقع، می‌توانید با اجرای **localtime.cgi** از طریق خط فرمان، شاهد محتوی باشید که مرورگر دریافت می‌کند. شکل ٧-١٩ نمایشی از خروجی است. به منظور برآورده کردن اهداف این فصل، خروجی را برای اینکه قابل فهم باشد، قالب‌بندی کرده‌ایم. توجه کنید که در اسکریپت CGI، بایستی خروجی شامل سرآیند Content-Type باشد، از آنجاییکه در یک مستند XHTML، سرویس‌دهنده وب این سرآیند را شامل می‌شود.

اسکریپت CGI سرآیند Content-Type، یک خط خالی و داده (XHTML، متن ساده و غیره) را در خروجی استاندارد چاپ می‌کند. زمانیکه اسکریپت CGI بر روی سرویس‌دهنده وب اجرا می‌شود، سرویس‌دهنده خروجی اسکریپت را بازیابی کرده، پاسخ HTTP را به ابتدای آن وارد و محتوی را به سرویس‌گیرنده ارسال می‌کند.

```
Content-Type: text/html

<?xml version = "1.0"?>
<!DOCTYPE html PUBLIC "_//W3C//DTD XHTML 1.1//EN"
     http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd>

<html xmlns = http://www.w3.org/1999/xhtml>
     <head>
          <title>Current date and time</title>
     </head>

     <body>
          <p>Wet Oct 13 10:22:18 2004</p>
     </body>
</html>
```

شکل ٧-١٩ | خروجی localtime.cgi به هنگام اجرا از طریق خط فرمان.

نمایش متغیرهای محیطی

برنامه شکل ٨-١٩ مبادرت به نمایش متغیرهای محیطی می‌کند که سرویس‌دهنده Apache HTTP برای اسکریپت‌های CGI تنظیم می‌نماید. این متغیرها حاوی اطلاعاتی در مورد محیط سرویس‌گیرنده و سرویس‌دهنده، همانند نوع مرورگر وب بکار رفته و مکان مستندی بر روی سرویس‌دهنده هستند. خطوط 14-23 آرایه رشته‌ای با اسامی متغیرهای محیطی CGI را مقداردهی اولیه کرده‌اند. خط 37 آغاز جدول XHTML است که داده‌ها در آن به نمایش در می‌آیند.

```
1  // Fig. 19.8: environment.cpp
2  // Program to display CGI environment variables.
3  #include <iostream>
4  using std::cout;
5
6  #include <string>
```

```
7  using std::string;
8
9  #include <cstdlib>
10 using std::getenv;
11
12 int main()
13 {
14    string environmentVariables[ 24 ] = {
15       "COMSPEC", "DOCUMENT_ROOT", "GATEWAY_INTERFACE",
16       "HTTP_ACCEPT", "HTTP_ACCEPT_ENCODING",
17       "HTTP_ACCEPT_LANGUAGE", "HTTP_CONNECTION",
18       "HTTP_HOST", "HTTP_USER_AGENT", "PATH",
19       "QUERY_STRING", "REMOTE_ADDR", "REMOTE_PORT",
20       "REQUEST_METHOD", "REQUEST_URI", "SCRIPT_FILENAME",
21       "SCRIPT_NAME", "SERVER_ADDR", "SERVER_ADMIN",
22       "SERVER_NAME","SERVER_PORT","SERVER_PROTOCOL",
23       "SERVER_SIGNATURE","SERVER_SOFTWARE" };
24
25    cout << "Content-Type: text/html\n\n"; // output HTTP header
26
27    // output XML declaration and DOCTYPE
28    cout << "<?xml version = \"1.0\"?>"
29       << "<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.1//EN\" "
30       << "\"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd\">";
31
32    // output html element and some of its contents
33    cout << "<html xmlns = \"http://www.w3.org/1999/xhtml\">"
34       << "<head><title>Environment Variables</title></head><body>";
35
36    // begin outputting table
37    cout << "<table border = \"0\" cellspacing = \"2\">";
38
39    // iterate through environment variables
40    for ( int i = 0; i < 24; i++ )
41    {
42       cout << "<tr><td>" << environmentVariables[i] << "</td><td>";
43
44       // attempt to retrieve value of current environment variable
45       char *value = getenv( environmentVariables[ i ].c_str() );
46
47       if ( value != 0 ) // environment variable exists
48          cout << value;
49       else
50          cout << "Environment variable does not exist.";
51
52       cout << "</td></tr>";
53    } // end for
54
55    cout << "</table></body></html>";
56    return 0;
57 } // end main
```

شکل ۸-۱۹ | بازیابی متغیرهای محیطی از طریق تابع getenv.

خطوط 52-42 هر سطر از جدول را چاپ می‌کنند. اجازه دهید تا از نزدیک به بررسی این خطوط بپردازیم. خط 42 یک دنباله آغازین **<tr>** (سطر جدول) را که نشاندهنده ابتدای یک سطر جدید در جدول است را در خروجی قرار می‌دهد. خط 52 دنباله پایانی متناظر **</tr>** را که دلالت بر انتهای سطر دارد، در خروجی قرار می‌دهد. هر سطر از جدول حاوی دو سلول است که برای نام متغیر محیطی و داده

مرتبط با آن متغیر در نظر گرفته شده‌اند. دنباله شروع **<td>** در خط 42 آغاز یک سلول جدید در جدول است. حلقه **for** در خطوط 53-40، در میان 24 شی رشته حرکت می‌کند. نام هر متغیر محیطی در سمت چپ سلول به نمایش در می‌آید (خط 42). خط 45 مبادرت به بازیابی مقدار مرتبط با متغیر محیطی توسط فراخوانی تابع **getenv** از **<cstdlib>** کرده و مقدار رشته‌ای برگشت داده شده از فراخوانی تابع **environmentVariables[i].c_str()** است. تابع **getenv** یک رشته *** char** حاوی مقدار یک متغیر محیطی مشخص شده را برگشت می‌دهد یا اگر متغیر محیطی وجود نداشته باشد، اشاره‌گر **null** برگشت می‌دهد.

خطوط 50-47 محتویات را در سلول راست چاپ می‌کنند. اگر متغیر محیطی وجود داشته باشد (یعنی **getenv** اشاره‌گر **null** برگشت ندهد)، خط 48 مقدار برگشتی توسط تابع **getenv** را چاپ می‌کند. اگر متغیر محیطی وجود نداشته باشد، خط 50 پیغام مناسبی چاپ می‌نماید. اجرای نمونه این برنامه در شکل ۸-۱۹ به هنگام اجرای این مثال بر روی سرویس‌دهنده Apache HTTP آورده شده است.

۱۰-۱۹ ارسال ورودی به اسکریپت CGI

اگرچه متغیرهای محیطی از پیش تنظیم شده اطلاعات زیادی دارند، اما می‌خواهیم انواع مختلفی از اطلاعات را در اسکریپت CGI خود همانند نام کاربر یا پرس‌وجوی موتور جستجو داشته باشیم. متغیر محیطی **QUERY_STRING** مکانیزمی است که اینکار را انجام می‌دهد. متغیر **QUERY_STRING** حاوی اطلاعاتی است که به URL در ضمن یک تقاضا الصاق می‌شود. برای مثال، URL

```
www.somesite.com/cgi-bin/script.cgi?state=California
```

سبب می‌شود تا مرورگر وب تقاضای یک اسکریپت (cgi-bin/script.cgi) را با رشته پرس‌وجو (state=California) از www.somesite.com انجام دهد. سرویس‌دهنده وب رشته پرس‌وجوی پس از **?** را در متغیر محیطی **QUERY_STRING** ذخیره می‌سازد. رشته پرس‌وجو پارامترهای فراهم می‌آورد که تقاضا برای یک سرویس‌گیرنده مشخص را بهینه‌سازی می‌کنند. دقت کنید که علامت سوال (**?**) بخشی از منبع درخواستی و رشته پرس‌وجو نمی‌باشد. این کاراکتر فقط نقش جدا کننده مابین این دو را بازی می‌کند.

برنامه شکل ۹-۱۹ مثال ساده‌ای از یک اسکریپت CGI است که داده را خوانده و از طریق **QUERY_STRING** ارسال می‌کند. به روش‌های مختلف می‌توان رشته پرس‌وجو را قالب‌بندی کرد. اسکریپت CGI که رشته پرس‌وجو را می‌خواند باید از نحوه تفسیر داده قالب‌بندی شده مطلع باشد. در مثال شکل ۹-۱۹ رشته پرس‌وجو حاوی دنباله‌ای از جفت‌های مقدار–نام است که توسط آمپرسنج (**&**) بصورت **name=Jill&age=22** از هم متمایز شده‌اند.

در خط 16 از شکل ۹-۱۹ رشته "QUERY_STRING" به تابع **getenv** ارسال شده که رشته پرس‌وجو یا اشاره‌گر **null** را در صورتیکه سرویس‌دهنده، متغیر محیطی **QUERY_STRING** را تنظیم نکرده باشد، برگشت می‌دهد. اگر این متغیر محیطی وجود داشته باشد (یعنی **getenv** اشاره‌گر **null** برگشت ندهد)، خط 17 مجزا **getenv** را فراخوانی می‌کند. این بار رشته پرس‌وجوی برگشتی به متغیر رشته‌ای **query** تخصیص داده می‌شود. پس از قرار دادن سرآیند و برخی از دنباله‌های شروع XHTML و عنوان (خطوط 19-29) به بررسی اینکه در **query** داده وجود دارد یا خیر می‌پردازیم (خط 32). اگر چنین نباشد، پیغامی به کاربر عرضه می‌شود و از وی می‌خواهد تا یک رشته پرس‌وجو به URL اضافه کند. همچنین یک لینک به URL افزوده‌ایم که شامل یک رشته پرس‌وجوی ساده است. داده رشته پرس‌وجو می‌تواند تعیین کننده یک فوق لینک در صفحه وب در زمان کدگشایی باشد. محتویات رشته پرس‌وجو توسط خط 36 چاپ می‌شود.

این مثال ساده به توصیف نحوه دسترسی داده ارسالی به اسکریپت CGI در رشته پرس‌وجو پرداخته است. در ادامه این فصل با مثال‌های آشنا خواهید شد که نحوه تقسیم یک رشته پرس‌وجو به قسمتهای سودمندتر اطلاعاتی را نشان می‌دهند که می‌توان با استفاده از متغیرهای مجزا از آنها نگهداری کرد.

