Lua – następna generacja

Blask księżyca

Lua to unikalny język skryptowy, który posiada znaczną liczbę wielbicieli wywodzących się z dwóch nisz rynkowych: użytkowników Linuksa oraz programistów gier. W czym tkwi urok tego języka? Co w nim takiego niezwykłego? Steven Goodwin prowadzi dochodzenie w tej sprawie.

STEVEN GOODWIN

www.photocase.de

Z lingwistycznego punktu widzenia, Lua to zwykły język programowania. Posiada wszystkie cechy, które czynią go językiem użytecznym (zmienne, funkcje, elementy sterujące). Brakuje mu jednak możliwości, jakie posiadają inne języki, np. obsługi wyrażeń regularnych (jak Perl) i szybkości (jak język C). Jego siłą nie są indywidualne cechy, lecz funkcjonalność całości oraz sposób, w jaki łączy się on z zewnętrznym światem.

W niniejszym artykule przyjrzyjmy się Lua jako narzędziu do adaptacji i konfiguracji. Pokażemy sposób na dostosowywanie określonego oprogramowania z wykorzystaniem Lua. Jako przykładu użyjemy terminalowej przeglądarki WWW elinks. Opiszemy kroki, jakie musi wykonać użytkownik, aby zaimplementować taką funkcjonalność w swoich własnych aplikacjach. Ale rozpoczniemy od zwyczajowej lekcji historii...

Wschód Księżyca

Początki Lua miały miejsce w 1993 roku. Jest to język stworzony przez brazylijskich naukowców z grupy Tecgraf, będącej laboratorium wydziału informatyki Papieskiego Katolickiego Uniwersytetu Rio de Janeiro (Pontifical Catholic University of Rio de Janeiro – PUC-Rio). Jego zadaniem było dostarczenie jasnych metod rozszerzenia aplikacji poprzez prosty proceduralny język oraz tradycyjne typy danych. Wszystkie te cechy pozostały podstawową częścią Lua do dnia dzisiejszego, czyli do wersji 5.0.2, zawierającej drobne poprawki. Rozwój języka jest dość powolny – w całej 11-letniej historii ukazało się jedynie 12 publicznych wydań. Rekompensatą za tą powolność jest niewielka szansa na znalezienie dziury w tak dopracowanej aplikacji. Każde kolejne wydanie jest niewiarygodnie stabilne.

Podstawowa struktura Lua (nazwa ta w języku portugalskim oznacza księżyc) składa się z interpretera języka (program lua), który generuje i wykonuje kod binarny oraz zestawu opcjonalnych bibliotek (realizujących obsługę wejścia, wyjścia, funkcje matematyczne itd.) napisanych w standardowym C. Do tego jest jeszcze kompilator (luac) do generowania kodu pośredniego. Ponieważ kod Lua jest wysoce standaryzowany, działa na prawie każdej platformie wyposażonej w kompilator C. Ze względu na modularną budowę może się on „zmieścić” w urządzeniach przenośnych. Wersja linuksowa ma objętość nieco ponad 100 kB, a domyślny zestaw bibliotek zajmuje dalsze 72 kB. Wiele projektów typu embedded korzysta z tak niewielkich rozmiarów i przenośności języka. Używa go kilka dużych firm tworzących gry: Criterion Studios, LucasArts (Grim Fandango) i BioWare (MDK2 i Baldur's Gate). Jeżeli masz ochotę przejrzeć listę projektów, w których wykorzystano Lua, polecam odwiedzić stronę [1].

Harvest Moon

Lua posiada wszystkie cechy, jakich można się spodziewać od funkcjonalnego języka skryptowego (włączając w to także dostępny na WWW podręcznik [2]). Przede wszystkim mamy do dyspozycji typy danych, nie jest ich zbyt wiele (Tabela 1), ale wystarczająco dużo do zaspokojenia większości zwykłych potrzeb programisty. Jeśli chodzi o zmienne, to nie muszą być one deklarowane. Typ zmiennej jest określany na podstawie kontekstu, w jakim jest używana. Próby wykorzystania niezdefiniowanych zmiennych powodują, że przyjmują one typ nil, czego skutkiem może być niepowodzenie niektórych operacji (np. łączenia łańcuchów). W języku Lua pojęcie liczby (number) jest ekwiwalentne do typu double w języku C. Wykorzystując Lua w wersji 5, można zmienić go na float lub nawet int i zrekompilować Lua, dodając tylko:

