Making a Start with the Perl Net::LDAP Assignment

1 What to Do

1. Search for your own entry under `ou=sys,o=ICT` with:

   ```
   $ ldapsearch -x -h ldap1.tyict.vtc.edu.hk -b 'ou=sys,o=ICT' \
   '(uid=⟨your student ID⟩)'
   ```

2. Search for your own `ou` under `o=ICT` with:

   ```
   $ ldapsearch -x -h ldap1.tyict.vtc.edu.hk -b 'o=ICT' '(ou=⟨your student ID⟩)'
   ```

3. Search for the `People` and `Group` `ou`s on `ldap1.tyict.vtc.edu.hk` with:

   ```
   $ ldapsearch -x -s one -h ldap1 -b 'ou=sys,o=ICT' '(ou=*)'
   ```

   so that you can look at them and understand what you need to do for the next exercise.

   - Note: the Windows computers in the labs have `ldapsearch` also. With this, omit `-x`. Omit the quotes. This program has no default filter, so to search for all entries in my `ou`, you could make a search like this:

     ```
     C:\> ldapsearch -s one -h ldap1 -b ou=nicku,o=ICT (objectClass=*)
     ```

4. Install the `Term::ReadKey` Perl module like this:

   ```
   $ sudo perl -MCPAN -e shell

   cpan shell -- CPAN exploration and modules installation (v1.7601)
   ReadLine support enabled

   cpan> install Term::ReadKey
   ```

5. Write a program using `Net::LDAP` that does the following:

   (a) Creates a new `Net::LDAP` object by connecting to the LDAP server `ldap1.tyict.vtc.edu.hk`

   (b) Use my code at `http://nicku.org/snm/assignments/assignment-perl-ldap/programs/read_password.pl` to get your password.

   (c) Binds as your DN: `uid=`⟨*your student ID*⟩`,ou=People,ou=sys,o=ICT`. Your password will be the same as it was on `nicku.org` last week.

   (d) Creates an entry `ou=People,ou=`⟨*your student ID*⟩`,o=ICT`

   (e) Creates an entry `ou=Group,ou=`⟨*your student ID*⟩`,o=ICT`

   (f) Update these on the server.

   (g) Add support for `start_tls` if you are not using Windows.

2 How to Pass Parameters? How to...

Some people have asked me, "Nick, how can I pass parameters to this subroutine?", where they have shown me a nice subroutine that wants parameters passed to it. Well, here is a simple example; a subroutine that takes a `NET::LDAP::Entry` object and adds it to a `Net::LDAP` object:

```
sub add_entry($$) {
    my ( $ldap, $entry ) = @_;
    my $mesg = $entry->update( $ldap );
    die "cannot add entry: ", $mesg->error if $mesg->code;
}
```

However, suppose the entry is already in the directory? Do we want our program to die in a screaming heap just because it is already in the directory? No? Well, we can see in `Net::LDAP::Constant` that there is a constant `LDAP_ALREADY_EXISTS`. We can test `$mesg->code` to see if it has that value. As

```
$ perldoc Net::LDAP::Constant
```

tells us, we must import that symbol before we can use it. So we can write this instead if you like:

```
use Net::LDAP qw( LDAP_ALREADY_EXISTS );
sub add_entry($$) {
    my ( $ldap, $entry ) = @_;
    my $mesg = $entry->update( $ldap );
    if ( $mesg->code == LDAP_ALREADY_EXISTS ) {
        warn "The entry ", $entry->dn, " already is in the directory\n";
        return;
    } elsif ( $mesg->code ) {
        die "cannot add entry: ", $mesg->error;
    }
    return 1;
}
```

Note that this function can return and still fail, so we should return a false value if we failed, and a true value if we were successful. The idiom for returning a false value is to just put an empty `return` statement. This works whether we are expecting an array or a scalar value. The value will be `undef` in a scalar context, or an empty list in a list context.

2.1 Function Prototypes

What is the "`($$)`" business in the two function definitions above? In Perl, these are called *function prototypes*, although they are different from function prototypes in C or C++. If a function prototype appears before you call the function, then (in this case), there will be a compile-time error if the function is called with anything other than two scalar parameters. It also means that you can omit the parentheses after the function name in the function call, since Perl knows exactly how many parameters to expect. There are some other benefits to using function prototypes in Perl. Read

```
$ perldoc perlsub
```

for more about them.