۱۱-۱۹ استفاده از فرم‌های XHTML برای ارسال ورودی

داشتن سرویس‌گیرنده‌های که مستقیماً ورودی را وارد URL می‌کند روش چندان کاربرپسندی نیست. خوشبختانه، XHTML قابلیتی را از طریق فرم‌ها در صفحات وب فراهم آورده که می‌تواند روش بسیار مناسبی برای کاربران در وارد کردن اطلاعاتی باشند که به یک اسکریپت CGI ارسال خواهند شد.

```
1  // Fig. 19.9: querystring.cpp
2  // Demonstrating QUERY_STRING.
3  #include <iostream>
4  using std::cout;
5
6  #include <string>
7  using std::string;
8
9  #include <cstdlib>
10 using::getenv;
11
12 int main()
13 {
14    string query = "";
15
16    if ( getenv( "QUERY_STRING" ) ) // QUERY_STRING variable exists
17       query = getenv( "QUERY_STRING" );//retrieve QUERY_STRING value
18
19    cout << "Content-Type: text/html\n\n"; // output http header
20
21    // output XML declaration and DOCTYPE
22    cout << "<?xml version = \"1.0\"?>"
23       << "<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.1//EN\" "
24       << "\"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd\">";
25
```

```
26    // output html element and some of its contents
27    cout << "<html xmlns = \"http://www.w3.org/1999/xhtml\">"
28       << "<head><title>Name/Value Pairs</title></head><body>";
29    cout << "<h2>Name/Value Pairs</h2>";
30
31    // if query contained no data
32    if ( query == "" )
33       cout<<"Please add some name-value pairs to the URL above.<br/> Or"
34         <<" try <a href=\"querystring.cgi?name=Joe&age=29\">this</a>.";
35    else // user entered query string
36       cout << "<p>The query string is: " << query << "</p>";
37
38    cout << "</body></html>";
39    return 0;
40 } // end main
```

شکل ۹-۱۹ | خواندن ورودی از QUERY_STRING.

عنصر form

عنصر **form** یک فرم XHTML بوجود می‌آورد. معمولاً این عنصر دو صفت دریافت می‌کند. صفت اول **action** است که مشخص کننده منبع سرویس‌دهنده برای اجرا شدن در زمانی است که کاربر فرم را تسلیم می‌کند. برای اهداف ما، معمولاً **action** یک اسکریپت CGI خواهد بود که داده فرم را پردازش می‌نماید. صفت دوم که در عنصر **form** بکار گرفته می‌شود، **method** است، که شناسه تقاضای HTTP می‌باشد (یعنی **get** یا **post**) تا به هنگام تسلیم فرم توسط مرورگر به سرویس‌دهنده وب بکار گرفته شود. در این بخش مثال‌های با استفاده از هر دو نوع تقاضای **get** و **post** مطرح کرده‌ایم. یک فرم XHTML می‌تواند حاوی هر تعداد از عناصر باشد. جدول شکل ۱۰-۱۹ بطور خلاصه به معرفی چند عنصر فرم پرداخته است.

نام عنصر	نوع صفت	توضیح
input	**text**	یک فیلد تک خطی برای وارد کردن متن فراهم می‌آورد.
	password	همانند **text** است، اما هر کاراکتر تایپ شده را بصورت ستاره (*) ظاهر می‌کند.
	checkbox	یک جعبه‌چک به نمایش در می‌آورد که می‌تواند انتخاب شود **(true)** یا از انتخاب خارج گردد **(false)**.
	radio	دکمه‌های رادیوی همانند جعبه‌چک‌ها هستند بجز اینکه فقط یک دکمه رادیویی در گروه دکمه‌های رادیویی می‌تواند در هر بار انتخاب شود.
	button	یک دکمه به نمایش در می‌آورد.
	submit	یک دکمه است که داده فرم را مطابق **action** فرم تسلیم می‌کند.
	image	همانند **submit** است، اما بجای دکمه یک تصویر به نمایش در می‌آورد.
	reset	یک دکمه به نمایش در می‌آورد که فیلدهای فرم را به مقادیر پیش‌فرض آنها باز می‌گرداند.
	file	یک فیلد متنی و دکمه به نمایش در می‌آورد که به کاربر امکان می‌دهد تا فایلی

برنامه‌نویسی وب ـــــــ

```
27
28    // output html element and some of its contents
29    cout << "<html xmlns = \"http://www.w3.org/1999/xhtml\">"
30       << "<head><title>Using GET with Forms</title></head><body>";
31
32    // output xhtml form
33    cout << "<p>Enter one of your favorite words here:</p>"
34       << "<form method = \"get\" action = \"getquery.cgi\">"
35       << "<input type = \"text\" name = \"word\"/>"
36       << "<input type = \"submit\" value = \"Submit Word\"/></form>";
37
38    if ( query == "" ) // query is empty
39       cout << "<p>Please enter a word.</p>";
40    else // user entered query string
41    {
42       int wordLocation = query.find_first_of( "word=" ) + 5;
43       wordString = query.substr( wordLocation );
44
45       if ( wordString == "" ) // no word was entered
46          cout << "<p>Please enter a word.</p>";
47       else // word was entered
48          cout << "<p>Your word is: " << wordString << "</p>";
49    } // end else
50
51    cout << "</body></html>";
52    return 0;
53 } // end main
```

شکل ۱۱-۱۹ | استفاده از روش get به همراه فرم XHTML.

زمانیکه کاربر عبارتی را در فیلد متنی word وارد و بر روی دکمه Submit Word کلیک کرد، اسکریپت مجدداً تقاضا می‌شود. این بار، نام فـی‌لد ورودی (word) و مقدار وارد شده توسط کاربر در متغیر محیطی **QUERY_STRING** جای خواهند داشت. یعنی اگر کاربر کلمه "technology" را وارد و بر روی Submit Word کلیک کند به **QUERY_STRING** مقدار **value=technology** تخصیص می‌یابد. دقت کنید که رشته پرس‌وجو به URL در فیلد Address مرورگر به همراه علامت سوال (**?**) قبل از آن افزوده می‌شود.

در اجرای دوم اسکریپت، رشته پرس‌وجو کدگشایی می‌شود. خط 42 از متد **find_first_of** برای جستجوی پرس‌وجو به منظور یافتن اولین پیشامد از **=word** استفاده کرده است، که یک مقدار صحیح برگشت می‌دهد که نشاندهنده موقعیت آن در رشته است. سپس خط 42 مقدار 5 را به مقدار برگشتی توسط **find_first_of** اضافه می‌کند تا **wordLocation** با موقعیت اولین کاراکتر وارد شده توسط کاربر تنظیم شود. تابع **substr** در خط 43 مابقی رشته آغاز شونده از **wordLocation** را برگشت می‌دهد. خط 45 تعیین می‌کند که کاربر کلمه‌ای وارد کرده است یا خیر. اگر چنین باشد، خط 48 کلمه وارد شده توسط کاربر را چاپ می‌کند.

تقاضای post

در دو مثال قبلی از روش **get** برای ارسال داده به اسکریپت‌های CGI از یک متغیر محیطی استفاده کردیم. عموماً مرورگرهای وب با سرویس‌دهنده‌های وب توسط فرم‌های تسلیم شده به روش **HTTP post** در تعامل قرار می‌گیرند. برنامه‌های CGI محتویات تقاضاهای **post** را با استفاده از استاندارد ورودی می‌خوانند. برای اینکه مقایسه‌ای انجام دهیم، اجازه دهید مجدداً برنامه ۱۱-۱۹ را با استفاده از روش **post** پیاده‌سازی کنیم (بعنوان برنامه شکل ۱۲-۱۹). توجه کنید کد بکار رفته در دو برنامه واقعاً یکسان هستند فرم XHTML در خطوط 45-43 نشان می‌دهد که در حال استفاد از روش **post** به منظور تسلیم داده فرم هستیم.

```
1 // Fig. 19.12: post.cpp
2 // Demonstrates POST method with XHTML form.
3 #include <iostream>
4 using std::cout;
5 using std::cin;
6
7 #include <string>
8 using std::string;
9
10 #include <cstdlib>
11 using std::getenv;
12 using std::atoi;
13
14 int main()
15 {
16    char postString[ 1024 ] = ""; // variable to hold POST data
17    string dataString = "";
18    string nameString = "";
19    string wordString = "";
20    int contentLength = 0;
21
22    // content was submitted
23    if ( getenv( "CONTENT_LENGTH" ) )
24    {
25       contentLength = atoi( getenv( "CONTENT_LENGTH" ) );
26       cin.read( postString, contentLength );
27       dataString = postString;
28    } // end if
29
30    cout << "Content-Type: text/html\n\n"; // output header
31
32    // output XML declaration and DOCTYPE
33    cout << "<?xml version = \"1.0\"?>"
34       << "<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.1//EN\" "
35       << "\"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd\">";
36
37    // output XHTML element and some of its contents
38    cout << "<html xmlns = \"http://www.w3.org/1999/xhtml\">"
39       << "<head><title>Using POST with Forms</title></head><body>";
40
41    // output XHTML form
42    cout << "<p>Enter one of your favorite words here:</p>"
43       << "<form method = \"post\" action = \"post.cgi\">"
44       << "<input type = \"text\" name = \"word\" />"
45       << "<input type = \"submit\" value = \"Submit Word\" /></form>";
46
47    // data was sent using POST
```

```
48    if ( contentLength > 0 )
49    {
50       int nameLocation = dataString.find_first_of( "word=" ) + 5;
51       int endLocation = dataString.find_first_of( "&" ) - 1;
52
53       // retrieve entered word
54       wordString = dataString.substr(
55          nameLocation, endLocation - nameLocation );
56
57       if ( wordString == "" ) // no data was entered in text field
58          cout << "<p>Please enter a word.</p>";
59       else // output word
60          cout << "<p>Your word is: " << wordString << "</p>";
61    } // end if
62    else // no data was sent
63       cout << "<p>Please enter a word.</p>";
64
65    cout << "</body></html>";
66    return 0;
67 } // end main
```

شکل ۱۲-۱۹ | استفاده از روش post به همراه فرم XHTML.

