SCSJ 2733
Introduction to Fortran

Mohsin Mohd Sies

Adapted from notes by
Kadin Tseng
Boston University

Outline

- Objectives
- Introduction
- Fortran History
- Compiling

Objectives

- write simple Fortran programs
- understand and modify existing Fortran code

Introduction

Two fundamentally different types of high-level languages:

- *Interpreted* language
  - MATLAB, Python, Java
  - Translation to machine-language is performed incrementally at run time
- *Compiled* language
  - Fortran, C, C++
  - Translation is performed once, then executable is run as frequently as needed without further translation

Introduction (cont'd)

- Compiled languages run faster.
  - Large-scale computing is usually done with compiled language
- Interpreted languages more convenient but slower
  - e.g., no need to declare variables; do things on-the-fly
  - MATLAB can be an order of magnitude slower than C/fortran (code dependent)

Fortran History

- Before Fortran, programs were written in assembly language (very tedious to say the least)
  - low-level commands such as “load x from memory into register 7” or “add values in registers 10 and 11 and write result to register 4”
- Fortran was the first widely-used high-level computer language
  - 1957
  - Developed by IBM for scientific applications
  - Program written on a specially formatted green sheet, then entered as punched cards

Fortran History

- Fortran 66 (1966)
- Fortran 77 (1978)
- Fortran 90 (1991)
  - "fairly" modern (structures, etc.)
  - Current "workhorse" Fortran
- Fortran 95 (minor tweaks to Fortran 90)
- Fortran 2003
  - Gradually being implemented by compiler companies
  - Object-oriented support
  - Interoperability with C is in the standard

What Language Should I Use?

- Generally, use the language you know best
- Interpreted languages are great for
  - Interactive applications
  - Code development and debugging
  - Algorithm development
- For major number crunching, compiled languages are preferred (Fortran, C, C++)

