helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

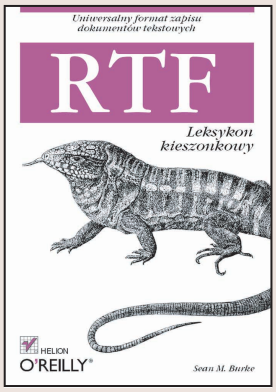
KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

RTF. Leksykon kieszonkowy



Autor: Sean Burke
Tłumaczenie: Marcin Jędrysiak
ISBN: 83-7361-191-6
Tytuł oryginału: RTF Pocket Guide
Format: B5, stron: 174

RTF (Rich Text Format) jest formatem dokumentów. Nie jest to jednak język znaczników, który mógłby być użyty do ręcznego zakodowania całego dokumentu (choć jest to możliwe). Według założeń jego twórców, RTF jest formatem danych, które mogą być odczytywane i zapisywane przez aplikacje dowolnego rodzaju. Założenia te zostały spełnione: obecnie setki aplikacji korzysta właśnie z tego formatu. Elastyczność RTF sprawia, że jest to idealny format do wielu zastosowań i może być wykorzystany zarówno do generowania faktur i raportów, jak i do tworzenia słowników na podstawie baz danych słów.

Niniejsza książka stanowi wygodne źródło wiedzy na temat formatu RTF i omawia sposoby użycia tego standardu, włącznie z informacjami niezbędnymi do napisania programu generującego pliki RTF.

W leksykonie opisano:

- Podstawową składnię RTF
- Zapis akapitów w RFT
- Formatowanie znaków
- Strukturę dokumentu RTF
- Funkcje dodatkowe
- Style w kodzie RTF
- Tworzenie tabel
- Tworzenie plików pomocy MS Windows

Cennym uzupełnieniem informacji o RTF jest rozdział poświęcony pisaniu programów, generujących pliki RTF, wraz z przykładowymi kodami źródłowymi. Ta niewielka książeczka będzie z pewnością przydatna wszystkim, którzy chcą skorzystać z tego uniwersalnego i wygodnego formatu zapisu dokumentów tekstowych.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Rozdział 1. Przedstawienie formatu RTF 5

Dlaczego format RTF? 6
Podział książki 7
„Witaj świecie" w dokumencie RTF 9
Przedstawienie prostego dokumentu RTF 11
Podstawowa składnia formatu RTF 15
Akapity 21
Formatowanie znaków 35
Struktura dokumentu 47
Dodatkowe funkcje 62
Style 84
Tabele 92

Rozdział 2. Tworzenie plików pomocy MS Windows 111

Plik pomocy RTF — podstawy 111
Hiperłącza i wyskakujące łącza 114
Dodatkowe informacje o tematach 117
Rysunki 121
Problemy z plikami pomocy RTF 125
Dodatkowe informacje 126

Rozdział 3. Przykładowe programy 127

Generator kalendarza 127
Program wyświetlający zawartość katalogu 134
Program do projektowania kopert na płyty (origami) 140
Narzędzie do pobierania metadanych RTF 152
Uwagi na temat analizy plików RTF 155

Rozdział 4. Użyteczne informacje 157

Tabela znaków ASCII-RTF 157
Sekwencje ucieczki dla polskich znaków 165
Kody języków RTF 166
Konwersja odległości na twipy 168

Skorowidz 171

Rozdział 3. Przykładowe programy

Standard RTF, podobnie jak każdy inny format dokumentów, nabiera znaczenia dopiero w kontekście jego wykorzystania, czyli sposobu formatowania, odczytywania i zapisywania dokumentów. Kwestie analizowania i przetwarzania kodu RTF wykraczają jednak poza tematykę niniejszej książki, aczkolwiek przedstawione w tym rozdziale przykładowe programy przedstawiają typowe problemy i trudności, na jakie można natknąć się w czasie generowania dokumentów RTF z poziomu aplikacji. Wszystkie omawiane narzędzia i programy pokazują sposób wykorzystania różnych funkcji, jakie zapewnia standard RTF.

Wszystkie opisane w tym rozdziale programy zostały napisane w Perlu. Aby ułatwić poznanie ich działania, w kodzie źródłowym umieszczono obszerne komentarze. Jeżeli Czytelnik planuje generowanie własnych dokumentów RTF z poziomu aplikacji Perl, warto skorzystać z kilku przydatnych modułów, jakie są dostępne w bibliotece CPAN (*http://search.cpan.org*). Godne polecenia są przede wszystkim moduły *RTF::Writer* i *RTF::Generator*. Żaden z tych modułów nie zostanie jednak użyty w poniższych przykładach, ponieważ przedstawiają one sposoby generowania dokumentów RTF bez pośrednictwa dodatkowych interfejsów API lub modułów Perl.

Generator kalendarza[1]

Przedstawiony tu program tworzy nowy plik RTF, zapisuje prolog dokumentu oraz kolejne akapity (poszczególne akapity

[1] Program tworzy kalendarz w języku angielskim. Nie zmieniłem tego, ponieważ pozwala to na omówienie pewnych funkcji, które nie są używane w polskich dokumentach, a które mogą być przydatne. Wtreści rozdziału umieściłem informacje, które pozwolą czytelnikom zmodyfikować kod samodzielnie — *przyp. tłum.*

są tworzone za pomocą pętli `while`) i zamyka dokument, dodając pojedynczy znak „}". Każdy akapit stanowi nagłówek dla strony kalendarza. Na poszczególnych stronach umieszczane są nagłówki dla kolejnych dni tygodnia. Gotowy kalendarz można wydrukować, dzięki czemu zawsze dostępny będzie przydatny terminarz do zapisywania notatek, terminów spotkań i podobnych informacji.

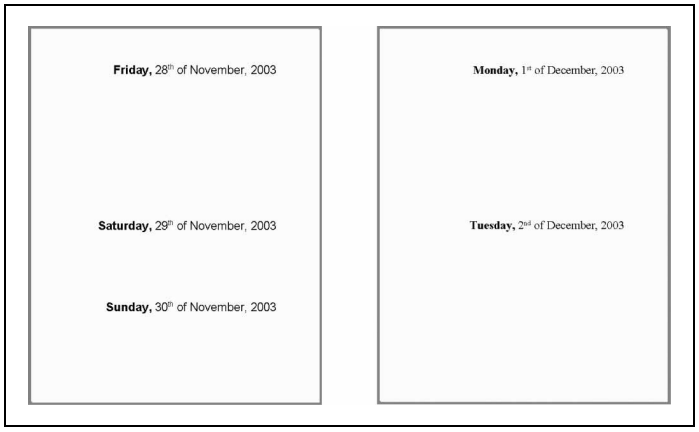
Aby uzyskać puste miejsce pomiędzy kolejnymi nagłówkami, należy użyć polecenia `\sa` z parametrem w postaci dość dużej liczby.