سرویس‌دهنده وب مبادرت به ارسال داده **post** به اسکریپت CGI از طریق ورودی استاندارد می‌کند. داده همانند رشته **QUERY_STRING** کدگشایی می‌شود، اما متغیر محیطی **QUERY_STRING** تنظیم نمی‌شود. بجای آن، روش **post** اقدام به تنظیم متغیر محیطی **CONTENT_LENGTH** می‌کند تا نشاندهنده تعداد کاراکترهای داده باشد که همراه تقاضای post ارسال شده‌اند.

اسکریپت CGI از مقدار متغیر محیطی **CONTENT_LENGTH** برای پردازش حجم صحیحی از داده‌ها استفاده می‌کند. در اینصورت، خط 25 مقدار را خوانده و آنرا با فراخوانی تابع **atoi** تبدیل به مقدار صحیح (integer) می‌کند. خط 26 تابع **cin.read** را برای خواندن کاراکترها از ورودی استاندارد و ذخیره کاراکترها در آرایه **postString** فراخوانی می‌نماید. خط 27 داده **postString** را به یک رشته با تخصیص آن به **dataString** تبدیل می‌کند.

در فصل‌های اولیه، داده را از ورودی استاندارد و با استفاده از عبارتی همانند

```
cin >> data;
```

می‌خواندیم. همین روش در ارتباط با اسکریپت CGI کاربرد دارد که به همین منظور از عبارت **cin.read** استفاده کرده‌ایم. بخاطر دارید که **cin** داده را از ورودی استاندارد تا رسیدن به اولین کاراکتر خط جدید (newline)، فاصله یا tab هر کدام زودتر دیده شود، می‌خواند. ساختار CGI مستلزم افزوده شدن خط جدید پس از آخرین جفت نام–مقدار نیست. با اینکه برخی از مرورگرها یک خط جدید یا **EOF** الصاق می‌کنند، اما نیازی به انجام اینکار نیست. اگر **cin** با مرورگری بکار رود که فقط جفت‌های نام–مقدار را اضافه می‌کند، **cin** بایستی منتظر خط جدید باشد که هرگز نخواهد رسید. در چنین حالتی، سرویس‌دهنده شروع به شمارش زمان کرده و عاقبت اسکریپت CGI خاتمه می‌یابد. از اینرو، **cin.read** بر **cin** ترجیح داده می‌شود، چرا که برنامه‌نویس می‌تواند مقدار دقیق خواندن داده را تعیین کند.

۱۲-۱۹ سرآیندهای دیگر

یک اسکریپت CGI می‌تواند سرآیندهای دیگر HTTP را در کنار **Content-Type** بکار گیرد. در بسیاری از موارد، سرویس‌دهنده این سرآیندهای اضافی را به سرویس‌گیرنده ارسال می‌کند بدون اینکه آنها را اجرا کند. برای مثال، سرآیند **Refresh** در عبارت زیر مبادرت به هدایت سرویس‌گیرنده به مکان جدید پس از تعیین زمان مشخص می کند:

```
Referesh:"5: URL = http://www.deitel.com/newpage.html."
```

پنج ثانیه پس از اینکه مرورگر وب این سرآیند را دریافت کرد، مرورگر تقاضای منبع مشخص شده در URL را می‌کند. بطور جایگزین، سرآیند **Referesh** می‌تواند URL را نادیده بگیرد، که در اینحالت صفحه جاری پس از سپری شدن زمان، نوسازی می‌گردد.

ساختار CGI بر این نکته دلالت دارد که سرآیندها از نوع‌های خاص بجای آنکه مستقیماً به سرویس‌گیرنده ارسال شوند، توسط سرویس‌دهنده پردازش می‌شوند. اولین این سرآیندها، سرآیند **Location** است. همانند **Referesh**، این سرآیند، سرویس‌گیرنده را به مکان جدید هدایت می‌کند:

```
Location: http:/www.deitel.com/newpage.html
```

اگر به همراه یک URL نسبی (یا مجازی) بکار رود (یعنی Location:/newpage.html)، سرآیند **Location** به سرویس‌دهنده نشان می‌دهد که جهت حرکت در طرف سرویس‌دهنده انجام می‌شود بدون اینکه سرآیند **Location** به سرویس‌گیرنده برگردانده شود. در اینحالت، مستند بصورت راندو شده در مرورگر وب ظاهر می‌شود.

همچنین ساختار CGI شامل سرآیند **Status** است که به سرویس‌دهنده دستور می‌دهد تا خط وضعیت را بکار گیرد (همانند HTTP/1.1 200 OK). معمولاً سرویس‌دهنده خط وضعیت متناسب را به سرویس‌گیرنده ارسال می‌کند. با این همه، CGI به برنامه‌نویسان امکان داده تا تغییری در پاسخ وضعیت بوجود آورند. برای مثال، ارسال این سرآیند

```
status: 204 No Response
```

نشان می‌دهد که اگرچه تقاضا با موفقیت صورت گرفته، اما سرویس‌گیرنده، نمی‌تواند صفحه جدید را در پنجره مرورگر به نمایش در آورد.

به طور خلاصه CGI به اسکریپت‌ها اجازه می‌دهد تا با سرویس‌دهنده‌ها به سه روش در تعامل قرار گیرند:

۱- از طریق ارسال سرآیندها و محتویات به سرویس‌دهنده از طریق خروجی استاندارد.

۲- با تنظیم متغیرهای محیطی سرویس‌دهنده، که مقادیر آنها در درون اسکریپت قابل استفاده است.

۳- از طریق **POSTed**، کدگشایی داده URL که سرویس‌دهنده به ورودی استاندارد اسکریپت ارسال می‌کند.

۱۳-۱۹ مبحث آموزشی: صفحه وب تعاملی

برنامه شکل‌های ۱۳-۱۹ و ۱۴-۱۹ نحوه پیاده‌سازی یک *سردر*(*portal*) تعاملی برای وب سایت ساختگی Bug2Bug Travel را نشان می‌دهند. در این برنامه از سرویس‌گیرنده در ارتباط با نام و کلمه عبور سئوال شده، سپس اطلاعاتی در مورد سفر هفتگی براساس داده‌های وارد شده به نمایش در می‌آورد. برای ساده‌تر شدن کار، این مثال رمزگذاری بر روی داده‌های ارسالی به سرویس‌دهنده را انجام نمی‌دهد. بهتر است که داده‌های با اهمیت همانند کلمات عبور، حتماً رمزگذاری شوند. مبحث رمزگذاری خارج از قلمرو آموزشی این کتاب است.

برنامه شکل ۱۳-۱۹ صفحه آغازین را نشان می‌دهد. این صفحه یک مستند استاتیکی XHTML است که حاوی یک فرم بوده و داده‌ها را به اسکریپت CGI بنام **portal.cgi** پست می‌کند (خط 16). فرم حاوی یک فیلد برای دریافت نام کاربر (خط 18) و یکی برای دریافت کلمه عبور است (خط 19). [نکته: برخلاف اسکریپت‌های CGI که در شاخه **cgi-bin** سرویس‌دهنده وب جای داده می‌شوند، این مستند XHTML در شاخه **hotdocs** سرویس‌دهنده وب قرار داده شده است.]

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
3     "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
4
5  <!-- Fig. 19.13: travel.html   -->
6  <!-- Bug2Bug Travel Homepage   -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title>Bug2Bug Travel</title>
11    </head>
12
13    <body>
14       <h1>Welcome to Bug2Bug Travel</h1>
15
16       <form method = "post" action = "/cgi-bin/portal.cgi">
17          <p>Please enter your name and password:</p>
18          <input type = "text" name = "namebox" />
19          <input type = "password" name = "passwordbox" />
20          <p>password is not encrypted</p>
21          <input type = "submit" name = "button" />
22       </form>
23    </body>
24 </html>
```

شکل ۱۳-۱۹ | سردر تعاملی برای ایجاد یک صفحه وب همراه با فیلد کلمه عبور.

برنامه شکل ۱۴-۱۹ حاوی اسکریپت CGI است. ابتدا، اجازه دهید تا به بررسی نحوه بازیابی نام و کلمه عبور کاربر از ورودی استاندارد و ذخیره آنها در رشته‌ها بپردازیم. تابع **find** از کلاس **string**، مبادرت به جستجوی **dataString** در خط 30 برای یافتن اولین پیشامد برای **namebox=** می‌کند. این تابع موقعیت قرارگیری **namebox=** را در رشته برگشت می‌دهد. برای بازیابی مقدار مرتبط با **namebox** یعنی مقدار

وارد شده توسط کاربر، مبادرت به انتقال موقعیت در رشته به میزان 8 کاراکتر به سمت جلو کرده‌ایم. اکنون برنامه حاوی یک مقدار صحیح است که به موقعیت شروع اشاره می‌کند. بخاطر دارید که رشته پرس‌وجو حاوی جفت‌های نام–مقدار است که توسط نمادهای تساوی و آمپرسنج (**&**) از یکدیگر متمایز می‌شوند. برای یافتن انتهای موقعیت داده، بدنبال کاراکتر **&** جستجوی انجام می‌دهیم (خط 31). طول کلمه وارد شده با عبارت محاسباتی **endNamelocation - namelocation** تعیین می‌شود. از روش مشابهی برای تعیین موقعیت شروع و پایانی کلمه عبور استفاده کرده‌ایم (خطوط 33-32). خطوط 38-36 مقادیر فیلدها را به متغیرهای **nameString** و **passwordString** تخصیص می‌دهند. از **nameString** در خط 52 برای چاپ یک پیغام خوش آمدگویی شخصی استفاده کرده‌ایم. سفر جاری توسط خطوط 66-53 به نمایش در می‌آید.