Tabela 1: Typy danych w Lua

Typ	Nazwa identyfikatora	Nazwa dla lua_istyp
Nil	LUA_TNIL	nil
Number	LUA_TNUMBER	number
Bool	LUA_TBOOLEAN	boolean
String	LUA_TSTRING	string
Table	LUA_TTABLE	table
Function	LUA_TFUNCTION	cfunction
Userdata	LUA_TUSERDATA	userdata

Uwagi

Identyfikator może być używany do konwersji stałej w łańcuch wewnątrz C, wykorzystując lua_typename(lua_State *L, int type), lua_isnumber akceptuje zarówno liczby (123) jak i liczbowe łańcuchy ("123"), lua_toboolean zwraca 0 dla false, nil oraz 1 dla pozostałych

```
#define LUA_NUMBER float
```

przed włączeniem #include lua.h – każdy korzystający z wersji 4 lub wcześniejszej będzie musiał nieco zmodyfikować kod.

W Lua istnieje oczywiście tradycyjna grupa instrukcji do sterowania przepływem. Obejmuje ona następujące wzorce:

```
do block end
while exp do block end
repeat block until exp
if exp then block end
if exp then block else ➥
otherblock end
if exp then block elseif exp ➥
then otherblock end [i tak ➥
dalej...]
for var=start,end[,krok] do ➥
block end
for var in iterator do block end
```

Jak widać, nie ma tu niczego niezwykłego. Programiści C powinni być jednak świadomi, że wartość 'end' jest także sprawdzana w pętli for. Zauważmy, że wszystkie wyrażenia wykorzystują słowo end dla zaznaczenia zakończenia bloku, zamiast bardziej tradycyjnych nawiasów klamrowych. To uproszczenie jest jawną próbą nakłonienia nie-programistów do wejścia w świat skryptów. Pierwszeństwo słów przed symbolami jest oczywiste, gdy zrozumiesz, że operatory: !, && i || zostały zastąpione ich alfabetycznymi odpowiednikami: not, and, oraz or.

Składnia Lua eliminuje niektóre ograniczenia obecne w innych językach. Na przykład funkcja może zwrócić bez żadnego problemu dwa (lub więcej) parametrów.

```
function onetwo()
return 1,2
end
one,two = onetwo()
```

Lua posiada funkcje lokalne i anonimowe. Funkcja lokalna to taka, którą można tylko wywołać z wnętrza funkcji, w której jest zadeklarowana. Jest to dosyć niezwykłe dla programistów C, natomiast oczywiste dla wielbicieli innych języków. Anonimowe funkcje możemy zawrzeć jako całą funkcję w miejscu, gdzie zwykle moglibyśmy umieścić wywołanie zwrotne. Zapobiega to potrzebie umieszczania dodatkowych funkcji wewnątrz kodu.

Aby uzyskać bardziej szczegółowe informacje dotyczące składni, warto przeczytać oryginalną wersję dokumentacji, dostępną w witrynie Lua [2]. Bardziej bezpośrednią pomoc można uzyskać dzięki liście dyskusyjnej [3], zajmującej się bardziej zaawansowanymi tematami.

Właściwości Lua jako języka programowania nie są tym, co decyduje o jego popularności wśród programistów. Natomiast możliwości zastosowania go jako narzędzia konfiguracyjnego powodują, że jest on uważany za przebój rynkowy. Dodanie skryptów Lua do własnego programu jest dla programisty proste. Lua być to wykorzystywany do tworzenia wtyczek do aplikacji (plug-in) lub do obsługi plików konfiguracyjnych. Nie musi to być statyczna konfiguracja, jak w większości innych aplikacji. Wykorzystując język programowania do wygenerowania konfiguracji, możesz stworzyć coś znacznie bardziej elastycznego i dynamicznego.

Dynamiczna konfiguracja jest czymś rzadkim. Tylko bardziej złożone aplikacje, typu Apache, posiadają takie możliwości. Ale nawet w takiej sytuacji dyrektywy warunkowe, takie jak IfModule, są rzadko używane i mają ograniczony zakres. Prawdziwie dynamiczna konfiguracja może ułatwić proces instalacyjny poprzez automatyzację – samodzielną adaptację do struktury katalogów, liczby użytkowników, wykorzystywanego pasma, obciążenia mikroprocesora itd.

Lua umożliwia także łatwą metodę dodania punktów zaczepienia (hooks) do oprogramowania, w celu dostosowania go do własnych potrzeb. Przyjrzymy się tej właściwości, dodając kilka punktów zaczepienia do przeglądarki tekstowej elinks.