Poniżej przedstawiono fragment kodu źródłowego pliku RTF:

```
...
{\pard\sa6060\qr\f0{\b Friday,} 28{\super th} of November,
↳2003\par}
{\pard\sa2900\qr\f0{\b Saturday,} 29{\super th} of November,
↳2003\par}
{\pard\sa2900\qr\f0{\b Sunday,} 30{\super th} of November,
↳2003\par}
{\pard\sa6060\qr\f0{\b Monday,} 1{\super st} of December,
↳2003\par}
{\pard\sa6060\qr\f0{\b Tuesday,} 2{\super nd} of December,
↳2003\par}
{\pard\sa6060\qr\f0{\b Wednesday,} 3{\super rd} of December,
↳2003\par}
{\pard\sa6060\qr\f0{\b Thursday,} 4{\super th} of December,
↳2003\par}
...
```

Deklaracje poszczególnych akapitów różnią się — dla soboty i niedzieli przydzielane jest mniej miejsca (tylko 2900 twipów zamiast 6060 twipów). To ustawienie jest w programie kontrolowane przez zmienną `$space`, która otrzymuje wartość stałej `DAY_BIG` lub `DAY_SMALL` w zależności od wartości zmiennej `$weekday`. Dla każdego miesiąca stosowana jest inna czcionka. Odbywa się to poprzez dodanie do `\f` wyniku operacji `month % 2`, gdzie `%` jest operatorem modułu w Perlu. Oczywiście wszystkie używane

czcionki są definiowane w tablicy czcionek. Rysunek 3.1 przedstawia przykładowe strony kalendarza.



Rysunek 3.1. Przykładowe strony kalendarza

Z działaniem programu mogą być związane dwa drobne problemy, które mogą ujawnić się po dokonaniu zmian w kodzie źródłowym.

Na pierwszy potencjalny problem można natknąć się przy próbie utworzenia polskiego kalendarza. Poszczególne ciągi w dokumencie RTF nie przechodzą przez procedurę obsługi sekwencji ucieczki. Nie ma potrzeby używania takiej procedury w oryginalnym kodzie programu, ponieważ nie są używane znaki typu „\", „{", „}" lub 8-bitowe znaki. Jeżeli jednak konieczne będzie użycie polskich nazw miesięcy i dni tygodnia, należy zmienić odpowiednio wiersz `printf`. W takim przypadku w kodzie programu mogą pojawić się wiersze w następującej postaci:

```
{\pard\sa6060\qr\f1{\b Poniedziałek}, 3 grudnia 2003\par}
```

Większość procesorów tekstu odrzuci dokument RTF zawierający taki wiersz, gdyż użyto w nim znaku *ł*. Ponieważ jest to 8-bitowy znak, należy zastąpić go właściwą sekwencją ucieczki (w tym przypadku będzie to sekwencja `\'b3`) w sposób opisany w rozdziale 1.

Drugi problem jest związany z faktem, że program umieszcza w kodzie dokumentu RTF wartości numeryczne bez sprawdzenia, czy są one liczbami całkowitymi. W przedstawionym wcześniej fragmencie kodu użyto wartości, które z całą pewnością są liczbami całkowitymi (6060 i 2900). Wyobraźmy sobie jednak, że program został przystosowany do druku na papierze o innych rozmiarach (obecnie używany jest papier w formacie US letter). Większy odstęp pomiędzy kolejnymi nagłówkami ustawiono na 10 cm, a mniejszy odstęp na 5 cm.

Zgodnie z tabelą „Konwersja różnych jednostek długości na twipy" w rozdziale 4. ustalono, że 10 cm to 5669 twipów, dlatego kod programu przyjmie następującą postać:

```
use constant DAY_BIG   => 5669;
use constant DAY_SMALL => DAY_BIG / 2; # polowa wartosci DAY_BIG
```

Sprawi to, że stała `DAY_SMALL` uzyska wartość 2834,5, a kod źródłowy dokumentu RTF będzie wyglądał tak jak poniżej:

```
{\pard\sa5669\qr\f0{\b Friday,} 28{\super th} of November,
↳2003\par}
{\pard\sa2834.5\qr\f0{\b Saturday,} 29{\super th} of November,
↳2003\par}
{\pard\sa2834.5\qr\f0{\b Sunday,} 30{\super th} of November,
↳2003\par}
{\pard\sa5669\qr\f0{\b Monday,} 1{\super st} of December,
↳2003\par}
```

Ponieważ parametrami słów kluczowych RTF mogą być tylko liczby całkowite, polecenie `\sa2834.5` zostanie zinterpretowane jako `\sa2834` (czyli polecenie `\sa` z argumentem 2834) i dwa znaki `.5`, przez co na wydruku pojawią się nagłówki `.5Saturday, 29th of`

November, 2003 i .5Sunday, 30th of November, 2003. Do przekształcania wartości numerowych na postać liczb całkowitych w większości języków programowania (włącznie z Perlem) służy polecenie int, które może być użyte w następujący sposób:

```
print RTF
 '{\pard\sa%s\qr\f%s{\b %s,} %s{\super %s} of %s, %s\par}'."\n",
 int($space),
 int($month % 2),
```

Drugie polecenie int jest w gruncie rzeczy niepotrzebne, ponieważ operator % w Perlu *zawsze* zwraca wartość całkowitą, ale warto zachować ostrożność.

Programiści znający dobrze formaty (s)printf szybko odgadną, że identyczny efekt można uzyskać za pomocą formatu %d (liczba całkowita wyrażona dziesiętnie), który zastąpi polecenie int:

```
print RTF
 '{\pard\sa%d\qr\f%d{\b %s,} %s{\super %s} of %s, %s\par}'."\n",
 $space,
 $month % 2,
```

Ta metoda działa równie dobrze jak poprzednia, a wybór jednej z nich jest zależny od osobistych upodobań programisty.

Przedstawione tu podejście do generowania dokumentów RTF może być zaadaptowane w celu drukowania kart pytań i odpowiedzi zamiast kalendarza. Karty tego typu są często stosowane do nauczania języków obcych. Jeżeli dokument jest drukowany dwustronnie, program powinien umieścić wszystkie pytania na jednej stronie (na przykład „a book" lub „to read"), a odpowiadające im odpowiedzi na drugiej (na przykład „książka" lub „czytać"). Po przecięciu gotowej strony na pół pozwoli uzyskać dwie karty pytań i odpowiedzi. Program generujący takie karty może pobierać dane wejściowe z pliku zawierającego kolejne wiersze w postaci *pytanie=odpowiedź*, na przykład a book=książka lub to read=czytać.