```
1  // Fig. 19.14: portal.cpp
2  // Handles entry to Bug2Bug Travel.
3  #include <iostream>
4  using std::cout;
5  using std::cin;
6
7  #include <string>
8  using std::string;
9
10 #include <cstdlib>
11 using std::getenv;
12 using std::atoi;
13
14 int main()
15 {
16    char postString[ 1024 ] = "";
17    string dataString = "";
18    string nameString = "";
19    string passwordString = "";
20    int contentLength = 0;
21
22    // data was posted
23    if ( getenv( "CONTENT_LENGTH" ) )
24       contentLength = atoi( getenv( "CONTENT_LENGTH" ) );
25
26    cin.read( postString, contentLength );
27    dataString = postString;
28
29    // search string for input data
30    int namelocation = dataString.find( "namebox=" ) + 8;
31    int endNamelocation = dataString.find( "&" );
32    int password = dataString.find( "passwordbox=" ) + 12;
33    int endPassword = dataString.find( "&button" );
34
35    // get values for name and password
36    nameString = dataString.substr(
37       namelocation, endNamelocation - namelocation );
38   passwordString = dataString.substr( password,endPassword - password );
39
40    cout << "Content-Type: text/html\n\n"; // output HTTP header
41
42    // output XML declaration and DOCTYPE
```

```
43    cout << "<?xml version = \"1.0\"?>"
44       << "<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.1//EN\" "
45       << "\"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd\">";
46
47    // output html element and some of its contents
48    cout << "<html xmlns = \"http://www.w3.org/1999/xhtml\">"
49       << "<head><title>Bug2Bug Travel</title></head><body>";
50
51    // output specials
52    cout << "<h1>Welcome " << nameString << "!</h1>"
53       << "<p>Here are our weekly specials:</p>"
54       << "<ul><li>Boston to Taiwan ($875)</li>"
55       << "<li>San Diego to Hong Kong ($750)</li>"
56       << "<li>Chicago to Mexico City ($568)</li></ul>";
57
58    if ( passwordString == "coast2coast" ) // password is correct
59       cout << "<hr /><p>Current member special: "
60          << "Seattle to Tokyo ($400)</p>";
61    else // password was incorrect
62       cout << "<p>Sorry. You have entered an incorrect password</p>";
63
64    cout << "</body></html>";
65    return 0;
66 } // end main
```

شکل ١٤-١٩ | سردر تعاملی صفحه.

اگر کلمه عبور عضویت صحیح باشد، خطوط 60-59 موارد استثنایی و خاصی را به نمایش در می‌آورند. اگر کلمه عبور اشتباه باشد، سرویس‌گیرنده مطلع می‌شود که کلمه عبور معتبر نیست و موارد استثنایی و خاص برای آن کاربر به نمایش در نمی‌آید.

توجه کنید که از یک صفحه استاتیک و یک اسکریپت CGI مجزا استفاده کرده‌ایم. البته می‌توانستیم هر دو کار را در یک اسکریپت CGI انجام دهیم.

١٤-١٩ کوکی

یکی از روش‌های پرکاربرد بهینه‌سازی تعامل‌های صورت گرفته با صفحات وب از طریق *Cookies* (*کوکی‌ها*) است. کوکی یک فایل متنی ذخیره شده توسط سایت وب بر روی هر کامپیوتر جداگانه است که به سایت اجازه می‌دهند تا فعالیت‌های بازدید کننده را ردگیری نماید. اولین باری که کاربر از سایت وب بازدید می‌کند، کامپیوتر کاربر یک فایل کوکی دریافت می‌کند، این کوکی هر بار که کاربر از آن سایت بازدید نماید، فعال می‌شود. اطلاعات جمع‌آوری شده بصورت یک رکورد بی‌نام هستند و حاوی داده‌ای می‌باشند که برای شخصی‌سازی محیط سایت بکار گرفته می‌شوند برای مثال، کوکی‌های موجود در برنامه‌های خرید، ممکن است هویت منحصر بفرد کاربران را ذخیره نمایند. هنگامی کاربر اقدام به افزودن ایتم‌های به کارت خرید *online* می‌کند یا اعمال دیگری انجام می‌دهد که نتیجه یک تقاضا از سرویس‌دهنده وب است، سرویس‌دهنده، کوکی را که حاوی اطلاعات منحصر بفرد کاربر است دریافت می‌نماید. سپس سرویس‌دهنده با استفاده از این اطلاعات، پردازش‌های مورد نیاز را انجام می‌دهد.

علاوه بر هویت بخشیدن به کاربران، کوکی‌ها می‌توانند دلالت بر سلایق مشتری‌ها نیز باشند. هنگامی یک برنامه وب، تقاضای از سوی یک سرویس‌گیرنده دریافت می‌کند، فرم وب می‌تواند اقدام به بررسی کوکی(هایی) ارسالی در دفعات قبل نماید و بلافاصله سلیقه مشتری را تشخیص داده و محصولات و سرویس‌های مطابق با آن سلیقه به نمایش در آورد.

به عنوان یک برنامه‌نویس، بایستی مطلع باشید که سرویس‌گیرنده‌ها می‌توانند کوکی‌ها را غیرفعال نمایند. در برنامه شکل‌های ۱۵-۱۹ الی ۱۷-۱۹ از کوکی‌ها برای ذخیره‌سازی و نگهداری اطلاعاتی در ارتباط با کاربر استفاده شده است.

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
3     "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
4
5  <!-- Fig. 19.15: cookieform.html -->
6  <!-- Cookie Demonstration        -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title>Writing a cookie to the client computer</title>
11    </head>
12
13    <body>
14       <h1>Click Submit to save your cookie data.</h1>
15
16       <form method = "post" action = "/cgi-bin/writecookie.cgi">
17          <p>Name:<br />
18             <input type = "text" name = "name" />
19          </p>
20          <p>Age:<br />
21             <input type = "text" name = "age" />
22          </p>
23          <p>Favorite Color:<br />
24             <input type = "text" name = "color" />
25          </p>
26          <p>
27             <input type = "submit" name = "button" value="Submit"/>
28          </p>
29       </form>
30    </body>
31 </html>
```

شکل ۱۵-۱۹ | مستند XHTML حاوی فرمی برای پست داده به سرویس‌دهنده.

```
1  // Fig. 19.16: writecookie.cpp
2  // Program to write a cookie to a client's machine.
3  #include <iostream>
4  using std::cin;
5  using std::cout;
6
7  #include <string>
8  using std::string;
9
10 #include <cstdlib>
11 using std::getenv;
12 using std::atoi;
13
```

```
14 int main()
15 {
16    char query[ 1024 ] = "";
17    string dataString = "";
18    string nameString = "";
19    string ageString = "";
20    string colorString = "";
21    int contentLength = 0;
22
23    // expiration date of cookie
24    string expires = "Friday, 14-MAY-10 16:00:00 GMT";
25
26    // data was entered
27    if ( getenv( "CONTENT_LENGTH" ) )
28    {
29       contentLength = atoi( getenv( "CONTENT_LENGTH" ) );
30       cin.read(query,contentLength);//read data from standard input
31       dataString = query;
32
33       // search string for data and store locations
34       int nameLocation = dataString.find( "name=" ) + 5;
35       int endName = dataString.find( "&" );
36       int ageLocation = dataString.find( "age=" ) + 4;
37       int endAge = dataString.find( "&color" );
38       int colorLocation = dataString.find( "color=" ) + 6;
39       int endColor = dataString.find( "&button" );
40
41       // get value for user's name
42       nameString = dataString.substr(
43          nameLocation, endName - nameLocation );
44
45       if ( ageLocation > 0 ) // get value for user's age
46          ageString = dataString.substr(
47             ageLocation, endAge - ageLocation );
48
49       if ( colorLocation > 0 )// get value for user's favorite color
50          colorString = dataString.substr(
51             colorLocation, endColor - colorLocation );
52
53       // set cookie
54       cout << "Set-Cookie: Name=" << nameString << "age:"
55          << ageString << "color:" << colorString
56          << "; expires=" << expires << "; path=\n";
57    } // end if
58
59    cout << "Content-Type: text/html\n\n"; // output HTTP header
60
61    // output XML declaration and DOCTYPE
62    cout << "<?xml version = \"1.0\"?>"
63       << "<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.1//EN\" "
64       << "\"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd\">";
65
66    // output html element and some of its contents
67    cout << "<html xmlns = \"http://www.w3.org/1999/xhtml\">"
68       << "<head><title>Cookie Saved</title></head><body>";
69
70    // output user's information
71    cout << "<p>A cookie has been set with the following"
72       << " data:</p><p>Name: " << nameString  << "<br/></p>"
73       << "<p>Age: " << ageString << "<br/></p>"
74       << "<p>Color: " << colorString << "<br/></p>"
75       << "<p>Click <a href=\"/cgi-bin/readcookie.cgi\">"
```

```
76       << "here</a> to read saved cookie data.</p></body></html>";
77    return 0;
78 } // end main
```

شکل ۱۶-۱۹ | نوشتن کوکی.

برنامه شکل ۱۵-۱۹ یک صفحه XHTML است که حاوی یک فرم بوده و مقادیری از طریق آن وارد می‌شوند. فرم مبادرت به ارسال اطلاعات به **writecookie.cgi** می‌کند (شکل ۱۶-۱۹). این اسکریپت CGI داده موجود در متغیر **CONTENT_LENGTH** را بازیابی می‌کند.

خط 24 از شکل ۱۶-۱۹ مبادرت به اعلان و مقداردهی اولیه رشته **expires** برای ذخیره‌سازی تاریخ انقضا کوکی می‌کند که تعیین کننده مدت زمانی است که کوکی می‌تواند بر روی ماشین سرویس‌گیرنده مقیم باشد. این مقدار می‌تواند یک رشته باشد، همانند مقداری که در این مثال بکار گرفته شده است، یا می‌تواند یک مقدار نسبی باشد. برای نمونه "**+30d**" مبادرت به تنظیم تاریخ انقضا کوکی پس از 30 روز می‌کند. برای برآورده کردن اهداف این فصل، تاریخ انقضا عمداً با سال 2010 تنظیم شده تا مطمئن گردیم که برنامه بخوبی درآیند اجرا خواهد شد. البته می‌توانید تاریخ انقضا در این مثال را به هر تاریخی که مایل هستید، تغییر دهید. پس از انقضا کوکی، مرورگر آنها را حذف خواهد کرد.

