Separate Compilation

Managing a Program With Multiple Source Files

CSCI 3700 — Data Structures and Objects

Department of Computer Science and Information Systems
Youngstown State University

Robert W. Kramer

Outline

Goal

Create a program that

- Reads a list of up to 100000 numbers from *stdin*;
- Sorts the list;
- Outputs the list to *stdout*.

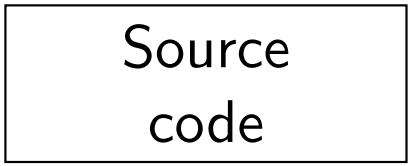
As a bonus, each function resides in a separate source file.

The Compilation Process

What happens when you run g++ (1/6)

When you run the g++ program...
g++ farms the work out to four separate utilities.

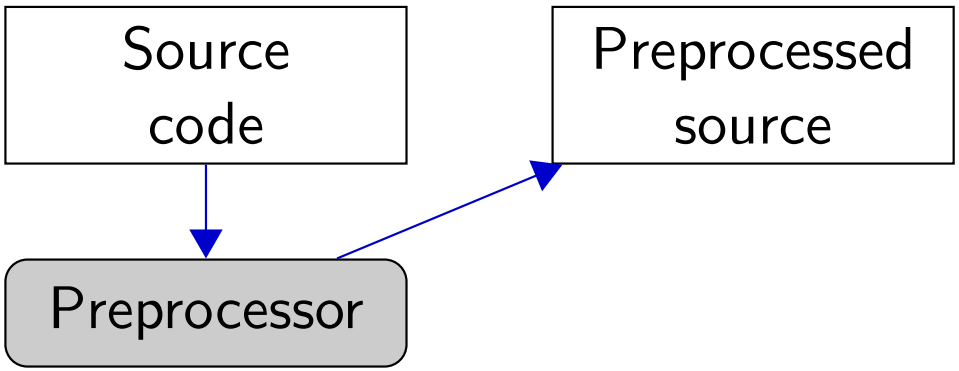
Starting with a basic source file...



The Compilation Process

What happens when you run g++ (2/6)

The source is fed to a *preprocessor*...

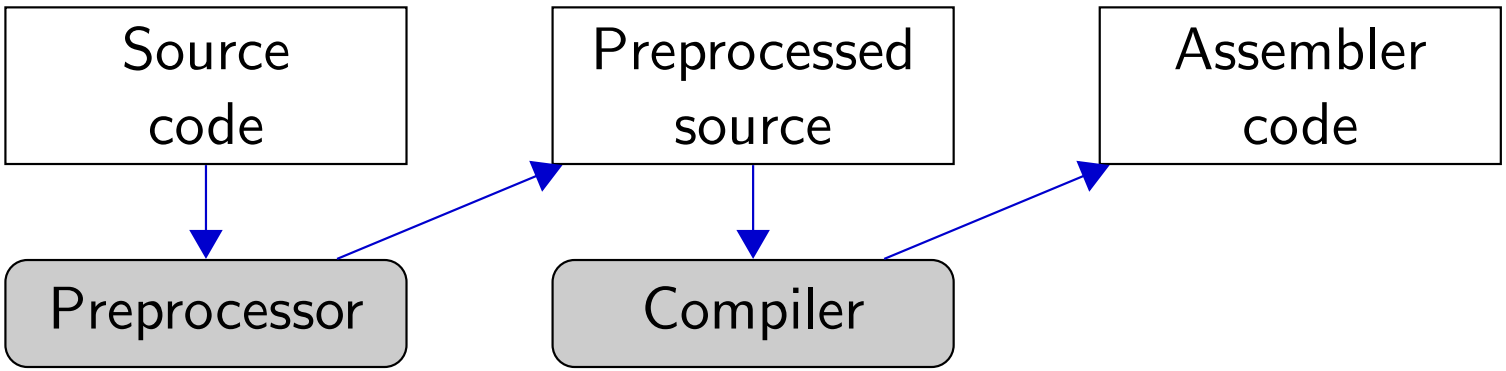


- The preprocessor performs text replacement (aka *macro expansion*)
- It handles all of the # lines
- The result is expanded source code
- Uses external *header files*

The Compilation Process

What happens when you run g++ (3/6)

The preprocessed source is fed to the *compiler*...