Poniżej przedstawiono pełny kod programu generującego kalendarz:

```
#!/usr/bin/perl
require 5;
use strict;
use warnings;
# (Program nalezy uruchamiac w normalnej wersji
# Perla z optymalna kontrola bledow)

# Definicje stalych:
use constant SECONDS_IN_DAY => 24 * 60 * 60;
use constant DAY_BIG   => 6060; # w twipach
use constant DAY_SMALL => 2900; # w twipach

# Otworzenie pliku i zapisanie prologu
open RTF, ">datebook.rtf" or die $!;
print RTF '{\rtf1\ansi\deff0
{\fonttbl{\f0\fswiss Arial;}{\f1\froman Times New Roman;}}
\deflang1033\plain\fs50
';

# Definicje tablic wyszukiwania z nazwami
# miesiecy i dni tygodnia
my @months = qw(
 January February March      April   May      June
 July    August   September  October November December
);
my @dows  = qw(
 Sunday Monday Tuesday Wednesday Thursday Friday Saturday
);

# Kalendarz jest tworzony na caly rok od biezacego dnia
my $then = time();
my $end = $then + 366 * SECONDS_IN_DAY;

while($then <= $end) {
  my($year,$month,$day, $weekday) = (gmtime($then))[5,4,3,6];
  my $space = DAY_BIG;
```

```
  $space    = DAY_SMALL if $weekday == 0 or $weekday == 6;

  printf RTF
   '{\pard\sa%s\qr\f%s{\b %s,} %s{\super %s} of %s, %s\par}' .
   ↳"\n",
    $space,
    $month % 2, # zmiana czcionki dla kazdego miesiaca
    $dows[$weekday], $day, th($day), $months[$month], $year +
    ↳1900,
  ;

  $then += SECONDS_IN_DAY;
}

# Zakonczenie i zamkniecie pliku oraz wyjscie z programu
print RTF "}";
close(RTF);
exit;


sub th {
  # Funkcja zwracajaca wlasciwy przyrostek dla dnia
  # miesiaca, na przyklad 3 => "rd", dzieki czemu
  # nie zostanie uzyta data "February 3th"!

  my $n = abs($_[0] || 0);
  return 'th' unless $n and $n == int($n);
  $n %= 100;
  return 'th' if $n == 11 or $n == 12 or $n == 13;
  $n %= 10;
  return 'st' if $n == 1;
  return 'nd' if $n == 2;
  return 'rd' if $n == 3;
  return 'th';
}

__END__
```

Program wyświetlający zawartość katalogu

Przedstawiony w tej części rozdziału program odczytuje podany katalog (lub bieżący katalog, jeżeli nie podano innego) i tworzy wydruk jego zawartości, który jest umieszczany w tabelach dokumentu RTF. Innymi słowy, jest to bardziej rozbudowana wersja polecenia `ls -l` w Uniksie lub `dir` w systemie MS-DOS.

Podstawowa struktura tego programu jest bardzo prosta. Po uruchomieniu program sprawdza obecność argumentu wiersza poleceń (zmienna `$ARGV[0]`), w razie potrzeby odczytuje bieżący katalog, a następnie wywołuje trzy podprocedury (`rtf_start`, `rtfdir` i `rtf_end`), a następnie kończy pracę.

Generowanie kodu tabeli odbywa się w funkcji `file_row`, której argumentami są nazwa, wielkość i data modyfikacji pliku. Funkcja umieszcza te informacje w komórce tabeli zbudowanej z trzech kolumn; na przykład wiersz tabeli zawierającej komórki `24,544`, `clip1.gif` i `2003-03-23 03:07` jest wyrażany w postaci następującego bloku kodu źródłowego RTF:

```
\trowd\trgaph90\clbrdrt\brdrw15\brdrs\clbrdrb\brdrw15
\brdrs\clbrdrl\brdrw15\brdrs\clbrdrr\brdrw15\brdrs\cellx1224
\clbrdrt\brdrw15\brdrs\clbrdrb\brdrw15\brdrs\clbrdrl\brdrw15
\brdrs\clbrdrr\brdrw15\brdrs\cellx6120
\clbrdrt\brdrw15\brdrs\clbrdrb\brdrw15\brdrs\clbrdrl\brdrw15
\brdrs\clbrdrr\brdrw15\brdrs\cellx8280
\pard\intbl\qr{\fs18\f1\b 24,544}\cell
\ql{\f2 clip1.gif}\cell
\qc{\fs18 2003-03-23 03\'3a07}\cell \row
```

Na szczęście ten mało czytelny kod dokumentu RTF jest tworzony automatycznie poprzez wywołanie funkcji `file_row(`*`nazwa, wielkość, data_modyfikacji`*`)`. Znaczenie poszczególnych poleceń do generowania tabel zostało przedstawione szczegółowo w podrozdziale „Tabele” w rozdziale 1.

Podprocedury `esc`, `in` i `cm` można wykorzystać we własnych programach. Pierwsza podprocedura implementuje prostą funkcję do obsługi sekwencji ucieczki, natomiast podprocedury `in` i `cm` pozwalają na podawanie wszystkich długości w calach lub centymetrach zamiast w twipach. Działanie obu podprocedur można sprawdzić w funkcji `file_row`, w której poszczególne polecenia `\cellx`*PrawyKoniec* zostały oparte na liście `in(.85), in(4.25), in(5.75)`. Z pewnością jest to znacznie wygodniejsza metoda definiowania odległości niż lista `1224, 6120, 8280`, która wymaga ręcznego przeliczania centymetrów lub cali na twipy.

Wybór funkcji `in(`*długość*`)` lub `cm(`*długość*`)` jest zależny od preferencji użytkownika, ponieważ jednostki długości i tak muszą zostać przeliczone na twipy przed utworzeniem tabeli. Warto porównać tę metodę ze sposobem użyciem polecenia `int` we wcześniejszym przykładzie programu generującego kod RTF.

