Compilation

Victor Eijkhout, Susan Lindsey

Fall 2024
last formatted: October 24, 2024

Intro to file types

1. **File types**

Text files	
Source	Program text that you write
Header	also written by you, but not really program text.

Binary files	
Object file	The compiled result of a single source file
Library	Multiple object files bundled together
Executable	Binary file that can be invoked as a command
Data files	Written and read by a program

2. Text files

- Source files and headers
- You write them: *make sure you master an editor*
- The computer has no idea what these mean.
- They get compiled into programs.

(Also 'just text' files: READMEs and such)

3. Binary files

- Programs. (Also: object and library files.)
- Produced by a compiler.
- Unreadable by you; executable by the computer.

Also binary data files; usually specific to a program.
(Why don't programs write out their data in readable form?)

Compilation

4. Compilers

Compilers: a major CS success story.

- The first Fortran compiler (Backus, IBM, 1954):
  multiple man-years.
- These days: semester project for graduate students.
  Many tools available (`lex`, `yacc`, `clang-tidy`)
  Standard textbooks ('Dragon book')
- Compilers are very clever!
  You can be a little more clever in assembly – maybe
  but compiled languages are $10\times$ more productive.

5. Compilation vs interpreted

- Interpreted languages: lines of code are compiled 'just-in-time'. Very flexible, sometimes very slow.
- Compiled languages: code is compiled to machine language: less flexible, very fast execution.
- Virtual machine: languages get compiled to an intermediate language
  (Pascal, Python, Java)
  pro: portable; con: does not play nice with other languages.
- Scientific computing languages:
  - Fortran: pretty elegant, great at array manipulation
    Note: Fortran20003 is modern; F77 and F90 are not so great.
  - C: low level, allows great control, tricky to use
  - C++: allows much control, more protection, more tools
    (kinda sucks at arrays)

6. Simple compilation

hello.c

```
int main() {
  printf("Hello world\n");
  return 0;
}
```

icc -o hello.exe hello.c →

hello.exe

- From source straight to program.
- Use this only for short programs.

```
%% icpc hello.c              %% icpc -o helloprog hello.c
%% ./a.out                   %% ./helloprog
hello world                  hello world
```
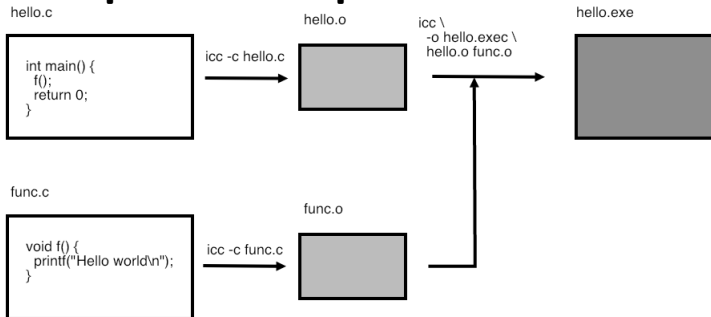
7. Exercise 1

Create a file with these contents, and make sure you can compile it:

```cpp
#include <iostream>
using std::cout;

int main() {
  cout << "hello world\n";
  return 0;
}
```

8. Separate compilation

- Large programs best broken into small files,
- ... and compiled separately (can you guess why?)
- Then 'linked' into a program; linker is usually the same as the compiler.

9. Exercise 2

Make the following files:

Main program: `fooprog.cpp`

```cpp
#include <iostream>
using std::cout;
#include <string>
using std::string;

extern void bar(string);

int main() {
  bar("hello world\n");
  return 0;
}
```

Subprogram: `foosub.cpp`

```cpp
#include <iostream>
using std::cout;
#include <string>
using std::string;

void bar( string s ) {
  cout << s << '\n';
}
```

10. **Exercise 2 continued**

- Compile in one:
  ```
  icpc -o program fooprog.c foosub.c
  ```
- Compile in steps:
  ```
  icpc -c fooprog.c
  icpc -c foosub.c
  icpc -o program fooprog.o foosub.o
  ```
  What files are being produced each time?

Can you write a shell script to automate this?

11. **Header files**

- extern is not the best way of dealing with 'external references'

- Instead, make a header file foo.h that only contains

    ```
    void bar(string);
    ```

- Include it in both source files:

    ```
    #include "foo.h"
    ```

- Do the separate compilation calls again.

Now is a good time to learn about makefiles …

12. Compiler options 101

- You have just seen two compiler options.
- Commandlines look like
  command [options] [argument]
  where square brackets mean: 'optional'
- Some options have an argument
  icpc -o myprogram mysource.c
- Some options do not.
  icpc -g -o myprogram mysource.c
- Question: does -c have an argument? How can you find out?
  icpc -g -c mysource.c