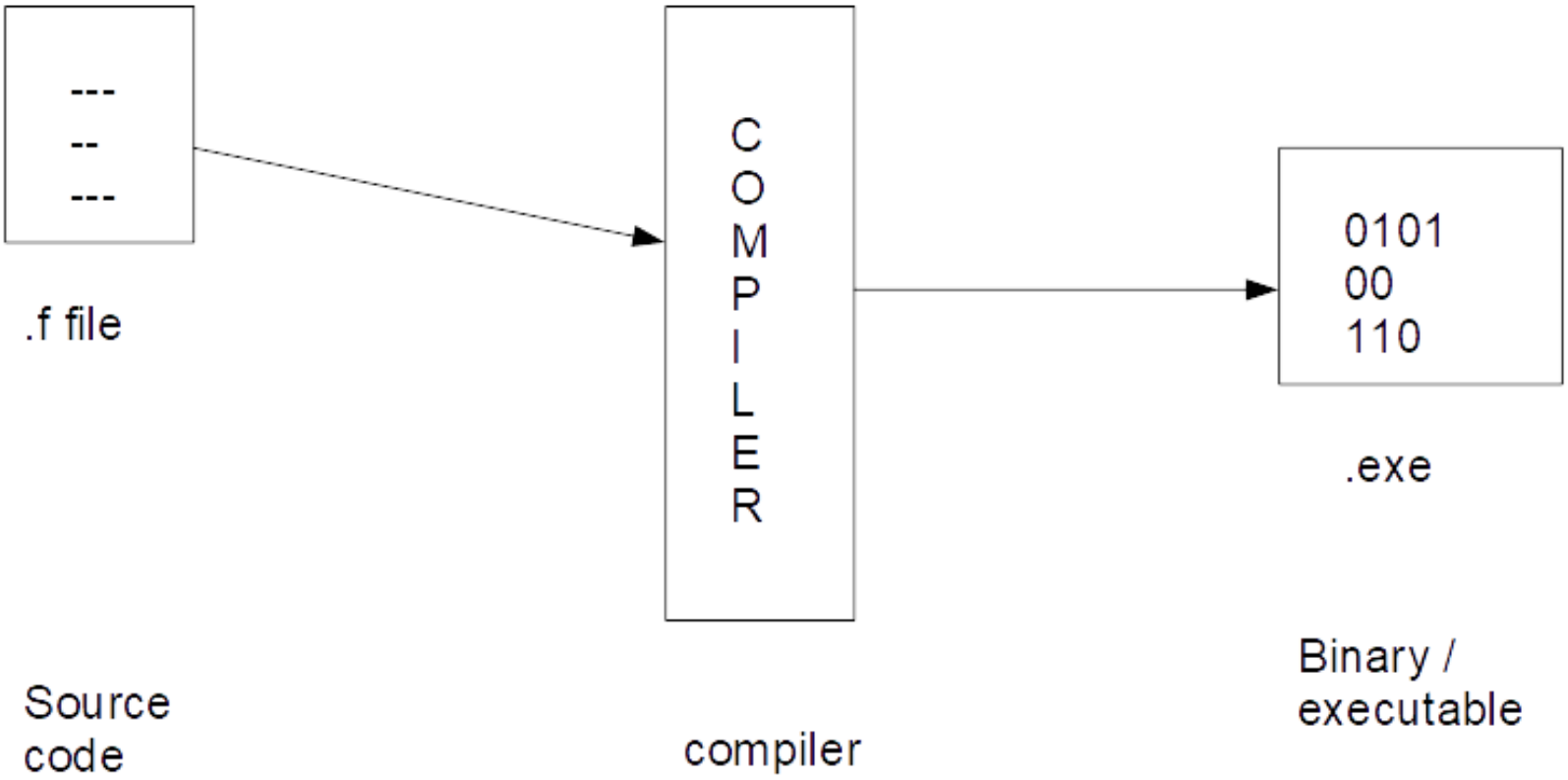
Coding

- Program is contained in a text file
  - called *source code* or *source file*
- Source code must be processed by a *compiler* to create an executable
- Source file suffix can be (.for, .f, .F, .f90, .F90, …)
- Since source file is simply text, it can be written using any text editor
  - Notepad++ is recommended for this course
  - Use a suitable fixed-width font such as `Courier`, `Consolas`, `Inconsolata`

Compilation

- A compiler is a program that reads source code and converts it to a form usable by the computer
- Internally, three steps are performed:
  - **preprocess** source code
  - **check** source code for syntax errors
  - **compiler** translates source code to assembly language
  - **assembler** translates assembly language to machine language
  - **linker** gathers machine-language modules and libraries
  - All these steps sometimes loosely referred to as “compiling”

Compiling



--

.f file
C
O
M
P
I
L
E
R
0101
00
110
.exe
Source
code
compiler
Binary /
executable

Compilation (cont'd)

- Code compiled for a given processor architecture will not generally run on other processors
  - AMD and Intel *are* compatible
- Code compiled on an operating system (e.g. Windows) will also not run on other operating systems (e.g. Linux, Mac)

Compilation (3)

- Compile hello.f on the terminal console :

```
Cygwin~$  gfortran -o hello hello.f
```

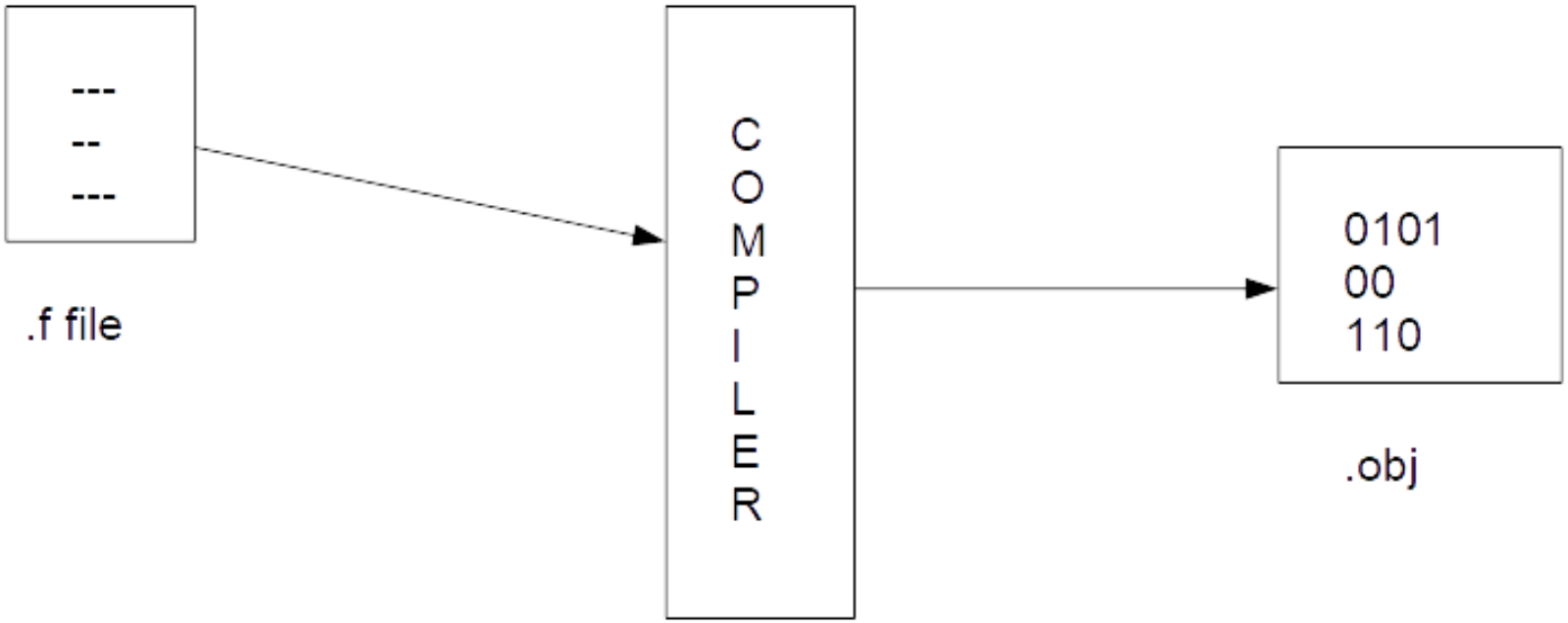
(-o lets you set executable name, `hello` is the executable file name)

- If it simply returns a Unix prompt, it worked
- If you get error messages, read them carefully and see if you can fix the source code and re-compile
- Once it compiles correctly, type the executable name at the Unix prompt, and it will print your string

```
Cygwin~$  ./hello
```

Advanced Compiling

Creating Libraries



Advanced Compiling