Poniżej przedstawiono pełny kod źródłowy omawianego programu:

```
#!/usr/bin/perl
require 5;
use strict;
use warnings;
# (Program nalezy uruchamiac w normalnej wersji
# Perla z optymalna kontrola bledow)

my $dir = $ARGV[0] || '.';
 # Jezeli nie podano argumenta wiersza polecenie,
 # nalezy wyswietlic zawartosc biezacego katalogu

if($dir eq '.' or $dir eq './') {
  use Cwd; # zaladowanie biblioteki funkcji getcwd
  $dir = getcwd();
  # Teraz mozna wyswietlic pelna sciezke, a nie tylko "."
}
```

```
# Przerwanie pracy, jesli nie jest to poprawny katalog!
die "Nie mozna odczytac katalogu $dir"
 unless -e $dir and -d _ and -r _;

# Wywolanie podprocedur wykonujacych wszystkie
# operacje i wyjscie z programu:
rtf_start( $dir );
rtfdir( $dir );
rtf_end( $dir );
exit;

#-----------------------------------------------------------------
# Trzy podprocedury pomocnicze:

sub in { int(1440 * $_[0]) }  #       cale -> twipy

sub cm { int( 567 * $_[0]) }  # centymetry -> twipy

sub esc {   # zwrocenie danego ciagu jako poprawny kod RTF
  my $in = $_[0];
  $in =~ s{([^-"'?,\$\._a-zA-Z0-9 ])}
          {sprintf("\\'%02x",ord($1))}eg;
   # Procedura wykonuje wiecej operacji niz jest to wymagane,
   # ale nie obsluguje Unicode -- patrz rozdzial 1.
  return $in;
}
#-----------------------------------------------------------------

# Deklaracja trzech zmiennych globalnych do zapisywania danych
↳pliku:
my( @Items, %Size_of, %Time_of );

sub rtfdir {       # wykonanie glownego zadania programu
  my $dir = $_[0];
  scan_dir($dir);  # pobranie danych

  print '{\pard\sa200\qc \fs50\b\i ', esc($dir), ' \par}', "\n\n";
   # naglowek strony: nazwa katalogu
```

```
  file_row($_, $Size_of{$_}, $Time_of{$_} ) foreach @Items;
   # jeden wiersz dla kazdego pliku

  print '{\pard\sb1440\sa1440\fs28\b\i [Pusty!]}' unless @Items;

  return;
}

#-------------------------------------------------------------

sub scan_dir {
  #  Zebranie danych pliku
  #   (uzywane przez funkcje rtfdir)
  # Wykorzystywane sa typowe funkcje systemu plikow.
  #
  my $dir = $_[0];
  opendir(IN, $dir) or die "Nie mozna otworzyc $dir: $!";
  my( $item, $full_path );

  while(defined( $item = readdir(IN) )) {
    next if $item eq '.' or $item eq '..';
    push @Items, $item;
    ( $Size_of{$item}, $Time_of{$item} ) = (-1,-1);

    my $full_path = "$dir/$item";
    ( $Size_of{$item}, $Time_of{$item} ) =
    ↳( stat($full_path) )[7,9]
     if -f $full_path;
  }

  closedir(IN);
  @Items = sort by_name @Items;
    # Mozna zmienic "by_name" na "by_time" lub "by_size"
  return;
}

# Trzy sposoby sortowania:
sub by_name { use locale; uc($a) cmp uc($b) }
```

```
sub by_time { $Time_of{$a} <=> $Time_of{$b} }
sub by_size { $Size_of{$a} <=> $Size_of{$b} }

#-------------------------------------------------------------

sub file_row {
  # Wygenerowanie wiersza tabeli z nazwa, wielkoscia i data pliku
  #  (uzywane przez funkcje rtfdir)

  my($name, $size, $time) = @_;
  my $name_format = '';
  if($size == -1) { # Nasz znacznik dla katalogu
    $name_format = '\b '; # pogrubienie
    $time = '';
    $size = '';

  } else { # To jest plik...
    $time = format_date($time);
    1 while $size =~ s/(\d)(\d\d\d(,|$))/$1,$2/
     # Dodanie przecinkow do wielkosci pliku: 12345 => "12,345"
  }

  print '\trowd\trgaph90'; # Poczatek deklaracji wiersza RTF
  my $borders = join "", map '\clbrdr' . $_ . '\brdrw15\brdrs',
    qw(t b l r); # wlaczenie wszystkich krawedzi

  print $borders, '\cellx', $_, "\n"
    for in(.85), in(4.25), in(5.75);   # deklaracje prawej
    krawedzi komorki
      # wszystkie wartosci mozna dowolnie zmieniac

  print    # emisja komorek i ich zawartosci:
   '\pard\intbl\qr{\fs18\f1\b ',  esc($size) ,'}\cell ',
    # wyrownanie do prawej, 9 punktow, Courier, pogrubienie
   '\ql{\f2 ', $name_format, esc($name), '}\cell ',
    # wyrownanie do lewej, Times Roman
   '\qc{\fs18 ', esc($time), '}\cell ',
    # wyposrodkowanie, 9 punktow, Arial
```

```
    '\row', "\n\n"
  ;

  return;
}
#-------------------------------------------------------------

sub rtf_start {
  my $dir = $_[0];
  print <<'END_RTF_START';
{\rtf1\ansi\deff0
{\fonttbl {\f0\fswiss Arial;}{\f1\fmodern Courier New;}
{\f2\froman Times;}}
\deflang1024\widowctrl\plain\fs24\noproof

END_RTF_START

  # Naglowek strony zawierajacy sciezke i godzine:
  print '{\header \pard\qr\plain\f0\fs18\i\noproof ',
    esc($dir . ' ' . format_date(time)),
    ' - p\chpgn\par}', "\n\n";
  return;
}

sub rtf_end {
  my $dir = $_[0];
  print "\n}\n";
  return;
}

sub format_date {
  my $time = $_[0];
  my($Y,$M,$D,$h,$m) = ( localtime( $time ) )[ 5,4,3,2,1 ];
  return sprintf "%04d-%02d-%02d %02d:%02d", $Y+1900, $M+1, $D,
  ↳$h,$m;
}

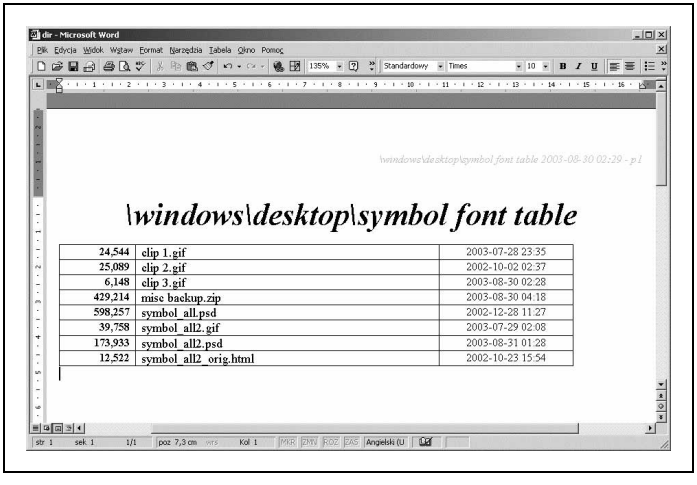
__END__
```

Program domyślnie nie otwiera nowego pliku, a jedynie wysyła kod RTF do standardowego wyjścia. Użytkownik może jednak przekierować wyjście w powłoce. Poniżej pokazano sposób wykonania tej operacji w systemie Windows:

```
C:\> perl -S rtfdir.pl "\windows\desktop\symbol font table" >
↳%TEMP%\dir.rtf

C:\> start %TEMP%\dir.rtf
```

Rysunek 3.2 przedstawia uzyskany dokument w oknie *MS Word*.



Rysunek 3.2. Wynik programu rtfdir

Program do projektowania kopert na płyty (origami)

Kolejny przykładowy program generuje pojedynczą stronę, która po właściwym zagięciu może posłużyć jako koperta dla płyty

CD. To narzędzie będzie szczególnie przydatne dla osób, które zawsze mają więcej nagranych płyt CD-R niż pustych pudełek na nie.[2]

Po uruchomieniu program oczekuje na wprowadzenie informacji, które zostaną wydrukowane na etykiecie, a następnie generuje plik RTF z etykietą umieszczoną we właściwym miejscu strony. Drukowane są również linie wskazujące miejsce zagięcia strony. Sposób tworzenia koperty na płytę i kolejność zaginania strony przedstawiono dokładnie na rysunkach od 3.5 do 3.8.

Program wykorzystuje dwie zaawansowane funkcje RTF, a mianowicie dokładne pozycjonowanie akapitów i rysowanie linii.

Tekst jest rozmieszczany za pomocą poleceń `\pvpg\phpg \posyN \abswW \abshN`, które zostały przedstawione w podrozdziale „Dokładne pozycjonowanie akapitów" w rozdziale 1. Bez funkcji dokładnego pozycjonowania akapitów nie ma żadnej gwarancji, że tekst będzie widoczny po zagięciu strony. Ta sama metoda jest zresztą używana do drukowania tekstu na naklejkach i kopertach.

Bardzo ważną rolę odgrywa polecenie `\absh-`*`DokładnaWysokość`*. Jeżeli użytkownik wprowadzi zbyt dużo informacji, program ukryje nadmiarowy tekst. Co więcej, wszystkie informacje, jakie są wpisywane przez użytkownika, przechodzą przez procedurę `esc`, która zapewnia obsługę nietypowych znaków. Jeżeli więc zostanie podany tekst `jakieś pliki`, program automatycznie zmodyfikuje go, przez co ciąg wynikowy będzie miał prawidłową postać `jakie\'9c pliki`.