پس از بدست آوردن داده از فرم، برنامه یک کوکی ایجاد می‌کند (خطوط 56-54). در این مثال، یک کوکی ایجاد می‌کنیم که یک خط متنی حاوی جفت‌های نام–مقدار از داده پست شده که توسط کولن (:) از هم متمایز شده‌اند را ذخیره سازد. این خط بایستی قبل از اینکه سرآیند در سرویس‌گیرنده نوشته شود، خارج گردد. خط متنی با سرآیند **Set-Cookie:** آغاز می‌شود و نشان می‌دهد که مرورگر بایستی داده‌های ورودی را در یک کوکی ذخیره سازد. در این برنامه مبادرت به تنظیم سه صفت برای کوکی کرده‌ایم: یک جفت نام–مقدار حاوی داده که ذخیره خواهد شد، یک جفت نام–مقدار حاوی تاریخ انقضا و یک جفت نام–مقدار حاوی URL از دامنه سرویس‌دهنده (همانند www.deitel.com) برای اینکه کوکی معتبر باشد. در این مثال، **path** با هیچ مقداری تنظیم نشده است و از اینرو کوکی از طریق هر سرویس‌دهنده‌ای در دامنه سرویس‌دهنده که کوکی در آن نوشته شده است، قابل خواندن است. توجه کنید که جفت‌های نام–مقدار توسط سیمکولن از یکدیگر جدا شده‌اند. ما فقط از کاراکترهای کولن در درون داده کوکی خود استفاده کرده‌ایم تا تداخلی با قالب سرآیند **Set-Cookie:** بوجود نیاید. زمانیکه همان داده‌های به نمایش درآمده در شکل ۱۵-۱۹ را وارد کنیم، خطوط 56-54 داده "Name=Zoeage:24color:Red" را در کوکی ذخیره می‌کنند. خطوط 57-59 صفحه وبی ارسال می‌کنند تا نشان داده شود کوکی در سرویس‌گیرنده نوشته شده است.

برنامه شکل ۱۷-۱۹ کوکی نوشته شده در برنامه ۱۶-۱۹ را خوانده و اطلاعات ذخیره شده در آنرا به نمایش در می‌آورد. زمانیکه سرویس‌گیرنده تقاضای به سرویس‌دهنده ارسال می‌کند، مرورگر وب سرویس‌گیرنده بدنبال هر کوکی نوشته شده از سوی آن سرویس‌دهنده می‌گردد. این کوکی‌ها توسط مرورگر به سرویس‌دهنده بعنوان بخشی از تقاضا بازپس فرستاده می‌شوند. بر روی سرویس‌دهنده، متغیر محیطی **HTTP_COOKIE** کوکی‌های سرویس‌گیرنده را ذخیره می‌کند. خط 20 تابع **getenv** را با متغیر محیطی **HTTP_COOKIE** بعنوان پارامتر فراخوانی کرده و مقدار برگشتی را در **dataString** ذخیره می‌کند. جفت‌های نام–مقدار کدگشایی شده و رشته‌های مطابق با طرح کدگشایی بکار رفته در برنامه شکل ۱۶-۱۹ ذخیره می‌شوند (خط 24-23). خطوط 55-36 محتویات کوکی را در صفحه چاپ می‌کنند.

```
1  // Fig. 19.17: readcookie.cpp
2  // Program to read cookie data.
3  #include <iostream>
4  using std::cin;
5  using std::cout;
6
7  #include <string>
8  using std::string;
9
10 #include <cstdlib>
11 using std::getenv;
12
13 int main()
14 {
15    string dataString = "";
16    string nameString = "";
17    string ageString = "";
18    string colorString = "";
19
20    dataString = getenv( "HTTP_COOKIE" ); // get cookie data
21
22    // search through cookie data string
23    int nameLocation = dataString.find( "Name=" ) + 5;
24    int endName = dataString.find( "age:" );
25    int ageLocation = dataString.find( "age:" ) + 4;
26    int endAge = dataString.find( "color:" );
27    int colorLocation = dataString.find( "color:" ) + 6;
28
29    // store cookie data in strings
30    nameString = dataString.substr(
31       nameLocation, endName - nameLocation );
32    ageString = dataString.substr(
33       ageLocation, endAge - ageLocation );
34    colorString = dataString.substr( colorLocation );
35
36    cout << "Content-Type: text/html\n\n"; // output HTTP header
37
38    // output XML declaration and DOCTYPE
39    cout << "<?xml version = \"1.0\"?>"
40       << "<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.1//EN\" "
41       << "\"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd\">";
42
```

```
43    // output html element and some of its contents
44    cout << "<html xmlns = \"http://www.w3.org/1999/xhtml\">"
45         << "<head><title>Read Cookies</title></head><body>";
46
47    if ( dataString != "" ) // data was found
48       cout << "<h3>The following data is saved in a cookie on"
49         <<" your computer</h3><p>Name:" << nameString <<"<br/></p>"
50          << "<p>Age: " << ageString << "<br/></p>"
51          << "<p>Color: " << colorString << "<br/></p>";
52    else // no data was found
53       cout << "<p>No cookie data.</p>";
54
55    cout << "</body></html>";
56    return 0;
57 } // end main
```

شکل ۱۷-۱۹ | برنامه، کوکی‌های ارسالی از کامپیوتر سرویس‌گیرنده را می‌خواند.

۱۵-۱۹ فایل‌های طرف سرویس‌گیرنده

در بخش قبلی، به توصیف نحوه نگهداری اطلاعات وضعیت که در ارتباط با کاربر هستند از طریق کوکی‌ها پرداختیم. مکانیزم‌های دیگری برای انجام اینکار وجود دارند که ایجاد فایلهای طرف سرویس‌گیرنده (*server-side files*) است، یعنی فایل‌های که بر روی سرویس‌گیرنده قرار داده می‌شوند یا بر روی شبکه سرویس‌دهنده. این روش تا حدی از امنیت بیشتر برخوردار است و مناسب نگهداری اطلاعات مهمتر است. در این مکانیزم، فقط یکنفر با داشتن مجوز دسترسی قادر به تغییر در فایل‌های موجود بر روی سرویس‌دهنده است. برنامه شکل‌های ۱۸-۱۹ و ۱۹-۱۹ از کاربران می‌خواهد تا اطلاعات تماس را وارد کرده، سپس آنرا بر روی سرویس‌دهنده ذخیره می‌کنند. شکل ۲۰-۱۹ فایلی را که توسط اسکریپت ایجاد شده، به نمایش در آورده است.

مستند XHTML در شکل ۱۸-۱۹ داده فرم را به اسکریپت CGI در شکل ۱۹-۱۹ پست می‌کند. در اسکریپت CGI، خطوط 92-45 پارامترهای که توسط سرویس‌گیرنده ارسال شده‌اند را کدگشایی می‌کنند. خط 105 نمونه‌ای از استریم فایل خروجی ایجاد می‌کند (*out File*) که فایلی را برای الصاق کردن باز می‌کند. اگر فایل **clients.txt** وجود نداشته باشد، آنرا ایجاد می‌کند. خطوط 116-114 اطلاعات شخصی را در فایل قرار می‌دهند.

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
3     "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
4
5  <!-- Fig. 19.18: savefile.html           -->
6  <!-- Form to input client information    -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title>Please enter your contact information</title>
11    </head>
12
13    <body>
14       <p>Please enter your information in the form below.</p>
```

```
15       <p>Note: You must fill in all fields.</p>
16       <form method = "post" action = "/cgi-bin/savefile.cgi">
17          <p>
18             First Name:
19             <input type = "text" name = "firstname" size = "10" />
20             Last Name:
21             <input type = "text" name = "lastname" size = "15" />
22          </p>
23          <p>
24             Address:
25             <input type = "text" name = "address" size="25"/><br />
26             Town: <input type = "text" name = "town" size = "10" />
27             State: <input type = "text" name="state"size="2"/><br/>
28             Zip Code: <input type = "text" name="zipcode"size="5" />
29             Country: <input type = "text" name="country"size="10" />
30          </p>
31          <p>
32             E-mail Address: <input type = "text" name = "email" />
33          </p>
34             <input type = "submit" value = "Enter" />
35             <input type = "reset" value = "Clear" />
36       </form>
37    </body>
38 </html>
```

شکل ۱۸-۱۹ | مستند XHTML اطلاعات تماس کاربر را می‌خواند.

```
1  // Fig. 19.19: savefile.cpp
2  // Program to enter user's contact information into a
3  // server-side file.
4  #include <iostream>
5  using std::cerr;
6  using std::cin;
7  using std::cout;
8  using std::ios;
9
10 #include <fstream>
11 using std::ofstream;
12
13 #include <string>
14 using std::string;
15
16 #include <cstdlib>
17 using std::getenv;
18 using std::atoi;
19 using std::exit;
20
21 int main()
22 {
23    char postString[ 1024 ] = "";
24    int contentLength = 0;
25
26    // variables to store user data
27    string dataString = "";
28    string firstname = "";
29    string lastname = "";
30    string address = "";
31    string town = "";
32    string state = "";
33    string zipcode = "";
34    string country = "";
35    string email = "";
```

```
36
37    // data was posted
38    if ( getenv( "CONTENT_LENGTH" ) )
39       contentLength = atoi( getenv( "CONTENT_LENGTH" ) );
40
41    cin.read( postString, contentLength );
42    dataString = postString;
43
44    // search for first '+' character
45    string::size_type charLocation = dataString.find( "+" );
46
47    // search for next '+' character
48    while ( charLocation < string::npos )
49    {
50       dataString.replace( charLocation, 1, " " );
51       charLocation = dataString.find( "+", charLocation + 1 );
52    } // end while
53
54    // find location of firstname
55    int firstStart = dataString.find( "firstname=" ) + 10;
56    int endFirst = dataString.find( "&lastname" );
57    firstname = dataString.substr(firstStart,endFirst - firstStart );
58
59    // find location of lastname
60    int lastStart = dataString.find( "lastname=" ) + 9;
61    int endLast = dataString.find( "&address" );
62    lastname = dataString.substr( lastStart, endLast - lastStart );
63
64    // find location of address
65    int addressStart = dataString.find( "address=" ) + 8;
66    int endAddress = dataString.find( "&town" );
67    address=dataString.substr(addressStart,endAddress-addressStart );
68
69    // find location of town
70    int townStart = dataString.find( "town=" ) + 5;
71    int endTown = dataString.find( "&state" );
72    town = dataString.substr( townStart, endTown - townStart );
73
74    // find location of state
75    int stateStart = dataString.find( "state=" ) + 6;
76    int endState = dataString.find( "&zipcode" );
77    state = dataString.substr( stateStart, endState - stateStart );
78
79    // find location of zip code
80    int zipStart = dataString.find( "zipcode=" ) + 8;
81    int endZip = dataString.find( "&country" );
82    zipcode = dataString.substr( zipStart, endZip - zipStart );
83
84    // find location of country
85    int countryStart = dataString.find( "country=" ) + 8;
86    int endCountry = dataString.find( "&email" );
87    country = dataString.substr( countryStart, endCountry - countryStart );
88
89    // find location of e-mail address
90    int emailStart = dataString.find( "email=" ) + 6;
91    int endEmail = dataString.find( "&submit" );
92    email = dataString.substr( emailStart, endEmail - emailStart );
93
94    cout << "Content-Type: text/html\n\n"; // output header
95
96    // output XML declaration and DOCTYPE
97    cout << "<?xml version = \"1.0\"?>"
```

