Lab 8

Module, Makefile

DCSLAB Haeram Kim / 2024-05-08

Table of contents

Modulize

Why modulize?

Audience | Developers | Topic | Linux | News

Linux in 2020: 27.8 million lines of code in the kernel, 1.3 million in systemd

By **Swapnil Bhartiya** - January 7, 2020

38003

#include “something.cc”

```
#include <iostream>

using namespace std;

int foo() {
  cout << "Foo!" << endl;
  return 0;
}

int bar() {
  cout << "Bar!" << endl;
  return 0;
}

int main() {
  foo();
  bar();
}
```

```
#include <iostream>

using namespace std;

int foo() {
  cout << "Foo!" << endl;
  return 0;
}
```

```
#include <iostream>

using namespace std;

int bar() {
  cout << "Bar!" << endl;
  return 0;
}
```

```
#include "bar.cc"
#include "foo.cc"

int main() {
  foo();
  bar();
}
```

Header file

- Separate declaration and definition
  - Avoid duplicated definition
    - Slow compile time
  - Encapsulation and abstraction
  - Code reusability

- Header file (.h, .hpp): Declaration
- Source file (.cc, .cpp): Definition (Implementation)

Header file (Cont'd)

```
// foo.cc
#include <iostream>
#include "foo.h"

using namespace std;

int foo() {
  cout << "Foo!" << endl;
  return 0;
}
```

```
// foo.h
int foo();
```

```
// main.cc
#include "bar.h"
#include "foo.h"

int main() {
  foo();
  bar();
}
```

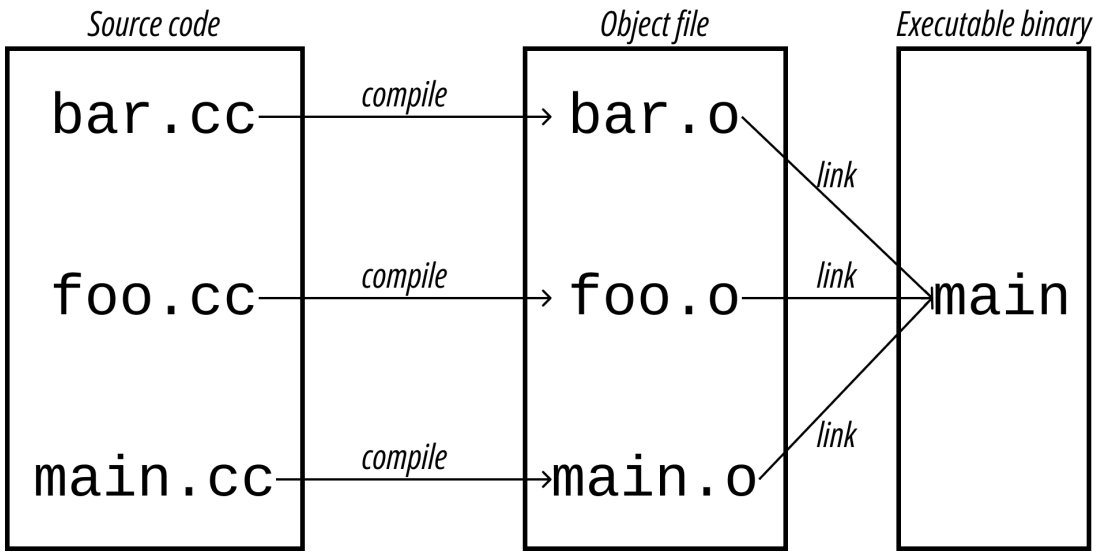
```
// bar.h
int bar();
```

```
// bar.cc
#include <iostream>
#include "bar.h"

using namespace std;

int bar() {
  cout << "Bar!" << endl;
  return 0;
}
```

Link vs include

```
> g++ main.cc
Undefined symbols for architecture arm64:
  "bar()", referenced from:
      _main in main-92d28a.o
  "foo()", referenced from:
      _main in main-92d28a.o
ld: symbol(s) not found for architecture arm64
clang: error: linker command failed with exit code 1 (use -v to see invocation)
```

Object file: compile w/o link




```
g++ -c bar.cc
g++ -c foo.cc
g++ -c main.cc
g++ bar.o foo.o main.o -o main
```

Makefile

Why Makefile?

- Avoid typing `g++ …` every time
- Faster then script (e.g. shell script)
  - …compile only changes

Basic syntax

```
TARGET: PREREQUISITES
    RECIPE
```

```
main: foo.o bar.o main.o
    g++ foo.o bar.o main.o -o main

foo.o: foo.h foo.cc
    g++ -c foo.cc

bar.o: bar.h bar.cc
    g++ -c bar.cc

main.o: main.cc foo.h bar.h
    g++ -c main.cc
```

Build w/ Makefile

```
> make
g++ -c foo.cc
g++ -c bar.cc
g++ -c main.cc
g++ foo.o bar.o main.o -o main
```

```
> ./main
Foo!
Bar!
```

Variables

- `=` to declare a variable
- `$()` to use a variable

CC, CXXFLAGS, OBJS

```
CC = g++
CXXFLAGS = -Wall -O2
OBJS = foo.o bar.o main.o

main : $(OBJS)
    $(CC) $(CXXFLAGS) $(OBJS) -o main

foo.o : foo.h foo.cc
    $(CC) $(CXXFLAGS) -c foo.cc

bar.o : bar.h bar.cc
    $(CC) $(CXXFLAGS) -c bar.cc

main.o : main.cc foo.h bar.h
    $(CC) $(CXXFLAGS) -c main.cc
```

Q&A