Listing 1: hello.lua

```
-- Komentarz. Złośliwa uwaga➥
 komentująca brak oryginalności➥
 programisty!
function hello()
write("Witaj Lua!")
end
hello()
```

Zaczepić się...

Przechwytywanie to metoda, za pomocą której program (w naszym przypadku elinks) wywołuje specjalną funkcję za każdym razem, kiedy musi zrobić coś ważnego. „Coś ważnego" może oznaczać np. przejście do URL lub pobranie strony HTML z serwera. Normalnie funkcja ta nic nie zrobi i zwróci kontrolę do głównego programu oraz umożliwi przejście do URL.

Kiedy przechwytywanie jest umieszczone w specjalnej funkcji, sterowanie jest przekazywane z głównego programu do funkcji przechwytujących. Od tego momentu funkcja przechwytująca przejmuje kontrolę nad danymi i może np. podmienić URL. Ponieważ przechwytywanie jest oprogramowywane z programu napisanego w Lua – możemy go nadpisywać zgodnie z naszymi osobistymi preferencjami.

Chcę na przykład odwiedzić jedną z moich własnych witryn internetowych *www.BlueDust.com*, wpisując bd. Mogę to zrobić tworząc obsługę przechwytywania w procedurze „skocz do URL":

```
if (url == „bd") then
return „www.bluedust.com"
end
```

Błyskawiczna konfiguracja!

Specjalna wersja elinks posiada znaczną liczbę różnych przechwytywań (Tabela 2 –

Tabela 2: Haczyki i zakamarki

Funkcja przechwytywania	Wywoływana gdy...	Powinna zwrócić...	Uwagi
goto_url_hook(url, current_url)	URL jest wprowadzany w oknie dialogowym Go to URL	Nowy URL lub nil, aby anulować	
follow_url_hook(url)	URL został wybrany	Nowy URL lub nil, aby anulować	
pre_format_html_hook(url, html)	Dokument został pobrany	Zmodyfikowany łańcuch lub nil, jeśli nie dokonano żadnych zmian	Może usunąć reklamy/śmieci ze źle zaprojektowanych stron internetowych
lua_console_hook(string)	coś jest wprowadzone w konsoli Lua (wpisz, w elinks)	Komenda (run, eval, goto-url, or nil), po której następuje odpowiedni argument (program do uruchomienia, kod Lua do ewaluacji, URL do którego należy przejść lub odpowiednio nil)	
quit_hook()	elinks tuż przed wyjściem	Nic	Zwalnianie zasobów

Haczyki i zakamarki). Możemy wykorzystać niektóre z nich do własnych celów.

Program elinks wywołuje nasz skrypt. Możliwe jest jednak, aby i nasz skrypt wywołał elinks poprzez funkcje zwrotne, co zapewnia pobieranie informacji takich jak strona tytułowa. Jakie parametry powinny być przekazywane wraz ze skrótami klawiszowymi w celu ich wykorzystania? URL, lista zakładek? W tym miejscu do gry wchodzą funkcje wywoływane zwrotnie (callback function).

Funkcje wywoływane zwrotnie są specjalnymi funkcjami wybranymi przez elinks, z których możemy (jako skrypt) skorzystać. Pozwala to na wywołanie ich, jakby były częścią skryptu Lua, włączając w to funkcje current_url i current_title. Dzięki temu będziemy mogli utworzyć proste przechwytywanie w elinks, czego efektem będzie skrócony URL bieżącej strony internetowej.

Do pracy

Naszym pierwszym zadaniem (by zaoszczędzić sobie późniejszego bólu głowy) jest upewnienie się, że elinks został rzeczywiście skompilowany z obsługą skryptów.

Rysunek1: Rezultat pracy funkcji przechwytującej.

Można to sprawdzić przez otwarcie okna dialogowego About, korzystając z kombinacji klawiszy Alt+H (aby otworzyć menu Help), a następnie naciskając A. Pomiędzy wieloma funkcjami powinny znajdować się słowa: *Scripting (Lua)*.

Taka wersja elinks jest domyślnie zawarta w większości dystrybucji, można też ją pobrać z adresu [4]. Gdy mimo wszystko posiadasz elinks bez obsługi Lua, można to rozwiązać kompilując program ponownie przy pomocy opcji:

```
$./configure --with-lua
$ make
# make install
```

Utworzy to przykładowy plik hooks.lua w katalogu elinks/contrib/lua. Warto go przeczytać w celu zgłębienia innych możliwości wykorzystania skryptów Lua.