```
98        << "<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.1//EN\" "
99        << "\"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd\">";
100
101    // output html element and some of its contents
102     cout << "<html xmlns = \"http://www.w3.org/1999/xhtml\">"
103        << "<head><title>Contact Information entered</title></head><body>";
104
105      ofstream outFile( "clients.txt", ios::app ); // output to file
106
107     if ( !outFile ) // file was not opened properly
108     {
109        cerr << "Error: could not open contact file.";
110        exit( 1 );
111     } // end if
112
113     // append data to clients.txt file
114     outFile <<firstname << " " << lastname << "\n" <<address << "\n"
115        << town << " " << state << " " << country << " " << zipcode
116        << "\n" << email << "\n\n";
117
118     // output data to user
119     cout << "<table><tbody><tr><td>First Name:</td><td>" << firstname
120        << "</td></tr><tr><td>Last Name:</td><td>" << lastname
121        << "</td></tr><tr><td>Address:</td><td>" << address
122        << "</td></tr><tr><td>Town:</td><td>" << town
123        << "</td></tr><tr><td>State:</td><td>" << state
124        << "</td></tr><tr><td>Zip Code:</td><td>" << zipcode
125        << "</td></tr><tr><td>Country:</td><td>" << country
126        << "</td></tr><tr><td>Email:</td><td>" << email
127        << "</td></tr></tbody></table></body>\n</html>\n";
128     return 0;
129 } // end main
```

شکل ۱۹-۱۹ | ایجاد فایل طرف سرویس‌دهنده برای ذخیره اطلاعات کاربر.

```
Jane Doe
123 Main Street
Boston MA USA 12345
jan@doe.com
```

شکل ۲۰-۱۹ | محتویات فایل clients.txt.

چند نکته با اهمیت در ارتباط با این برنامه وجود دارد. اول اینکه، هیچ گونه عملیات اعتبارسنجی داده، قبل از نوشتن آنها بر روی دیسک انجام نداده‌ایم. معمولاً، بایستی اسکریپت به بررسی صحت داده‌ها بپردازد. دوم اینکه، فایل ما در شاخه **cgi-bin** قرار دارد که در دسترس عموم است. هر کسی که نام فایل را بداند می‌تواند به آسانی آنرا پیدا کرده و به محتویات دسترسی دسترسی داشته باشد.

این اسکریپت بقدر کافی از کفایت عرضه بر روی اینترنت برخوردار نیست، اما مثالی از نحوه استفاده از فایلهای طرف سرویس‌گیرنده به منظور ذخیره‌سازی اطلاعات است. از آنجا که فایل‌ها بر روی سرویس‌دهنده ذخیره می‌شوند، کاربران نمی‌تواند آنها را تغییر دهند مگر اینکه مجوز انجام اینکار را از مدیر سرویس‌دهنده کسب کرده باشند. بنابر این ذخیره این فایل‌ها بر روی سرویس‌دهنده امن‌تر از

ذخیره‌سازی داده‌های کاربر در کوکی‌ها است. [نکته: برخی از سیستم‌ها اطلاعات کاربر را در پایگاه داده‌های حفاظت شده ذخیره می‌کنند که از سطح امنیتی بالاتری برخوردار است.]

در این بخش با نحوه نوشتن داده در یک فایل طرف سرویس‌گیرنده آشنا شدید. در بخش بعدی شما را با نحوه بازیابی داده‌ها از فایل طرف سرویس‌گیرنده با استفاده از تکنیک‌های معرفی شده در فصل هفدهم آشنا خواهیم کرد.

۱۹-۱۶ مبحث آموزشی کارت خرید

بسیاری از وب سایت‌های تجاری دارای برنامه‌های کاربردی کارت خرید هستند که به مشتریان امکان خرید راحت ایتم‌هایی از وب را فراهم می‌آورند. این سایت‌ها هر آنچه که مشتری می‌خواهد خرید کند ثبت کرده و روش خرید online آسانی در اختیار وی قرار می‌دهند. سپس مشتریان با استفاده از یک کارت خرید الکترونیکی اقدام به خرید می‌کنند، همانطوری که از یک فروشگاه عادی خرید می‌نمایند. همانطوری که کاربران اقدام به افزودن آیتم‌های مورد نظر خود به کارت خرید می‌کنند، سایت محتویات کارت را به روز می‌نماید. پس از تایید کاربر (مشتری)، هزینه آیتم‌ها از کارت خرید دریافت می‌شود. برای آشنایی با تجارت الکترونیکی و کارت خرید در دنیای واقعی، پیشنهاد می‌کنیم تا سری به کتابفروشی Amazan به آدرس www.amazon.com بزنید.

کارت خرید که در این بخش پیاده‌سازی می‌شود (شکل‌های ۲۱-۱۹ الی ۲۴-۱۹) به کاربران امکان خرید کتاب‌های را از یک کتابفروشی فرضی که فقط چهار کتاب می‌فروشد را می‌دهد (به شکل ۲۳-۱۹ نگاه کنید). در این مثال، از چهار اسکریپت، دو فایل طرف سرویس‌دهنده و کوکی‌ها استفاده شده است.

برنامه شکل ۲۱-۱۹ اولین اسکریپت از چهار اسکریپت یاد شده است، که صفحه login (ورود) است. این اسکریپت از تمام اسکریپت‌های این بخش پیچیده‌تر است.

```
// Fig. 19.21: login.cpp
// Program to output an XHTML form, verify the
// username and password entered, and add members.
#include <iostream>
using std::cerr;
using std::cin;
using std::cout;
using std::ios;

#include <fstream>
using std::fstream;

#include <string>
using std::string;

#include <cstdlib>
using std::getenv;
using std::atoi;
using std::exit;

```

```
21  void header();
22  void writeCookie();
23
24  int main()
25  {
26     char query[ 1024 ] = "";
27     string dataString = "";
28
29     // strings to store username and password
30     string userName = "";
31     string passWord = "";
32
33     int contentLength = 0;
34     bool newMember = false;
35
36     // data was posted
37     if ( getenv( "CONTENT_LENGTH" ) )
38     {
39        // retrieve query string
40        contentLength = atoi( getenv( "CONTENT_LENGTH" ) );
41        cin.read( query, contentLength );
42        dataString = query;
43
44        // find username location
45        int userLocation = dataString.find( "user=" ) + 5;
46        int endUser = dataString.find( "&" );
47
48        // find password location
49        int passwordLocation = dataString.find( "password=" ) + 9;
50        int endPassword = dataString.find( "&new" );
51
52        if ( endPassword > 0 ) // new membership requested
53        {
54           newMember = true;
55           passWord = dataString.substr(
56              passwordLocation, endPassword - passwordLocation );
57        } // end if
58        else // existing member
59           passWord = dataString.substr( passwordLocation );
60
61        userName = dataString.substr(
62           userLocation, endUser - userLocation );
63     } // end if
64
65     // no data was retrieved
66     if ( dataString == "" )
67     {
68        header();
69        cout << "<p>Please login.</p>";
70
71        // output login form
72        cout << "<form method = \"post\" action = \"/cgi-bin/login.cgi\">"
73           << "<p>User Name: <input type = \"text\" name = \"user\"/><br/>"
74           << "Password: <input type = \"password\" name = \"password\"/>"
75           << "<br/>New? <input type = \"checkbox\" name = \"new\""
76           << " value = \"1\"/></p>"
77           << "<input type = \"submit\" value = \"login\"/></form>";
78     } // end if
79     else // process entered data
80     {
81        string fileUsername = "";
82        string filePassword = "";
```

```
83        bool userFound = false;
84
85        // open user data file for reading and writing
86        fstream userData( "userdata.txt", ios::in | ios::out);
87
88        if ( !userData ) // could not open file
89        {
90           cerr << "Could not open database.";
91           exit( 1 );
92        } // end if
93
94        // add new member
95        if ( newMember )
96        {
97           // read username and password from file
98           while ( !userFound && userData >> fileUsername >> filePassword )
99           {
100             if ( userName == fileUsername ) // name is already taken
101                userFound = true;
102          } // end while
103
104          if ( userFound ) // user name is taken
105          {
106             header();
107             cout << "<p>This name has already been taken.</p>"
108                << "<a href=\"/cgi-bin/login.cgi\">Try Again</a>";
109          } // end if
110          else // process data
111          {
112             writeCookie(); // write cookie
113             header();
114
115             // write user data to file
116             userData.clear(); //clear eof, allow write at end of file
117             userData << "\n" << userName << "\n" << passWord;
118
119             cout << "<p>Your information has been processed."
120                << "<a href=\"/cgi-bin/shop.cgi\">Start Shopping</a></p>";
121          } // end else
122       } // end if
123       else // search for password if entered
124       {
125          bool authenticated = false;
126
127          // read in user data
128          while ( !userFound && userData >> fileUsername >> filePassword )
129          {
130             // username was found
131             if ( userName == fileUsername )
132             {
133                userFound = true;
134
135                // determine whether password is correct
136                // and assign bool result to authenticated
137                authenticated = ( passWord == filePassword );
138             } // end if
139          } // end while
140
141          // user is authenticated
142          if ( authenticated )
143          {
144             writeCookie();
```

