Klausur UNIX-Datenbanken II

WS 1999/2000

Beispiellösung

Name _____

Vorname _____

Matrikel-Nr. _____

Studiengang _____

Semester _____

Aufgabe		Punkte	Bewertung
1	(Datenstrukturen)	16	
2	(Doppelte Worte)	14	
3	(Tabellen)	16	
4	(RegExp und Dateien)	10	
5	(Semantik von **my**)	12	
6	(DBI)	18	
7	(OO-Techniken)	14	
Summe		**100**	

Aufgabe 1 (16 Punkte)

In einer Logdatei werden Bestellungen, die bei Ihrem virtuellen Geschäft eingegangen sind, in einfacher Form verzeichnet. In jeder Zeile steht zu Beginn der Benutzername des Käufers, gefolgt von einer positiven Zahl (der Quantität) und dem Produktnamen. Die drei Felder sind durch Doppelpunkte getrennt. So könnte eine sehr kurze Logdatei aussehen:

```
StarshipHunter:1:Freighter
Rastur:1:Space Spice
ThePlunderers:2:Pedigree Bolts
StarshipHunter:2:Clipper
Rastur:3:Clipper
ThePlunderers:1:Spy Eye
```

Entwickeln Sie ein Perl-Skript, das Kaufempfehlungen ausgibt: Für einen vorgegebenen Produktnamen soll es alle anderen Produkte ausgeben, die von einem Käufer zusammen mit dem vorgegebenen Produkt gekauft worden sind. Die Logdatei wird über die Standardeingabe zur Verfügung gestellt. Sie dürfen von der Korrektheit des Eingabeformats ausgehen.

Beispiel: Würde bei obigem Log dem Perl-Skript der Produktname ‘‘Clipper’’ gegeben werden, so sollte ‘‘Freighter’’ und ‘‘Space Spice’’ ausgegeben werden, weil diese Produkte jeweils von jemanden gekauft worden sind, der auch ‘‘Clipper’’ gekauft hat.

Lösung:

```
#!/usr/local/bin/perl -w

use strict;

die "Usage: $0 product\n" unless @ARGV == 1;
my $product = shift @ARGV;

my %bought_by_customer = ();
my %bought_by_product = ();
while (<>) {
   chomp;
   my ($customer, $quantity, $product) = split /:/;
   $bought_by_customer{$customer}->{$product} = 1;
   $bought_by_product{$product}->{$customer} = 1;
}
exit 0 unless defined $bought_by_product{$product};
my %products = ();
foreach my $customer (keys %{$bought_by_product{$product}}) {
   foreach my $prod (keys %{$bought_by_customer{$customer}}) {
      $products{$prod} = 1 if $prod ne $product;
   }
}
print join("\n", sort keys %products), "\n";
```

Aufgabe 2 (14 Punkte)

Schreiben Sie ein Perl-Programm, das eine Reihe von Dateinamen als Parameter über die Kommandozeile erhält und aus den entsprechenden Dateien alle Zeilen ausgibt, in denen das gleiche Wort mehrfach direkt hintereinander vorkommt. Wird kein Parameter angegeben, sollte von STDIN gelesen werden.

```
thales$ cat a.txt
a aaa aa
aaa aa aa aaaa
thales$ cat b.txt
b bbb bbbb bb bbb
bbbb bbb bbb
bb bb bbbb b
thales$ ./dw.pl a.txt b.txt
aaa aa aa aaaa
bbbb bbb bbb
bb bb bbbb b
thales$
```

Lösung:

```
#!/usr/local/bin/perl

use strict;

while (<>) {
   print if /\b(\w+)\s+\1\b/;
}
```

Aufgabe 3 (16 Punkte)

Gegeben Sei eine kleine Datenbank, die für eine Reihe von Benutzern und Rechnern den jeweils letzten Login-Vorgang festhält:

Nach Überführung in die dritte Normalform kommen Sie dabei zu folgenden Tabellen (die Feldnamen, die jeweils zum Primärschlüssel gehören, sind fett angegeben):

- *User*: **login**, name
- *Host*: **hostname**, model
- *Login*: **login**, **hostname**, lastlogin

Wählen Sie nun die Art der Datenhaltung oder Abstraktion frei aus (entweder ASCII-Dateien, DBM-Dateien, per DBI zugängliche SQL-Datenbanken oder TBI) und schreiben Sie dafür folgende kleine Anwendung: Gegeben sei ein Rechnername und ein Datum (über die Kommandozeile). Auszugeben sind dann alle Benutzernamen (Komponente *name* bei *User*), die sich an dem fraglichen Tag auf dem Rechner zuletzt angemeldet haben.

Gehen Sie davon aus, daß *lastlogin* wie bei SQL beim Datentyp **datetime** repräsentiert wird in der Form "YYYY-MM-DD HH:MM:SS" und das Datum auf der Kommandozeile in der Form "YYYY-MM-DD" übergeben worden ist.

Lösung:

```
#!/usr/local/bin/perl -w

use TBI;
use strict;

my $cmdname = $0; $cmdname =~ s{.*/}{};
my $usage = "Usage: $cmdname host date\n";
die $usage unless @ARGV == 2;
my $host = shift; my $date = shift;

my $user = TBI->open("User");
my $login = TBI->open("Login");

foreach my $key ($login->select(host => $host)) {
   my %record = $login->fetch($key);
   next unless $record{lastlogin} =~ /^$date\s/o;
   my %user = $user->fetch($record{login});
   print $user{name}, "\n";
}
```

Aufgabe 4 (10 Punkte)

Gegeben sei eine Datei logins, die folgenden Aufbau hat:

login:Name:Vorname:PLZ:Ort:Strasse:Tel:Studiengang:Sonstiges

Schreiben Sie nun ein Perl-Skript, das wie folgt aufgerufen wird:

```
getuser value
```

Das Skript sollte alle Logins ausgeben, die mindestens ein Feld (außer login) haben, das mit `value` anfängt.