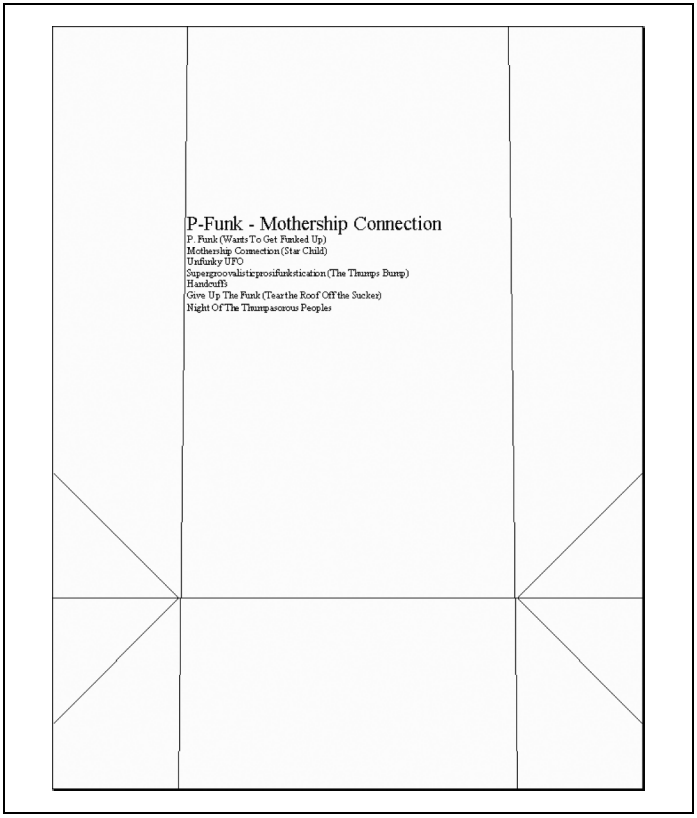
[2] Chciałbym podziękować Tomowi Hullowi z wydziału matematyki Merrimack College, który zaprojektował sposób zaginania strony. Więcej informacji można znaleźć pod adresami *http://web.merrimack.edu/hullt/* i *http://papercdcase.com/*.

Konstrukcja służąca do rysowania linii jest bardziej rozbudowaną wersją analogicznej konstrukcji z podrozdziału „Rysowanie linii" w rozdziale 1. Generowanie kodu RTF dla linii przechodzącej przez dwa punkty jest bardziej skomplikowane niż interpolacja dwóch par punktów (*x*,*y*), ponieważ konieczne jest wykonanie pewnych obliczeń matematycznych. Kod wykonujący to zadanie został umieszczony w podprocedurze `line`, która jest wywołana w różnych miejscach programu.

Rysunek 3.3 przedstawia wygląd pliku wygenerowanego przez przykładowy program.

Niestety, prostsze edytory tekstu nie obsługują niektórych zaawansowanych funkcji RTF, które służą, na przykład, do dokładnego pozycjonowania akapitów i rysowania linii. Prawidłowo odczytywany jest jedynie tekst znajdujący się w otwieranym pliku. Rysunek 3.4 przedstawia ten sam dokument w oknie programu *TextEdit*, czyli prostego edytora tekstu, który stanowi część systemu Mac OS X.

Czytelnik może się zastanawiać, dlaczego w programie nie są używane konkretne wartości, jak na przykład `line(0, 11880, 12240, 11880)`, ale bardziej skomplikowane konstrukcje typu `line(0, Page_Height*3/4, Page_Width, Page_Height*3/4)`. Istnieją dwie przyczyny, dla których zastosowano takie rozwiązanie. Po pierwsze, użycie takich wyrażeń pozwala lepiej zrozumieć geometrię zagięć i sposobów złożenia origami (w tym celu warto złożyć i ponownie rozłożyć kopertę, a następnie przyjrzeć się liniom składania). Druga przyczyna jest bardziej praktyczna — przy zmianie formatu papieru z US letter na A4 konieczna jest zmiana wszystkich rozmiarów kartki papieru. To zadanie można sobie znacznie ułatwić, jeżeli zamiast konkretnych wartości zdefiniowano zmienne, ponieważ wyrażenia typu `Page_Height*3/4` ułatwiają błyskawiczne przeliczanie rozmiarów.

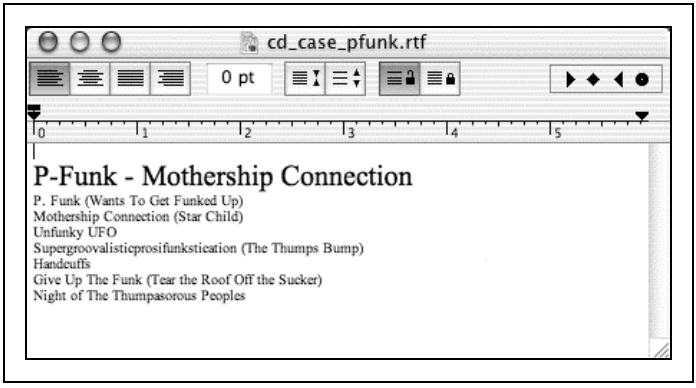


Rysunek 3.3. Koperta origami na płytę CD

Program może odczytywać dane wejściowe z pliku, na przykład:

```
% cdcase tytul_i_sciezki.txt
```

Oczywiście można również użyć standardowego wejścia, na przykład:



Rysunek 3.4. Wygląd dokumentu w programie TextEdit

```
% cdcase < tytul_i_sciezki.txt
```

lub:

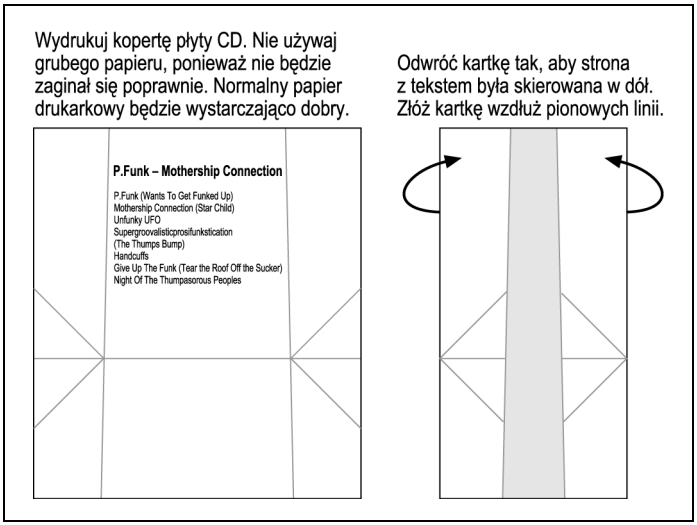
```
% ls -l ./backups5 | cdcase
```

Program może również odczytywać dane bezpośrednio z konsoli:

```
% cdcase
Wprowadz tekst dla okladki, a nastepnie nacisnij Enter i EOF
↳(control+z)
Tytuł płyty CD
Tytuł ścieżki 1
Tytuł ścieżki 2
Tytuł ścieżki 3
[Po zakończeniu należy nacisnąć control+D lub control+Z w nowym
↳wierszu]
```

Niezależnie od wybranej metody, program zapisze informacje o okładce płyty CD do pliku *cd_case.rtf*. Wyjście z programu odbywa się za pomocą klawiszy *Ctrl+C*. Jeżeli nie podano żadnych informacji, program utworzy pustą kopertę, czyli stronę bez nadruku, ale ze wszystkimi liniami zagięcia. Plik wynikowy można otworzyć w procesorze tekstu i dokonać edycji etykiety

lub natychmiast go wydrukować. Sposób złożenia koperty przedstawiono na rysunkach od 3.5 do 3.8.