```
145            header();
146
147            cout << "<p>Thank you for returning, " << userName << "!</p>"
148               << "<a href=\"/cgi-bin/shop.cgi\">Start Shopping</a>";
149         } // end if
150         else // user not authenticated
151         {
152            header();
153
154            if ( userFound ) // password is incorrect
155               cout << "<p>You have entered an incorrect password. "
156                  << "Please try again.</p>"
157                  << "<a href=\"/cgi-bin/login.cgi\">Back to login</a>";
158            else // user is not registered
159               cout << "<p>You are not a registered user.</p>"
160                  << "<a href=\"/cgi-bin/login.cgi\">Register</a>";
161         } // end else
162      } // end else
163   } // end else
164
165   cout << "</body>\n</html>\n";
166   return 0;
167 } // end main
168
169 // function to output header
170 void header()
171 {
172   cout << "Content-Type: text/html\n\n"; // output header
173
174   // output XML declaration and DOCTYPE
175   cout << "<?xml version = \"1.0\"?>"
176      << "<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.1//EN\" "
177      << "\"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd\">";
178
179   // output html element and some of its contents
180   cout << "<html xmlns = \"http://www.w3.org/1999/xhtml\">"
181      << "<head><title>Login Page</title></head><body>";
182 } // end function header
183
184 // function to write cookie data
185 void writeCookie()
186 {
187   string expires = "Friday, 14-MAY-10 16:00:00 GMT";
188   cout << "Set-Cookie: CART=; expires=" << expires << "; path=\n";
189 } // end function writeCookie
```

شکل ۲۱-۱۹ | برنامه‌ای که صفحه login را بوجود می‌آورد.

اولین شرط **if** در خط 37 تعیین می‌کند که آیا داده به برنامه ارسال شده است یا خیر. دومین شرط **if** (خط 66) تعیین می‌کند که آیا **dataString** تهی مانده است یا خیر (یعنی داده برای کدگشایی تسلیم نشده یا کدگشایی با موفقیت صورت نگرفته است). اولین بار که این برنامه را اجرا می‌کنیم، شرط اول برقرار نیست و شرط دوم برقرار است، از اینرو خطوط 77-72 فرم XHTML را در اختیار کاربر قرار می‌دهند، همانند اولین تصویر در شکل ۲۱-۱۹. زمانیکه کاربر فرم را پر کرده و بر روی دکمه **login** کلیک کند، فایل **login.cgi** مجدداً تقاضا می‌شود، این بار تقاضا حاوی داده پست شده است، از اینرو شرط موجود در خط 37 با **true** ارزیابی شده و شرط موجود در خط 66 با **false** ارزیابی می‌شود.

اگر کاربر داده را تسلیم کند، کنترل برنامه با بلوک **else** که از خط 79 شروع می‌شود، ادامه می‌یابد، مکانی که اسکریپت داده را پردازش می‌کند. خط 86 فایل **userdata.txt** را باز می‌کند، فایلی که حاوی کلیه اسامی کاربران و کلمات عبور برای اعضای موجود است. اگر کاربر جعبه‌چک **New** را انتخاب کند تا یک عضویت جدید ایجاد شود، شرط موجود در خط 95 با **true** ارزیابی شده و اسکریپت مبادرت به ثبت اطلاعات کاربر در فایل **userdata.txt** در سرویس‌دهنده می‌کند. خطوط 102-98 این فایل را خوانده، نام کاربر را با نام وارد شده مقایسه می‌کنند.

اگر نام کاربر در فایل از قبل وجود داشته باشد، حلقه موجود در خطوط 102-98 قبل از رسیدن به انتهای فایل خاتمه می‌یابد و خطوط 108-107 پیغام مناسبی در اختیار کاربر قرار داده و یک فوق لینک وی را به فرم باز می‌گرداند. اگر نام کاربری وارد شده در فایل **user.data** وجود نداشته باشد، خط 117 اطلاعات کاربر جدید را به فایل با فرمت

```
Bernard
blue
```

اضافه می‌کند. هر نام کاربری و کلمه عبور توسط یک کاراکتر خط جدید از هم متمایز می‌شوند. خطوط 120-119 یک فوق لینک به اسکریپت برنامه شکل ۲۲-۱۹ فراهم می‌آورند که به کاربران امکان خرید را می‌دهد.

آخرین سناریو ممکنه برای این اسکریپت برگشت دادن کاربران است (خطوط 162-123). این بخش از برنامه زمانی اجرا می‌شود که کاربر، نام و کلمه عبور را وارد کند، اما جعبه‌چک **New** را انتخاب نکند. در اینحالت، فرض می‌کنیم که کاربر در حال حاضر دارای یک نام کاربری و کلمه عبور در فایل **userdata.txt** است. خطوط 139-128 کل **userdata.txt** را خوانده و اقدام به یافتن نام کاربر وارد شده می‌کنند. اگر نام کاربر پیدا شود (خط 131)، تعیین می‌کنیم که آیا کلمه عبور وارد شده مطابق با کلمه عبور ذخیره شده در فایل است یا خیر (خط 137). اگر چنین باشد، متغیر بولی **authenticated** با **true** تنظیم می‌شود. در غیر اینصورت با **false** تنظیم می‌گردد. اگر هویت کاربر تایید شود (خط 142)، خط 144 تابع **writeCookie** را برای مقداردهی اولیه یک کوکی بنام **CART** فراخوانی می‌کند (خط 188) که توسط اسکریپت‌های دیگر برای ذخیره‌سازی داده مرتبط با کتاب‌های انتخابی توسط کاربر که به کارت خرید افزوده می‌شوند، بکار گرفته می‌شود. توجه کنید که این کوکی جایگزین هر کوکی موجود همنام شده و داده موجود از جلسه قبل از بین می‌رود. پس از ایجاد کوکی، اسکریپت پیغام خوش آمدگویی به کاربر را به نمایش درآورده و لینکی به **shop.cgi** فراهم می‌آورد که کاربر می‌تواند از آنجا اقدام به خرید کتاب کند (خطوط 148-147).

اگر کاربر تایید نشود، برنامه دلیل آن را مشخص می‌کند (خطوط 160-154). اگر کاربر پیدا شود اما تایید نشود، پیغامی به نمایش در آمده و نشان می‌دهد که کلمه عبور معتبر نبوده است (خطوط 157-155). یک فوق لینک برای صفحه **login** در نظر گرفته شده است که کاربر می‌تواند دوباره از آن طریق اقدام کند.

اگر نام کاربری و هم کلمه عبور هر دو پیدا نشوند، پس یک کاربر ثبت نشده اقدام به ورود کرده است. خطوط 160-159 پیغام مبنی بر اینکه کاربر دارای مجوزهای صحیح برای دسترسی به صفحه نیست به نمایش درآورده و لینکی فراهم می‌آورند که کاربر بتواند دوباره اقدام به ورود کند.

برنامه شکل ۲۲-۱۹ از مقادیر موجود در **catalog.txt** برای چاپ اطلاعات در یک جدول XHTML استفاده کرده (شکل ۲۵-۱۹) است، آیتم‌های که کاربر می‌تواند خرید کند (خطوط 82-45). ستون آخر در هر سطر شامل یک دکمه برای افزودن آن آیتم به کارت خرید است. خطوط 65-63 مقادیر مختلف برای هر کتاب را چاپ کرده و خطوط 76-71 فرمی حاوی دکمه **submit** را برای افزودن هر کتاب به کارت خرید فراهم می‌آورند. فیلدهای پنهان فرم خاص هر کتاب بوده و مرتبط با اطلاعات آن کتاب هستند. توجه کنید که نتیجه مستند XHTML به سرویس‌گیرنده حاوی چند فرم، یکی برای هر کتاب ارسال می‌شود. با این همه، کاربر می‌تواند فقط در هر بار یک فرم را تسلیم (submit) کند. جفت‌های نام–مقدار فیلدهای پنهان در میان فرم تسلیم شده به اسکریپت **viewcart.cgi** ارسال می‌شوند.

```
1  // Fig. 19.22: shop.cpp
2  // Program to display available books.
3  #include <iostream>
4  using std::cerr;
5  using std::cout;
6  using std::ios;
7
8  #include <fstream>
9  using std::ifstream;
10
11 #include <string>
12 using std::string;
13
14 #include <cstdlib>
15 using std::exit;
16
17 void header();
18
19 int main()
20 {
21    // variables to store product information
22    char book[ 50 ] = "";
23    char year[ 50 ] = "";
24    char isbn[ 50 ] = "";
25    char price[ 50 ] = "";
26
27    string bookString = "";
28    string yearString = "";
29    string isbnString = "";
30    string priceString = "";
31
```

```
32    ifstream userData( "catalog.txt", ios::in ); // open file for input
33
34    // file could not be opened
35    if ( !userData )
36    {
37       cerr << "Could not open database.";
38       exit( 1 );
39    } // end if
40
41    header(); // output header
42
43    // output available books
44    cout << "<center><br/>Books available for sale<br/><br/>"
45       << "<table border = \"1\" cellpadding = \"7\" >";
46
47    // file is open
48    while ( userData )
49    {
50       // retrieve data from file
51       userData.getline( book, 50 );
52       bookString = book;
53
54       userData.getline( year, 50 );
55       yearString = year;
56
57       userData.getline( isbn, 50 );
58       isbnString = isbn;
59
60       userData.getline( price, 50 );
61       priceString = price;
62
63       cout << "<tr><td>" << bookString << "</td><td>" << yearString
64          << "</td><td>" << isbnString << "</td><td>" << priceString
65          << "</td>";
66
67       // file is still open after reads
68       if ( userData )
69       {
70          // output form with buy button
71          cout << "<td><form method=\"post\" "
72             << "action=\"/cgi-bin/viewcart.cgi\">"
73             << "<input type=\"hidden\" name=\"add\" value=\"true\"/>"
74             << "<input type=\"hidden\" name=\"isbn\" value=\""
75             << isbnString << "\"/>" << "<input type=\"submit\" "
76             << "value=\"Add to Cart\"/>\n</form></td>\n";
77       } // end if
78
79       cout << "</tr>\n";
80    } // end while
81
82    cout << "</table></center><br/>"
83       << "<a href=\"/cgi-bin/checkout.cgi\">Check Out</a>"
84       << "</body></html>";
85    return 0;
86 } // end main
87
88 // function to output header information
89 void header()
90 {
91    cout << "Content-Type: text/html\n\n"; // output header
92
93    // output XML declaration and DOCTYPE
```