Następnie musimy utworzyć skrypt uruchamiany podczas startu elinks. Jest on umieszczony w ~/.elinks/hooks.lua i uruchamia się przy starcie przeglądarki. Jeśli kod został zawarty wewnątrz function, nic nie pojawi się na ekranie.

Pierwsza z funkcji dołączy kod goto_url_hook. Jak uprzednio wspomniano, jest on wywoływany po wciśnięciu klawisza g, służącego do zmiany strony internetowej – patrz Listing 2.

To naprawdę łatwe – odwiedź swoją najmniej lubianą stronę internetową. Następnie naciśnij „g", po którym następuje słowo kluczowe tiny i wciśnij Enter. Jeśli podobnie jak ja, wybierzesz Google jako swoją testową witrynę internetową, zostaniesz przeniesiony na stronę *tinyurl.com* – Rysunek 1.

Jak widać, nowy url *http://tinyurl.com/161* może być kopiowany, wklejany, przesyłany e-mailem etc. Jeśli wiesz, jak dodać skróty do elinks, możesz zaoszczędzić sobie kilku naciśnięć klawiszy. Czytelnicy, którzy uprzednio wczytali się w Tabelę 3, wiedzą o istnieniu funkcji bind_key. Popatrzmy na przykład jej użycia w Listingu 3.

Przykład demonstruje użyteczność ano-

Listing 2: goto_url_hook

```
function goto_url_hook (url, ➥
current_url)
if url == "tiny" then
  return "http://tinyurl.com/➥
create.php?url="..current_url
end
return url
end
```

Tabela 3: Wszystko i nic!

Nazwa funkcji	Cel
enable_systems_functions()	Umożliwia użycie pewnych funkcji (np. openfile). Patrz – Ramka 1.
current_url()	Pobiera URL bieżącej strony elinks
current_link()	Pobiera obecnie wybrany odnośnik (lub nil, jeśli żaden nie jest wybrany)
current_title()	Pobiera tytuł strony
current_document()	Pobiera stronę HTML jako łańcuch
current_document_formatted([width])	Pobiera stronę HTML sformatowaną do opcjonalnej szerokości
pipe_read(command)	Uruchamia daną komendę i odczytuje dane
execute(command)	Uruchamia daną komendę w zadanej powłoce (wykorzystując sh -c)
bind_key (keymap, keystroke, function)	Uruchamia funkcję po naciśnięciu klawisza keystroke. Powinien zwrócić komendę i parametr, taki jak lua_console_hook

Listing 3: Funkcja bind_key

```
bind_key ("main", "Ctrl-T",
  function ()
    return "goto_url", "http://
tinyurl.com/create.php?url=
"..current_url()
  end
  )
```

nimowych funkcji i łatwość, z jaką dwie wartości mogą być zwrócone z funkcji. W przypadku tego polecenia jest to instrukcja goto_url i parametr URL.

Aby nadać ostateczny szlif naszemu programowi, usuniemy zdublowaną informację dotyczącą URL, pisząc plik hooks.lua – patrz Listing 4.

Jak widać, Lua oferuje dużą łatwość dodawania funkcjonalności do oprogramowania. Nie potrzebujesz niczego więcej, oprócz odrobiny wyobraźni i zaprezentowanych tutaj metod. Zazwyczaj większa elastyczność dla końcowego użytkownika oznacza więcej komplikacji dla programisty. W przypadku Lua, tak jednak nie jest! Spójrzmy dlaczego...

Przenośność i współpraca

Jak w każdym elastycznym systemie, w Lua istnieją trzy podstawowe fazy: inicjalizacja (i deinicjalizacja), komunikacja przychodząca (gdzie skrypt „rozmawia” z programem w C) i komunikacja wychodząca (gdy C „rozmawia” ze skryptem). Wszystkie trzy obszary są bardzo proste i do ich obsługi można użyć zaprezentowanych poniżej podstawowych szablonów.