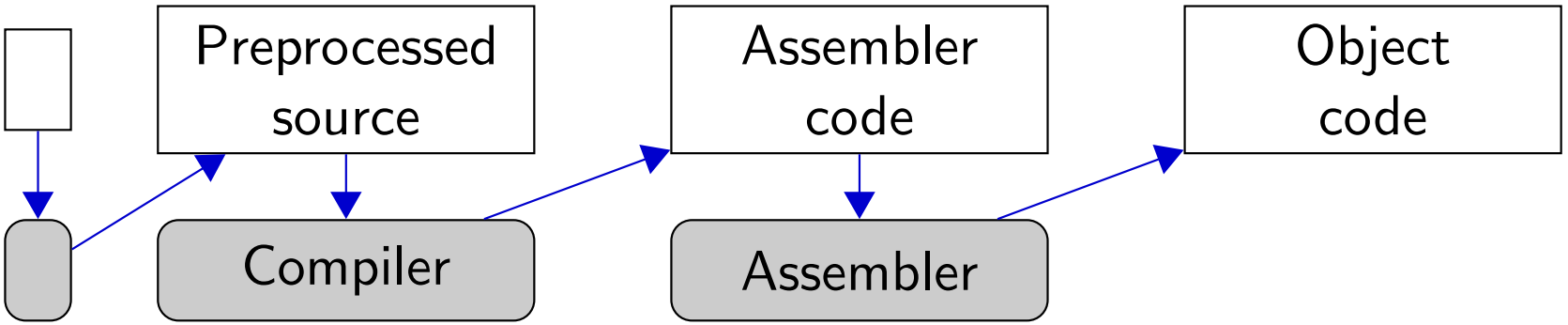


- The compiler translates your program into *assembly language*
  - Human readable
  - One step removed from machine language

The Compilation Process

What happens when you run g++ (4/6)

The assembler source is fed to the *assembler*...



- The assembler translates your program into *object code*

The Compilation Process

What happens when you run g++ (5/6)

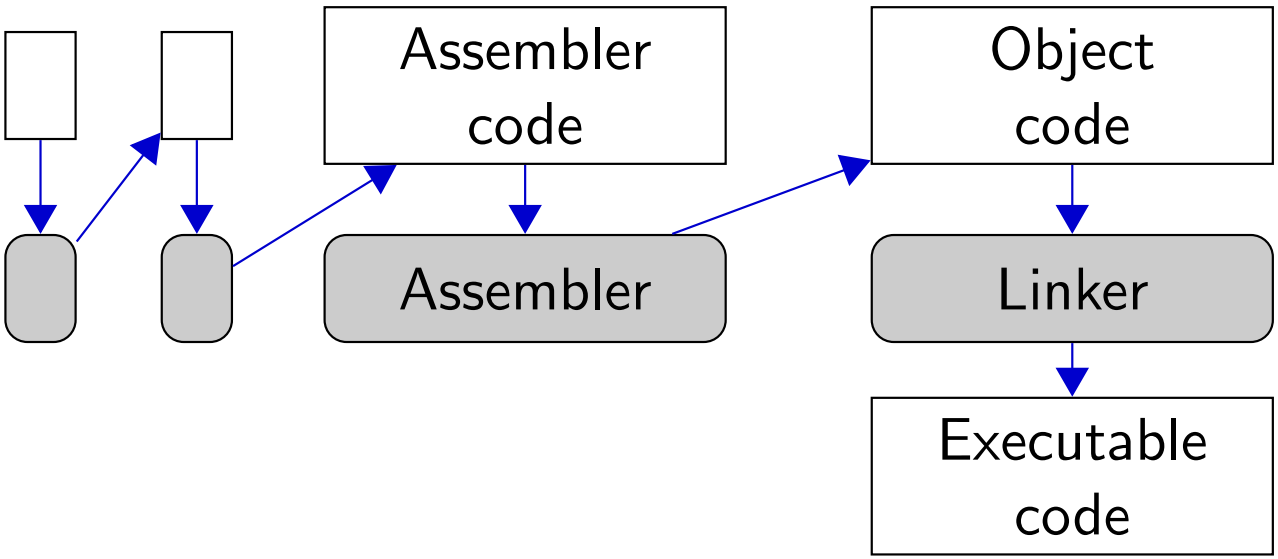
Object code consists of three parts:

- Machine language
- Holes for missing functions / globals
- Fillers for holes in other object code files

The Compilation Process

What happens when you run g++ (6/6)

The object code is fed to the *linkage editor*...



- The linker collects object files, patches holes and produces *executable code* (the `a.out` file)
- May include code / data from *libraries*

So What?

Why do we care what goes on behind the scenes?