```
94    cout << "<?xml version = \"1.0\"?>"
95       << "<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.1//EN\" "
96       << "\"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd\">";
97
98    // output html element and some of its contents
99    cout << "<html xmlns = \"http://www.w3.org/1999/xhtml\">"
100          << "<head><title>Shop Page</title></head><body>";
101    } // end function header
```

شکل ۲۲-۱۹ | اسکریپت CGI که به کاربران امکان خرید کتاب را می‌دهد.

پس از خرید یک کتاب توسط کاربر، اسکریپت **viewcart.cgi** تقاضا شده و ISBN کتاب خریداری شده به اسکریپت از طریق یک فیلد پنهان در فرم ارسال می‌شود. شکل ۲۳-۱۹ با خواندن مقدار از کوکی ذخیره شده بر روی سیستم کاربر آغاز بکار می‌کند (خط 35). هر داده کوکی موجود در رشته **cookieString** ذخیره شده است (خط 36). عدد ISBN وارد شده از فرم در شکل ۲۲-۱۹ در رشته **isbnEntered** ذخیره می‌شود (خط 52). سپس اسکریپت تعیین می‌کند که آیا کارت در حال حاضر حاوی داده می‌باشد یا خیر (خط 61). اگر نباشد، رشته **cookieString** مقدار وارد شده ISBN را دریافت می‌کند (خط 62). اگر کوکی در حال حاضر حاوی داده باشد، ISBN وارد شده به داده کوکی موجود الصاق می‌شود (خط 64). کتاب جدید در کوکی **CART** در خطوط 68-67 ذخیره می‌شود. خط 84 محتوی کارت را در جدولی با فراخوانی تابع **displayShoppingCart** به نمایش در می آورد.

تابع **displayShoopingCart** ایتم‌های موجود در کارت خرید را در یک جدول به نمایش در می‌آورد. خط 109 فایل طرف سرویس‌دهنده بنام **catalog.txt** را باز می‌کند. اگر فایل با موفقیت باز شود، خطوط 155-122 اطلاعات هر کتاب را از فایل دریافت می‌کنند. خطوط 138-125 این اطلاعات را در شی‌های رشته‌ای ذخیره می‌کنند. خطوط 148-140 تعداد دفعاتی که ISBN جاری در کوکی ظاهر شده است را می‌شمارند (یعنی کارت خرید). اگر کتاب جاری در کارت کاربر دیده شود، خطوط 154-151 یک سطر جدول حاوی عنوان کتاب، کپی رایت، ISBN و قیمت را به همراه تعداد کتاب درخواستی را به نمایش در می‌آورند.

```
1   // Fig. 19.23: viewcart.cpp
2   // Program to view books in the shopping cart.
3   #include <iostream>
4   using std::cerr;
5   using std::cin;
6   using std::cout;
7   using std::ios;
8
9   #include <fstream>
10  using std::ifstream;
11
12  #include <string>
13  using std::string;
14
15  #include <cstdlib>
16  using std::getenv;
```

```
17  using std::atoi;
18  using std::exit;
19
20  void displayShoppingCart( const string & );
21
22  int main()
23  {
24     char query[ 1024 ] = ""; // variable to store query string
25     string cartData; // variable to hold contents of cart
26
27     string dataString = "";
28     string cookieString = "";
29     string isbnEntered = "";
30     int contentLength = 0;
31
32     // retrieve cookie data
33     if ( getenv( "HTTP_COOKIE" ) )
34     {
35        cartData = getenv( "HTTP_COOKIE" );
36        cookieString = cartData;
37     } // end if
38
39     // data was entered
40     if ( getenv( "CONTENT_LENGTH" ) )
41     {
42        contentLength = atoi( getenv( "CONTENT_LENGTH" ) );
43        cin.read( query, contentLength );
44        dataString = query;
45
46        // find location of isbn value
47        int addLocation = dataString.find( "add=" ) + 4;
48        int endAdd = dataString.find( "&isbn" );
49        int isbnLocation = dataString.find( "isbn=" ) + 5;
50
51        // retrieve isbn number to add to cart
52        isbnEntered = dataString.substr( isbnLocation );
53
54        // write cookie
55        string expires = "Friday, 14-MAY-10 16:00:00 GMT";
56        int cartLocation = cookieString.find( "CART=" ) + 5;
57
58        if ( cartLocation > 4 ) // cookie exists
59           cookieString = cookieString.substr( cartLocation );
60
61        if ( cookieString == "" ) // no cookie data exists
62           cookieString = isbnEntered;
63        else // cookie data exists
64           cookieString += "," + isbnEntered;
65
66        // set cookie
67        cout << "Set-Cookie: CART=" << cookieString << "; expires="
68           << expires << "; path=\n";
69     } // end if
70
71     cout << "Content-Type: text/html\n\n"; // output HTTP header
72
73     // output XML declaration and DOCTYPE
74     cout << "<?xml version = \"1.0\"?>"
75        << "<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.1//EN\" "
76        << "\"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd\">";
77
78     // output html element and some of its contents
```

```
79     cout << "<html xmlns = \"http://www.w3.org/1999/xhtml\">"
80        << "<head><title>Shopping Cart</title></head>"
81        << "<body><center><p>Here is your current order:</p>";
82
83     if ( cookieString != "" ) // cookie data exists
84        displayShoppingCart( cookieString );
85     else
86        cout << "The shopping cart is empty.";
87
88     // output links back to book list and to check out
89     cout << "</center><br/>";
90     cout << "<a href=\"/cgi-bin/shop.cgi\">Back to book list</a><br/>";
91     cout << "<a href=\"/cgi-bin/checkout.cgi\">Check Out</a>";
92     cout << "</body></html>\n";
93     return 0;
94  } // end main
95
96  // function to display items in shopping cart
97  void displayShoppingCart( const string &cookieRef )
98  {
99     char book[ 50 ] = "";
100    char year[ 50 ] = "";
101    char isbn[ 50 ] = "";
102    char price[ 50 ] = "";
103
104    string bookString = "";
105    string yearString = "";
106    string isbnString = "";
107    string priceString = "";
108
109    ifstream userData( "catalog.txt", ios::in ); // open file for input
110
111    if ( !userData ) // file could not be opened
112    {
113       cerr << "Could not open database.";
114       exit( 1 );
115    } // end if
116
117    cout << "<table border = 1 cellpadding = 7 >";
118    cout << "<tr><td>Title</td><td>Copyright</td><td>ISBN</td>"
119       << "<td>Price</td><td>Count</td></tr>";
120
121    // file is open
122    while ( !userData.eof() )
123    {
124       // retrieve book information
125       userData.getline( book, 50 );
126       bookString = book;
127
128       // retrieve year information
129       userData.getline( year, 50 );
130       yearString = year;
131
132       // retrieve isbn number
133       userData.getline( isbn, 50 );
134       isbnString = isbn;
135
136       // retrieve price
137       userData.getline( price, 50 );
138       priceString = price;
139
140       int match = cookieRef.find( isbnString, 0 );
```

```
141       int count = 0;
142
143       // match has been made
144       while ( match >= 0 && isbnString != "" )
145       {
146          count++;
147          match = cookieRef.find( isbnString, match + 13 );
148       } // end while
149
150       // output table row with book information
151       if ( count != 0 )
152          cout << "<tr><td>" << bookString << "</td><td>" << yearString
153             << "</td><td>" << isbnString << "</td><td>" << priceString
154             << "</td><td>" << count << "</td></tr>";
155    } // end while
156
157    cout << "</table>"; // end table
158 } // end function displayShoppingCart
```

شکل ۲۳-۱۹ | اسکریپت CGI که به کاربران امکان مشاهده محتویات کارت خرید را می‌دهد.

برنامه شکل ۲۴-۱۹ صفحه‌ای است که به هنگام انتخاب لینک **check out** (یعنی خرید کتاب‌های موجود در کتاب خرید) به نمایش در می‌آید. این اسکریپت پیغامی به کاربر نشان داده و تابع writeCookie را فراخوانی می‌کند (خط 13) که اطلاعات جاری در کارت خرید را پاک می‌کند. شکل ۲۵-۱۹ نشان دهنده محتویات فایل **catalog.txt** است. این فایل بایستی در همان شاخه‌ای باشد که اسکریپت‌های CGI قرار دارند.

```
1   // Fig. 19.24: checkout.cpp
2   // Program to log out of the system.
3   #include <iostream>
4   using std::cout;
5
6   #include <string>
7   using std::string;
8
9   void writeCookie();
10
11  int main()
12  {
13     writeCookie(); // write the cookie
14     cout << "Content-Type: text/html\n\n"; // output header
15
16     // output XML declaration and DOCTYPE
17     cout << "<?xml version = \"1.0\"?>"
18        << "<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.1//EN\" "
19        << "\"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd\">";
20
21     // output html element and its contents
22     cout << "<html xmlns = \"http://www.w3.org/1999/xhtml\">"
23        << "<head><title>Checked Out</title></head><body><center>"
24        << "<p>You have checked out<br />"
25        << "You will be billed accordingly<br />To login again, "
26        << "<a href=\"/cgi-bin/login.cgi\">click here</a>"
27        << "</center></body></html>\n";
28     return 0;
29  } // end main
30
```

```
// function to write cookie
void writeCookie()
{
   // string containing expiration date
   string expires = "Friday, 14-MAY-10 16:00:00 GMT";

   // set cookie
   cout << "Set-Cookie: CART=; expires=" << expires << "; path=\n";
} // end writeCookie
```

شکل ۲٤-۱۹ برنامه تصفیه حساب.

```
visual Basic .NET How to Program
2002
0-13-029363-6
$50.00
C# How to Program
2002
0-13-062221-4
$49.95
C How to Program 4e
2004
0-13-142644-3
$88.00
Java How to Program 6e
2005
0-13-148398-6
$88.00
```

شکل ۲۵-۱۹ محتویات فایل catalog.txt

منابع اینترنت و وب

Apache

httpd.apache.org
www.apacheweek.com
linuxtoday.com/stories/18780.html

CGI

www.gnu.org/software/cgicc/cgicc.html
www.hotscripts.com
www.jmarshall.com/easy/cgi
www.w3.org/CGI
www.w3.org/Protocols