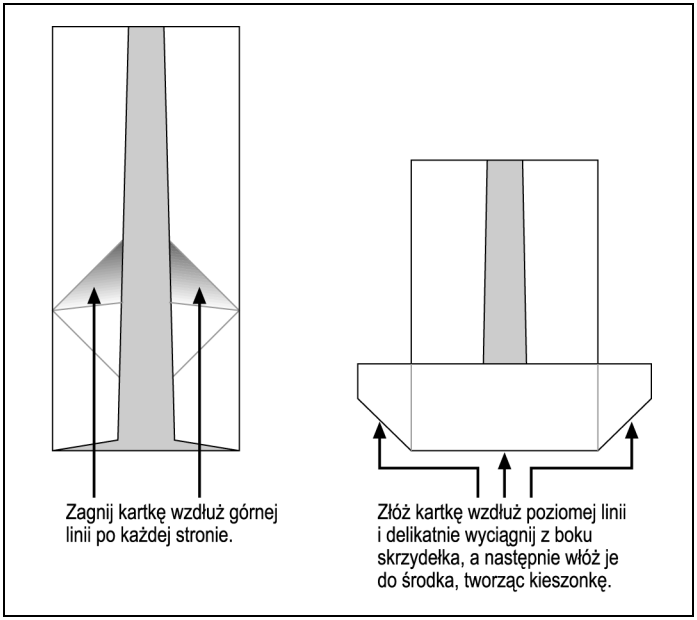
Rysunek 3.5. Krok 1 — pionowe złożenie kartki

Poniżej przedstawiono pełny kod programu:

```
#!/usr/bin/perl
require 5;
use strict;
use warnings;
# (Program nalezy uruchamiac w normalnej wersji
# Perla z optymalna kontrola bledow)

#---------------------------------------------------------------
# Deklaracja stalych:

use constant A4 => 0;  # Zmien 0 na 1, jesli drukujesz na
↳papierze A4
```



Rysunek 3.6. Krok 2 — poziome złożenie kartki

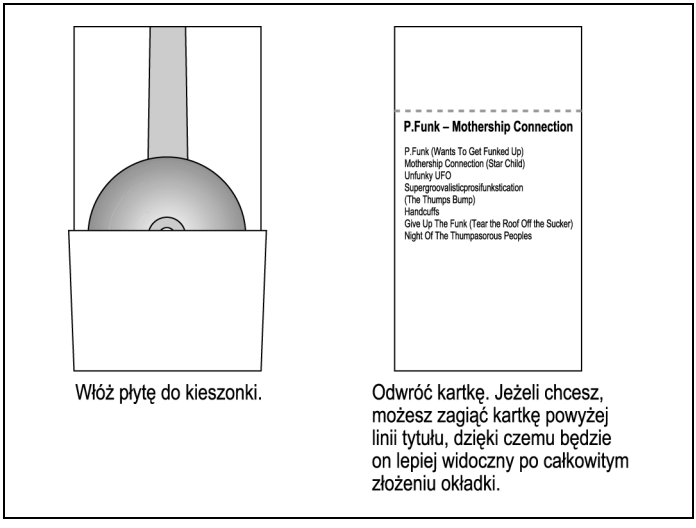
```
use constant Inches2twips => 1440; # (wspolczynnik konwersji)

use constant Page_Width  =>  A4 ? 11909 : (8.5 * Inches2twips);
use constant Page_Height =>  A4 ? 16834 : ( 11 * Inches2twips);

use constant Smidgen     => 1/8 * Inches2twips; # uwzglednienie
                                # niedokladnosci przy skladaniu
use constant CD_Diameter => Smidgen + 4.75 * Inches2twips;
use constant Wing_Size   => (Page_Width - CD_Diameter) / 2;

use constant First_Line_Height =>  40 * 10;

#-------------------------------------------------------------
# Deklaracja dwoch zmiennych globalnych:
```



Rysunek 3.7. Krok 3 — włożenie płyty CD

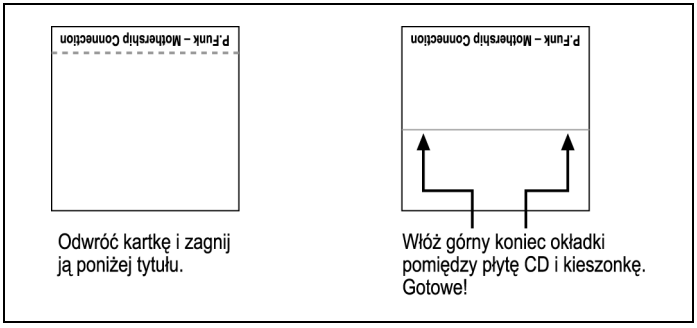
```
my $out = "cd_case.rtf";
my @lines;

# Glowna czesc programu:

get_input();
open(RTF, ">$out") or die "Nie mozna otworzyc $out do zapisu: $!";
doc_intro();
draw_fold_guides();
print_text();
close(RTF);
print "\nUtworzono $out (", -s $out, " bajtow)\n";
exit;

#-----------------------------------------------------------------

sub get_input {
```