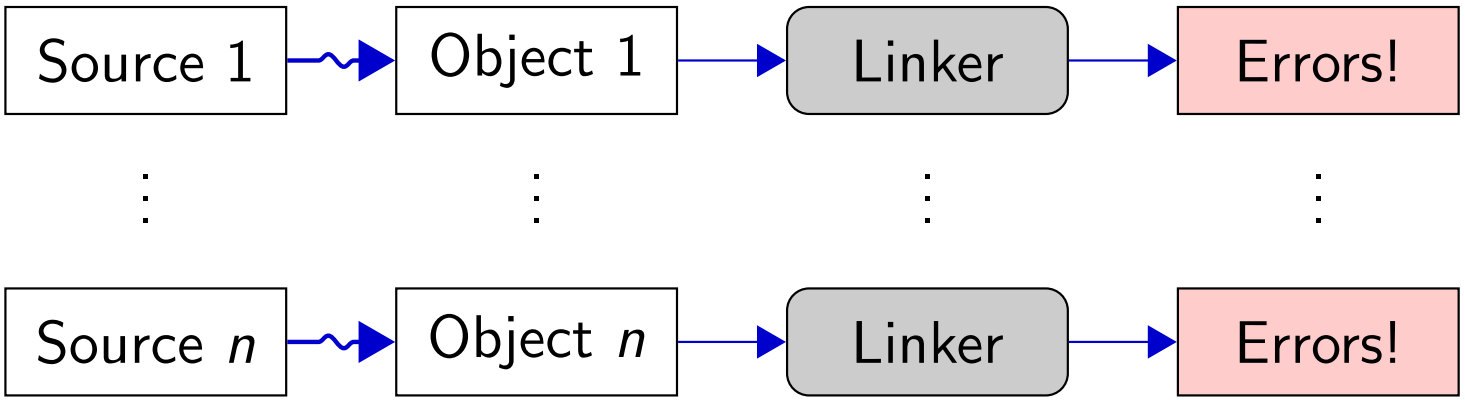
Two important observations:

1. The preprocessor, compiler and assembler do not care if the code / data your program references exist in your source code
   - These show up as holes in your object code
   - Patching holes is only important to the linker
2. In this process, the compiler uses the majority of the CPU time

Splitting the Source Code

Taking advantage of the compilation process (1/2)

Suppose we split the program into multiple source files and compile them

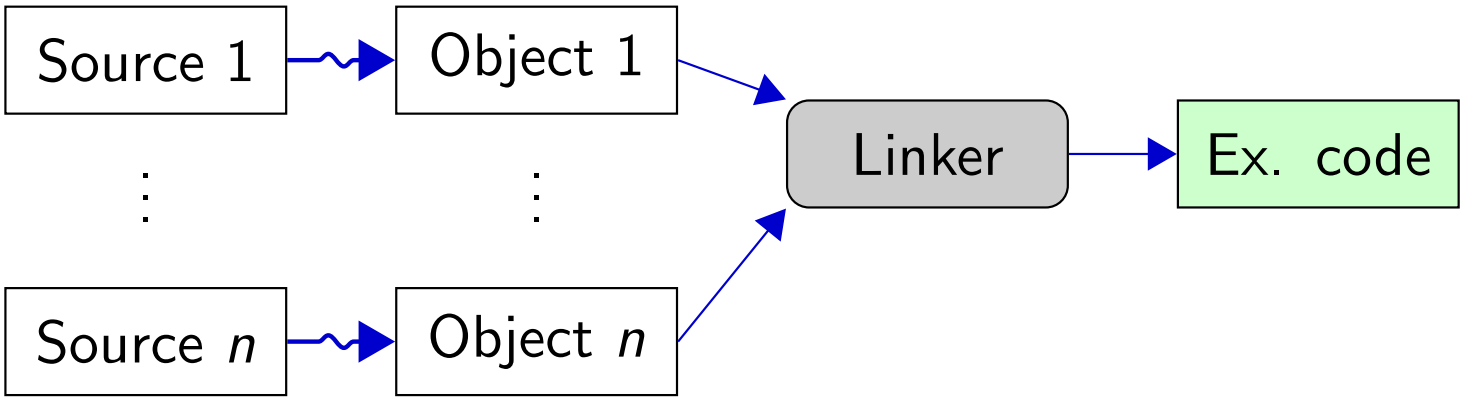


- Each (partial) source file has holes
- Linker can't patch holes from other source files

Splitting the Source Code

Taking advantage of the compilation process (2/2)

A better approach:



- Compile but don't link each source file
- Feed all object files to linker
- Now the linker sees all fillers to patch holes

So What?

What does all of this gain us?

- Only compile source files that have been changed
- Compiling unchanged source files yields unchanged object files
- Feed changed and unchanged object files to linker
- Less time overall
  - However, it requires more commands

Does This Really Save Time?

Less compiling time, more typing

The disadvantages to this approach:

- More commands to type
- Must remember which files to compile and which to link

There is a tool that manages this process

Build Managers

Tools for managing the construction of a file

A *build manager* handles the process of building a file

- Determines which commands to execute to compile and link
- Can build other types of files as well
- Uses timestamps to decide when to (re)build
- Uses a rules file to specify file relationships

In this lab, we will use the `make` utility

Speaking of Labs. . .

It's time to start the experimentation

Please do the following:

1. Create a folder for this course
2. Open a terminal window and use the `cd` command to move to that folder
3. Run the command `git clone` ↩ `ssh://`*`yourlogin`*`@gemini.cc.ysu.edu/home/u155/`↩ `rwkramer/Public/Repository/3700.public.git`
4. Create a folder for today's lab and `cd` into it
5. Copy the files from the lab folder in `3700.public` to today's lab directory
6. Follow the instructions in the enclosed PDF file