Programming Proverbs

- 16. “Build in debugging techniques.”

- Henry F. Ledgard, “Programming Proverbs: Principles of Good Programming with Numerous Examples to Improve Programming Style and Proficiency”, (Hayden Computer Programming Series), Hayden Book Company, 1st edition, ISBN-13: 978-0810455221, December 1975.

Creating shared libraries under GNU/Linux

- focus on the major advantage
- interfacing C, C++, Modula-2 with scripting languages
  - Python, Perl, Ruby, TCL
  - further focus examples around Python

Creating shared libraries under GNU/Linux

- Python's modules are either written in Python or are implemented as a shared library
  - or a combination of both

- we will briefly examine the following tools
  - `gcc`, `g++`, `libtool`, `swig`, `make` and `gm2`

Simple pedagogical example

- let us create a module to sum two integers, we will use `swig` to call C functions from Python

- **mymodule.i**

```
%module mymodule
%{
extern int sum (int a, int b);
%}
extern int sum (int a, int b);
```

Simple pedagogical example

mymodule.c

```
int sum (int a, int b)
{
  return a + b;
}
```

Simple pedagogical example

-
```
$ swig -python mymodule.i
```

- generates the following files:
  - `mymodule_wrap.c` and `mymodule.py`

Use gcc and libtool to compile and link the shared library

```
$ libtool --tag=CC --mode=compile gcc -g -I/usr/include/python2.7 \
  -c mymodule_wrap.c -o mymodule_wrap.lo
$ libtool --tag=CC --mode=compile gcc -g -I/usr/include/python2.7 \
  -c mymodule.c -o mymodule.lo
$ libtool --tag=CC --mode=link gcc -g mymodule.lo mymodule_wrap.lo \
  -rpath `pwd` -lc -lm -o libmymodule.la
$ cp .libs/libmymodule.so _mymodule.so
```

Use gcc and libtool to compile and link the shared library

testsum.py

```
#!/usr/bin/python

import mymodule

print mymodule.sum (1, 2)
```

Use gcc and libtool to compile and link the shared library

```
$ python testsum.py
3
```

libtool on GNU/Linux

- notice the file extensions `.lo` and `.la`
- `libtool` is told about the library dependents and where other shared libraries reside

Output from running previous libtool command

```
libtool: compile:  gcc -g -I/usr/include/python2.7 -c mymodule_wrap.c  -fPIC -DPIC -o .libs/mymodule_wrap.o
libtool: compile:  gcc -g -I/usr/include/python2.7 -c mymodule_wrap.c -o mymodule_wrap.o >/dev/null 2>&1
libtool: compile:  gcc -g -I/usr/include/python2.7 -c mymodule.c  -fPIC -DPIC -o .libs/mymodule.o
libtool: compile:  gcc -g -I/usr/include/python2.7 -c mymodule.c -o mymodule.o >/dev/null 2>&1
libtool: link: rm -fr  .libs/libmymodule.a .libs/libmymodule.la .libs/libmymodule.lai \
  .libs/libmymodule.so .libs/libmymodule.so.0 .libs/libmymodule.so.0.0.0
libtool: link: gcc -shared  -fPIC -DPIC  .libs/mymodule.o .libs/mymodule_wrap.o \
   -L`pwd`/lib64 -lc -lm    -Wl,-soname -Wl,libmymodule.so.0 -o .libs/libmymodule.so.0.0.0
libtool: link: (cd ".libs" && rm -f "libmymodule.so.0" && ln -s "libmymodule.so.0.0.0" "libmymodule.so.0")
libtool: link: (cd ".libs" && rm -f "libmymodule.so" && ln -s "libmymodule.so.0.0.0" "libmymodule.so")
libtool: link: ar cru .libs/libmymodule.a  mymodule.o mymodule_wrap.o
libtool: link: ranlib .libs/libmymodule.a
libtool: link: ( cd ".libs" && rm -f "libmymodule.la" && ln -s "../libmymodule.la" "libmymodule.la" )
```

- note that this is the output from the slide containing the three `libtool` commands

- `libtool` is a highly portable mechanism to compile and link shared libraries
  - the output from the `libtool` commands will be different under OSX and Windows and or different versions of `gcc` and `g++`