Rysunek 3.8. Krok 4 — zamknięcie koperty

```
    print "Wprowadz tekst dla okladki, a nastepnie nacisnij Enter
    ↪i EOF (",
      ($^O =~ m/MSWin/) ? "control+z)\n" : "control+d)\n"
     unless @ARGV;
      # Wybrany klawisz jest zalezny od systemu operacyjnego.
      #  Windows uzywa control+z, natomiast wszystkie inne
      #  systemy domyslnie wymagaja control+d.

    @lines = <>;  # Pobranie danych wejsciowych (z pliku,
     # standardowego wejscia lub konsoli)

    unless( grep m/\S/, @lines) { # brak danych wejsciowych
      print "\nNic nie wpisales? Zrobimy wiec pusta okladke.\n";
      @lines = ( '_' ); # pusty wiersz
    }

    chomp(@lines); # usuniecie znakow \n
    return;
}

#-------------------------------------------------------------

sub doc_intro { # Poczatek pliku RTF

    printf RTF '{\rtf1\ansi
```

```
\deff0{\fonttbl {\f0 \froman Times New Roman;}}
\paperw%s \paperh%s \deflang1033\plain\f0\fs20

{\pard\par}

', int(.5 + Page_Width), int(.5 + Page_Height);

  return;
}

#-------------------------------------------------------------

sub print_text {
  # Zapisanie zawartosci @lines w precyzyjnie rozmieszczonym
  # akapicie z uzyciem podprocedury esc($line).

  printf RTF '{\pard \pvpg\phpg
\posx%s \posy%s
\absw%s \absh-%s
',  map int($_ + .5), # uzycie tylko liczb calkowitych

    Wing_Size + Smidgen,
    Page_Height - Page_Height * 1/4
     - CD_Diameter - First_Line_Height,
    CD_Diameter - 2 * Smidgen,
    CD_Diameter - 2 * Smidgen + First_Line_Height,
  ;

  print RTF "\n\\f0\\fs20{\\fs40 ",
    # Powiekszenie pierwszego wiersza
    @lines ? esc(shift @lines) : '',
    "}\n";

  foreach my $line (@lines) {
    print RTF "\n\\line ", esc($line), "\n";
  }

  print RTF "\n\\par}\n}"; # koniec dokumentu
  return;
}
```

```
#-------------------------------------------------------------

sub esc {   # zwrocenie ciagu jako poprawny kod RTF
  my $in = $_[0];
  $in =~ s{([^-"'?,\$\._a-zA-Z0-9 ])}
          {sprintf("\\'%02x",ord($1))}eg;
   # Uzywane jest wiecej sekwencji ucieczki niz jest to wymagane
  return $in;
}

#-------------------------------------------------------------

sub draw_fold_guides { # rysowanie wszystkich linii!

  # rysowanie ------- na dole
  line(0,           Page_Height*3/4,
       Page_Width,  Page_Height*3/4 );

  # rysowanie  |  |   od gory do dolu
  line(Wing_Size + Smidgen/2, 0,
       Wing_Size, Page_Height );
  line(Page_Width - Wing_Size - Smidgen/2, 0,
       Page_Width - Wing_Size,  Page_Height );

  # rysowanie \    /  wedlug dolnej linii
  #           /    \
  line( Wing_Size, Page_Height*3/4,
        0, (Page_Height*3/4) + Wing_Size );
  line( Wing_Size, Page_Height*3/4,
        0, (Page_Height*3/4) - Wing_Size );
  line( Page_Width - Wing_Size, Page_Height*3/4,
        Page_Width, (Page_Height*3/4) + Wing_Size );
  line( Page_Width - Wing_Size, Page_Height*3/4,
        Page_Width, (Page_Height*3/4) - Wing_Size );
  return;
}

#-------------------------------------------------------------
# Podprocedura generujaca kod RTF dla linii
# pomiedzy dwoma punktami na stronie
```

```
my $z_count;  # zmienna licznika

sub line {  # rysowanie linii (x,y) do (x2,y2)
  my($x1,$y1,$x2,$y2) = map int(.5 + $_), @_;

  # Sprawdzenie istnienia czterech parametrow
  use Carp;
  croak "line() wymaga czterech parametrow" unless @_ == 4;

  # Uzyskanie wspolrzednych lewego gornego rogu strony
  my $nw_x = ($x1 < $x2) ? $x1 : $x2;
  my $nw_y = ($y1 < $y2) ? $y1 : $y2;

  $x1 -= $nw_x; $x2 -= $nw_x; $y1 -= $nw_y; $y2 -= $nw_y;
  # Wspolrzedne (x1,y1) i (x2,y2) odnosza sie do lewego gornego
  ↳rogu

  my $w = abs($x1 - $x2); # odleglosc w poziomie
  my $h = abs($y1 - $y2); # odleglosc w pionie
  $z_count++;

  printf RTF '{\*\do\dobxpage\dobypage\dodhgt%s\dpline
\dpptx%s\dppty%s \dpptx%s\dppty%s
\dpx%s\dpy%s \dpxsize%s\dpysize%s
\dplinew%s \dplinecor%s\dplinecog%s\dplinecob%s
}

',  $z_count,
    $x1,$y1, $x2,$y2,
    $nw_x, $nw_y, $w, $h,
    15,  # grubosc linii
    0,0,0, # kolor RGB linii
  ;
  return;
}

__END__
```

Narzędzie do pobierania metadanych RTF

W podrozdziale „Struktura dokumentu" w rozdziale 1. poznaliśmy metadane dokumentu, które mogą być wyrażone w postaci grupy `\info`. Są to takie metadane jak {`\author` *nazwa autora*} i {`\title` *tytuł dokumentu*}. W tej części książki poznamy prosty program, który wyszukuje pola metadanych w dokumentach RTF.

Załóżmy, że użytkownik otrzymał zadanie skatalogowania cyfrowej biblioteki z artykułami naukowymi. W bibliotece znajduje się kilka tysięcy takich artykułów w postaci plików RTF, a ich pełne tytuły są zapisane w polu {`\title` *tytuł dokumentu*}. Konieczne jest pobranie tytułu każdego dokumentu i przekazanie go do głównego katalogu artykułów. Typowy dokument rozpoczyna się od następującego kodu RTF, który jest generowany przez program *MS Word*:

```
{\rtf1\ansi\ansicpg1252\uc1 \deff27\deflang1033\deflangfe1033
{\fonttbl{\f0\froman\fcharset0\fprq2{*\panose
02020603050405020304}Times New Roman;}{\f27\froman\fcharset0
\fprq2{*\panose 02040502050405020303}Georgia;}}{\colortbl;
\red0\green0\blue0;\red0\green0\blue255;}{\stylesheet{\q1 \li0
\ri0\widctlpar\aspalpha\aspnum\faauto\adjustright\rin0\lin0
\itap0 \f27\fs22\lang1033\langfe1033\cgrid\langnp1033
\langfenp1033 \snext0 Normal;}{*\cs10 \additive Default
Paragraph Font;}}{\info{\title Spectroscopic study of blue com
pact galaxies. III. Empirical population synthesis}{\author
.}{\keywords dwarf, galaxies: evolution, galaxies: stellar con
tent, galaxies: star clusters}{\operator Xenotypo GmBH}{
\creatim\yr2003\mo6\dy11\hr2\min44}{\revtim\yr2003\mo6\dy11
\hr3\min29}{\version3}{\edmins2}{\nofpages1}{\nofwords0}
{\nofchars0}{*\company .}{\nofcharsws0}{\vern8247}}\widowctrl
\ftnbj\aenddoc\noxlattoyen\expshrtn\noultrlspc\dntblnsbdb
```

Program użytkownika musi pobrać tytuł tego dokumentu, czyli *Spectroscopic study of blue compact galaxies. III. Empirical population synthesis*.

Gdyby *Word* umieszczał każde pole metadanych w oddzielnym wierszu, wystarczyłoby użyć narzędzia *grep* do wyszukania ciągu `\title`, a następnie odczytać żądany wiersz w postaci `{\title Spectroscopic study of blue compact galaxies. III. Empirical population synthesis}`. Niestety, *MS Word*, podobnie jak większość procesorów tekstu, nie tworzy uporządkowanego i czytelnego kodu RTF. Oznacza to, że narzędzia typu *grep* nie będą zbyt przydatne.

Skutecznym rozwiązaniem tego problemu będzie wczytanie całego dokumentu do pamięci i wyszukanie tekstu, który jest zgodny ze wzorcem `{\title` *jakiś tekst*`}`. Ten sam wzorzec można wyrazić jako wyrażenie regularne Perla — `m/\{\\title\s*([^\}]+)\}/`.