Zacznijmy od początku:

```
#include <lua.h>
int main(int argc, char *argv[])
{
lua_State *pLua;
 pLua = lua_open();
 printf('Witaj Świecie!');
 lua_close(pLua);
return 0;
}
```

Używam Lua w wersji 5 – jest to moja osobista preferencja. Może ona prowadzić do drobnych problemów związanych z kompatybilnością. W wersji 4 lua_open pobiera jako parametr rozmiar stosu. Użytkownicy wersji 4 powinni zauważyć, że plik nagłówkowy musi zostać zmieniony na lua40/lua.h. Są to dwie przykładowe zmiany związane z kompatybilnością między wersjami. Kompilując program w C, musimy konsolidować go z biblioteką Lua:

```
$ gcc -llua mycode.c
```

Jeśli chcesz wykorzystać Lua do stworzenia samodzielnego programu, będzie potrzebna obsługa formy wejścia/wyjścia. Lua nie zawiera takiej obsługi, ponieważ większość aplikacji dostarcza swoich własnych I/O, uwalniając w ten sposób kod Lua od niepotrzebnego balastu. Zawsze można jednak skorzystać ze standardowej biblioteki języka C, dołączając następujący kod Lua:

```
lua_baselibopen(pLua);
lua_iolibopen(pLua);
lua_strlibopen(pLua);
lua_mathlibopen(pLua);
```

Podobnie można postępować z innymi bibliotekami, stosownie do potrzeb. Włączenie jakiejkolwiek z nich wymaga zawsze konsolidacji z biblioteką lualib. Przykładowo wykorzystanie biblioteki matematycznej C (Lua jej nie zawiera) wymaga następujących opcji kompilacji:

```
$ gcc -llua -llualib
-lm mycode.c
```

pLua przechowuje stan całego systemu Lua. Ponieważ Lua jest współużywalna, możemy wywołać lua_open tyle razy, ile chcemy. Żaden stan nie będzie w konflikcie z drugim, dając możliwość użycia Lua jako części wielowątkowego systemu. Oczywiście każda interakcja z Lua musi wówczas wykorzystywać wskaźnik.

Mając wskaźnik do Lua, możemy dostarczyć do niego kod, który przetworzy wbudowany interpreter języka:

```
lua_dostring(pLua,
'number=12345');
// inny kod
lua_dostring(pLua,
'print(number)');
```

Tak długo, jak przechowujemy pLua, wszystkie zmienne pozostaną w stanie Lua. Za każdym razem, gdy wywołujemy funkcję np. taką jak print, Lua oceni, czy funkcje zostały zadeklarowane i zwróci wszystkie rezultaty przechowywane w stanie pLua.

Teraz już widać, jak łatwo można zbudować taką prostą funkcją swój własny interpreter i debbuger. Przykładem jest funkcja dofile, która uruchamia kod zawarty w pliku, tak jakby był uruchamiany z linii poleceń. W przeciwieństwie do linii poleceń, gdy raz uruchomimy plik, stan pozostaje w pLua. Zmienne te razem z zadeklarowanymi funkcjami mogą być dostępne poprzez kod C lub plik Lua załadowany przez lua_dofile albo kod ewaluowany przez lua_dostring.

```
int result;
result = lua_dofile
(pLua, 'config.lua');
```

W tym przypadku result zwraca wynik uruchomienia pliku zewnętrznego.

Coś pożytecznego

Napiszmy teraz coś użytecznego w skrypcie config.lua, wywołującym funkcję w programie napisanym w C. Będzie to działać analogicznie jak funkcja current_url w elinks.

Musimy najpierw zarejestrować jedną z funkcji C w Lua – połączy to dwa światy różnych języków. Nadamy tej funkcji na-

Listing 4: Plik hooks.lua

```
function get_tiny(url)
  return "http://tinyurl.com/create.php?url="..url
end
function goto_url_hook (url, current_url)
if url == "tiny" then
  return get_tiny(current_url)
end
return url
end
bind_key ("main", "Ctrl-T", function () return "goto_url",
get_tiny( current_url() ) end )
```

zwę, która będzie wykorzystywana przez Lua. Po niej nastąpi nazwa naszej rzeczywistej funkcji w C:

```
lua_register(pLua, „factorial",
 c_factorial);
```

Od tego momentu możemy pozwolić Lua na przejęcie kontroli. Zorganizuje on całe wywoływanie funkcji, przekazywanie parametrów i zwracanie rezultatów. Wszystko co musimy zrobić, to zdefiniować pobieranie parametrów we właściwym porządku oraz zwracanie poprawnych rezultatów.

