B4 - Object-Oriented Programming

B-OOP-400

Unix Libraries

Arcade Bootstrap

2.0

Back to ld.so

+ Exercise 1

Read `dlopen(3)`, `dlsym(3)`, `dlclose(3)` and `dlerror(3)`.

+ Exercise 2

Now that you know a bit about how dynamic linking works, let's lay out the basic concepts for loading dynamic libraries.

When a dynamic library is loaded and unloaded, it is possible to call a user-defined function to initialize and finalize the library. To do so, **gcc** provides the following attributes:

- `__attribute__((constructor))`
- `__attribute__((destructor))`

The function carrying the `constructor` attribute is called by `dlopen` before it returns. The function carrying the `destructor` attribute is similarly called by `dlclose`.

- Create any number of shared libraries using **gcc**
- Each of these must contain 3 functions:
  - A procedure carrying the `constructor` attribute, taking no parameter
  - A procedure carrying the `destructor` attribute, taking no parameter
  - A function with any prototype you like, called `entryPoint`
- Each of these functions must print an appropriate message regarding its role to the standard output
- You must use a loop to load each library, call the `entryPoint` functions, and close each library
- Your main program must not explicitly print out anything!

Unlike functions, procedures don't return values.

Here's a sample output:

```
[libfoo] Loading foo library...
[libbar] Loading bar library...
[libgra] Loading gra library...
[libfoo] Entry point for foo!
[libbar] Entry point for bar!
[libgra] Another entry point!
[libfoo] foo closing...
[libbar] Closing bar...
[libgra] Gra's getting out...
```

To use `dlopen(3)`, `dlsym(3)` and `dlclose(3)` you must link your program to the `libdl` library, with the `-ldl` flag.

SWITCHING TO C++

+ REMINDER

As you now know, C++ has a particular compilation mechanism that allows for **parameter polymorphism**. This mechanism is called ***mangling***. The basic idea is to encode function names with the type of their parameters (and any namespace they might belong to).

Unfortunately, this results in symbols with obscure names...

As you should have understood by now, the string passed to `dlsym(3)` must match the exact function name. But how can you do that when the function name is modified by mangling?

C++ provides a solution to this problem: `extern "C"`.

```
extern "C"
{
    int entryPoint()
    {
    }
}

// Or...

extern "C" int entryPoint()
{
}
```

+ EXERCISE 3

Modify your dynamic libraries from the previous exercise so that you can compile them using **g++**.

+ With objects

The easiest technique to create a C++ library is to offer an entry point which is simply used as an object creator.

To do so, your program must have ann interface declaration, implemented by a class in the library. The entry point can then return an instance of that class, that the main program can use through the interface.

You may want to read that part one more time. And another.

For the next exercise, we'll use the following interface as an example:

```
class IDisplayModule {
public:
    virtual ~IDisplayModule() = default;

    virtual void init() = 0;
    virtual void stop() = 0;
    virtual const std::string &getName() const = 0;
};
```

+ Exercise 4

- Modify your dynamic libraries so that they each define a class that implements `IDisplayModule`
- Their `init()` and `stop()` functions must each print different messages to the standard output
- Each of your libraries must have a generic entry point that returns a polymorphic instance of the contained class implementing the interface

+ Library loader

The functions offered by **libdl** are those of a C API. As you are writing C++ code, it would be nice to create abstractions that let you use the functions from a higher level.

The solution offered here is to encapsulate the functions within an object, responsible for loading libraries and getting object instances from them.

+ Exercise 5

Complete the following `DLLoader` class:

```
template<typename T>
class DLLoader {
public:
    [...]
    T *getInstance([...]) [...];
    [...]
};
```

- The `getInstance` function must return an instance of the class implemented by the library.
- `DLLoader` will be instantiated once per library to load

You can add as many functions/attributes as you wish, but keep in mind that a simple object is a reliable object!