Niektóre dokumenty mogą jednak zawierać osadzone rysunki, przez co ich wielkość znacznie wzrasta. Należy jednak sobie przypomnieć, że grupa `\info` (zawierająca pole `\title` i wszystkie inne metadane) zawsze znajduje się na początku dokumentu, przed jakimikolwiek danymi. Teoretycznie również sekcje pliku RTF umieszczone przed grupą `\info` mogą być bardzo duże, ale w praktycznie wszystkich przypadkach ta grupa znajduje się w pierwszych 10 kB pliku. Wystarczy więc odczytać do pamięci tylko początkowy fragment dokumentu i wykonać żądaną operację.

Utworzyliśmy już wyrażenie warunkowe `m/\{\\title\s*([^\}]+)\}/` do wyszukiwania tekstu i odkryliśmy sposób badania dokumentów. Pozostała część rozwiązania wymaga jedynie otworzenia każdego dokumentu, którego nazwa została podana w wierszu poleceń. Odczytany tytuł musi być przekazany przez procedurę zastępującą sekwencję ucieczki właściwymi znakami:

```
#!/usr/bin/perl

use strict;

foreach my $in (@ARGV) {
  next unless -f $in and -r _; # Pominiecie nieczytelnych plikow
```

```
  open IN, $in  or  warn( "Nie mozna otworzyc $in do odczytu:
  ↳$!\n" ), next;
  read IN, $_, 10_000 or warn("Nie mozna dokonac odczytu z $in:
  ↳$!\n"), next
  close(IN);
  m/^\{\\rtf/s  or  warn( "To nie jest plik RTF: $in\n" ), next;

  m/\{\\title\s*([^\}]+)\}/s or warn("Brak tytulu w $in\n"), next;
  print "T $in: ", unesc($1), "\n";
}

sub unesc {
  my $x = $_[0];
  $x =~ s/[\cm\cj]//g;  # usuniecie niepotrzebnych znakow nowego
  ↳wiersza
  $x =~ s/\\'([a-fA-F0-9]{2})/pack("C", hex($1))/eg; #
  ↳dekodowanie \'xx
  return $x;
}

__END__
```

Większa część kodu jest odpowiedzialna za obsługę rzadkich lub nietypowych błędów. Program sprawdza, czy każdy plik *.rtf* można otworzyć i odczytać, a także czy zawiera na początku sekwencję znaków `{\rtf`. Najważniejsza operacja jest wykonywana przez wiersz zawierający wyrażenie regularne `m/\{\\title\s*([^\}]+)\}/s`, które zapisuje tytuł dokumentu do zmiennej `$1`. Uzyskany tytuł jest wyświetlany na ekranie.

Poniżej przedstawiono typową sesję pracy z programem, który zwraca tytuł dokumentu przekazanego poprzez wiersz poleceń:

```
% perl rtf_title.pl astro381.rtf
astro381.rtf Spectroscopic study of blue compact galaxies. III.
↳Empirical population synthesis
```

Uwagi na temat analizy plików RTF

Należy zwrócić uwagę, że mogą wystąpić pewne problemy z wyrażeniem regularnym `m/\{\\title\s*([^\}]+)\}/`, jakie zostało użyte do implementacji koncepcji wzorca `{\title jakiś tekst}`. Tworząc kod programu, przyjęto założenie, że tytuł nie może zawierać znaków „}" ani żadnych innych kodów formatujących (patrz rozdział 1.). Oznacza to, że nigdy nie będzie możliwe użycie tytułu `{\title Katastrofa {\i Challengera}}`, a jedynie `{\title Katastrofa Challengera}`. Użyte w naszym programie wyrażenie regularne zawiedzie, jeżeli znak „}" nie został wyrażony jako `\'7d` (co jest zalecane w niniejszej książce), ale jako `\}` (co również jest dozwolone w specyfikacji standardu RTF).

Wyobraźmy sobie, że otrzymaliśmy dokument zatytułowany *Optimizing {n,xN} Grammars*. Tytuł ten można wyrazić jako `{\title Optimizing \'7bN,xN\'7d Grammars}`. W takim przypadku wyrażenie regularne odszuka ciąg `Optimizing \'7bN,xN\'7d Grammars`, a podprocedura `unesc($1)` prawidłowo zastąpi sekwencje ucieczki, dzięki czemu uzyskamy pełny tytuł dokumentu. Równie dobrze można wyrazić ten sam tytuł w postaci `{\title Optimizing \{N,xN\} Grammars}`. Wyrażenie regularne odszuka teraz jedynie początkowy ciąg `Optimizing \`. Najprostszym sposobem rozwiązania tego problemu będzie upewnienie się, że żaden dokument nie został zakodowany w ten sposób. W tym celu wystarczy jedynie wyszukać sekwencję znaków `\{`. Na szczęście znaki „{" i „}" zwykle stosowane jedynie w matematyce i informatyce, dlatego problem występuje dość rzadko. Pozostawiam Czytelnikowi zadanie utworzenia wyrażenia regularnego, które w elegancki sposób rozwiąże ten problem, a jednocześnie w dalszym ciągu będzie zatrzymywało się po odczytaniu tylko znaku „{".

Znacznie poważniejszym problemem związanym z implementacją wzorca `{\title jakiś tekst}` jako wyrażenia regularnego `m/\{\\title\s*([^\}]+)\}/` jest fakt, że prawidłowo odczytywane są jedynie pola `\title` i inne metadane dokumentu. Nie jest

możliwe zastosowanie tego podejścia dla innych struktur RTF, w których wykorzystywane są kody formatujące i zagnieżdżone grupy.

Załóżmy, że konieczne jest przechwycenie zawartości wszystkich przypisów w dokumencie. Po zapoznaniu się z opisem kodu przypisów w rozdziale 1. użytkownik zadecydował, że wystarczy wykorzystać wyrażenie regularne, zastępując słowo kluczowe `title` przez słowo `footnote`. W efekcie uzyskano wyrażenie `m/\{\\footnote\s*([^\}]+)\}/`. Zastanówmy się jednak, co się stanie po odnalezieniu następującego kodu:

```
{\footnote\pard\plain\chftn
: See {\i Navajo Made Easier} by Irvy Goosen}
```

Użyty wzorzec zatrzyma się na pierwszym znaku „}", dlatego odczytany zostanie tylko ciąg: `See {\i Navajo Made Easier`, natomiast dalsza część przypisu (`by Irvy Goosen`) zostanie pominięta. Niestety, nie da się rozwiązać tego problemu poprzez modyfikację wyrażenia regularnego, gdyż takie wyrażenia nie obsługują zagnieżdżonych grup. W tym przypadku konieczne jest dopasowanie znaku „}" do otwierającego znaku „{" z uwzględnieniem wszystkich zagnieżdżonych grup `{...}`, jakie znajdują się wewnątrz tej pary znaków.

W takiej sytuacji najlepszym rozwiązaniem będzie użycie prawdziwego analizatora kodu RTF. Istnieją dwa podejścia do analizy takiego kodu. Bardzo proste narzędzia po prostu odczytują kolejne polecenia kodu źródłowego, natomiast złożone aplikacje kompilują kod RTF do postaci rozbudowanych drzew, których struktura reprezentuje poszczególne elementy dokumentu (na przykład akapity i tabele). Nie oznacza to jednak, że taki program zapewni informacje o żądanej strukturze grupy w pliku. Dalsza dyskusja na temat obu metod analizy wykracza jednak poza tematykę niniejszej książki, gdyż jest to głównie domena interfejsów API konkretnych bibliotek programistycznych w wybranym języku.