Ponieważ Lua wspiera zwracanie na raz wielu parametrów (argumentów narzuconego typu), do obsłużenia tego nie możemy wykorzystać żadnego prototypu języka C. Zamiast tego, parametry są odkładane (i pobierane) ze stosu. Możemy zatem sprawdzać stos, aby dowiedzieć się, ile na nim znajduje się elementów i jakiego są typu. Stos, podobnie jak zmienne, przechowuje wszystkie wspierane typy danych. I dlatego od nas (jako programistów C) zależy prawidłowe żądanie pobrania właściwego typu danych podczas zdejmowania ze stosu. Jest to dodatkowa zaleta, dzięki której jesteśmy pewni, że każda funkcja w C ma ten sam prototyp, gdy jest wykorzystywany poprzez Lua.

Procedura obliczająca silnię factorial może pobrać pewną narzuconą ilość liczb całkowitych i zwrócić silnię każdej z nich. Otrzymamy liczbę argumentów i obliczony rezultat dla każdej z nich, pod warunkiem, że jest to liczba. Kod będzie wyglądał jak ten na Listingu 5.

Zauważmy, że indeksy stosu są zliczane od 1 (a nie od 0) i że należy zwrócić liczbę parametrów odłożonych na powrotny stos. Poza tym pisanie swoich własnych funkcji jest proste. Typy danych mogą się różnić (patrz Tabela 1), ale zasady pozostają dokładnie takie same.

Listing 5: Procedura obliczająca silnię factorial

```
int compute(int n) { return n<2?1:n*compute
(n-1); }
int c_factorial(lua_State *pLua)
{
int params = lua_gettop(pLua);
int n, result;
int answers = 0;
  for(n=1; n<=params; n++) {
    if (lua_isnumber(pLua, n)) {
      result = compute( lua_tonumber(pLua, n) );
      lua_pushnumber(pLua, result);
      answers++;
    } else {
      // Error!
      break;
    }
  }
  return answers;
}
```

Zdobywcy Księżyca

Wywoływanie funkcji Lua z poziomu języka C jest łatwe. Działa to na tej samej zasadzie, co odkładanie danych na stos. W tym przypadku musimy najpierw umieścić nazwę funkcji, po której nastąpi w określonym porządku lista argumentów. Ponieważ typy danych różnią się pomiędzy C i Lua, do odłożenia odpowiedniego typu na stos należy użyć właściwej funkcji (Listing 6).

Będzie to miało taki sam skutek jak wywoływanie przez Lua jego własnych funkcji, dlatego:

```
result1, result2 =
swap_greeting('Steev', 'Witaj');
```

Zauważmy, że porządek parametrów został odwrócony po to, by poradzić sobie z ich zdjęciem ze stosu w następującym porządku: pierwszy na wejściu – ostatni na wyjściu.

Aby mieć wynik tej funkcji w kodzie C, będziemy musieli samemu odczytać dane ze stosu, a następnie jawnie je usunąć, jak to prezentuję poniżej:

```
pResult2 = lua_tostring
(pLua, -1);
pResult1 = lua_tostring
(pLua, -2);
lua_pop(L, 2);
```

Zauważ w tym miejscu odwrócony porządek i wykorzystanie ujemnych indeksów stosu.

Listing 6: Wywoływanie Lua

```
// Nazwa funkcji jest globalnym symbolem: musimy go wykorzystać
w zamian za łańcuch zawierający nazwę funkcji
lua_getglobal(pLua, „swap_greeting");
// Nasz pierwszy parametr
lua_pushstring(pLua, „Steev");
// Nasz drugi parametr
lua_pushstring(pLua, „Witaj);
// Wywołanie siebie samej
lua_call(pLua, 2/*liczba argumentów wejściowych*/, 2/*liczba argumentów
wyjściowych*/);
// Pobranie rezultatów do zmiennych Lua
lua_setglobal(pLua, 'result2');
lua_setglobal(pLua, 'result1');
```

Podsumowanie

Na kilku przykładach nauczyliśmy się podstaw nowego języka. Rozszerzyliśmy możliwości przeglądarki internetowej i odkryliśmy sposób na uzupełnienie naszego własnego projektu dynamicznymi skryptami konfiguracyjnymi. Wszystko to jest możliwe dzięki elastyczności Lua. Jestem pewien, że będziecie mieli pomysły na własne projekty: można np. dodać przesyłanie e-mailem stron (lub odnośników) bezpośrednio z programu Mutt, bądź usuwać reklamy z niektórych stron internetowych i wiele, wiele innych... ■

INFO

[1] Zastosowania Lua: *http://www.lua.org/uses.html*

[2] Dokumentacja Lua: *http://www.lua.org/manual/5.0/*

[3] Forum Lua: *http://archive.neotonic.com/archive/lua-l*

[4] Program elinks: *http://elinks.or.cz/download.html*