Lösung:

```
#!/usr/local/bin/perl -w

use strict;

my ($value) = ($ARGV[0]);
open(DB, "logins") or die "Cannot open logins";
while (<DB>) {
   print "$1 " if /^([^:]+).*:\Q$value\E/o;
}
close DB;
print "\n";
```

Aufgabe 5 (12 Punkte)

Kreuzen Sie bei den Teilaufgaben a) und b) jeweils die richtige Antwort an. Begründungen sind nicht notwendig.

(*a*) In Perl werden sehr gerne Variablen mit **my** deklariert. Welche Funktionalität hat **my**?

lokale Variable (wie bei C oder Modula-2)	
dynamisch allokierte Variable (wie malloc in C oder NEW in Modula-2)	X
statische Variable, die nur innerhalb einem Block sichtbar ist (analog zur Verwendung von **static** innerhalb von Funktionen in C oder innerhalb von Modulen deklarierte Variablen in Modula-2)	

(*b*) Wie lange währt die Lebenszeit einer mit **my** deklarierten Variable?

bis zum Ende des zugehörigen Blocks	
mindestens solange, bis keine Zugriffsmöglichkeit mehr auf die Variable besteht	X
bis sie explizit freigegeben wird	
bis zum Ende des Programms	

(*c*) Was gibt folgendes Programm aus?

```
#!/usr/local/bin/perl -w

sub f {
   my ($counter) = @_;
   my $a = sub { return ++ $counter };
   my $b = sub { return -- $counter };
   return ($a, $b);
}
my ($a, $b) = f(2);
my ($c, $d) = f(3);
print join(" ", &{$a}, &{$b}, &{$c}, &{$d}, &{$d}, &{$a}), "\n";
```

Ausgabe:

```
3 2 4 3 2 3
```

Aufgabe 6 (18 Punkte)

Unsere Passwd hat folgenden Aufbau:

login:pw:uid:gid:name:home:shell

Vervollständigen Sie folgendes Perl Skript, das den Inhalt der Passwd in eine My-SQL-Datenbank einfügt. Die Tabelle wurde mit folgender Anweisung erzeugt:

```
CREATE TABLE Passwd (
   login VARCHAR(8) NOT NULL PRIMARY KEY, INDEX(login),
   pw VARCHAR(14) NOT NULL,
   uid INTEGER NOT NULL,
   gid INTEGER NOT NULL,
   name VARCHAR(40) NOT NULL,
   home VARCHAR(80) NOT NULL,
   shell VARCHAR(40) NOT NULL
);
```

```
#!/usr/local/bin/perl -w

use DBI;
use Mysql::Admin qw(dbi_connect);
use strict;

my $dbdir = '/.../'; # Verzeichnis zur Datenbank

my $db = DBI->connect(dbi_connect($dbdir), "", "");
die "unable to connect to db: $DBI::errstr\n" unless defined $db;
$db->{RaiseError} = 1;
```

Lösung:

```
open(FH, "ypcat passwd |") or die "Cannot open passwd";
my $st = $db->prepare("insert into Passwd values (".join(", ", ("?") x 7).")");
my $record;
while (<FH>) {
   chomp;
   $st->execute(split /:/);
}
$st->finish;
$db->disconnect;
close FH;
```

Aufgabe 7 (14 Punkte)

Implementieren Sie ein vollständiges Perl-Modul, das eine Klasse für die Verwaltung von Informationen zu Freunden realisiert. Zu implementieren ist ein Konstruktor (ohne Parameter) und die beiden Operationen **addinfo** (mit dem Namen eines Freundes und eines Textes, der Informationen über den Freund enthält) und **getinfo**, das die bislang angesammelten Informationen in Form einer Liste zurückliefert. So könnte ein Beispiel in der Verwendung dieses Moduls aussehen:

```
my $friends = new Friends;
$friends->addinfo("Bob", "Birthday: May, 13th");
$friends->addinfo("Mary", "Favourite Colour: Blue");
$friends->addinfo("Bob", "Favourite Dish: Lasagne");
my @infos = $friends->getinfo("Bob");
# @infos = ("Birthday: May, 13th", "Favourite Dish: Lasagne");
my $other = new Friends;
my @other_infos = $other->getinfo("Bob");
# @other_infos = ();
```

In Ihrer Implementierung reicht es aus, wenn Sie normale Datenstrukturen verwenden, die dem jeweiligen Objekt zugeordnet sind. Sie brauchen diese Daten also nicht in einer Datei abzulegen.

Lösung:

```
package Friends;

use strict;
use vars qw(@ISA);
require Exporter;
@ISA = qw(Exporter);

sub new {
   my ($package) = @_;
   my $self = bless {}, $package;
}

sub addinfo {
   my ($self, $friend, $info) = @_;
   push(@{$self->{$friend}}, $info);
}

sub getinfo {
   my ($self, $friend) = @_;
   if (defined $self->{$friend}) {
      return @{$self->{$friend}};
   } else {
      return ();
   }
}

1;
```