

Kenneth M. Anderson Software Methods and Tools CSCI 3308 - Fall Semester, 2004

### Reuse in a Unix Environment

- Two commonly reused software objects in Unix environments
  - source code
  - object code
- Source code Reuse
  - Pro: Can modify to suit new context
  - Con: MUST modify to suit new context
- Object code Reuse
  - Pro: No compilation required; just header file and lib
  - Con: No ability to change functionality; Arch-specific

September 27, 2004

© University of Colorado, 2004

# Libraries

### Unix Library

- a collection of object files, used for some purpose
  - e.g. math libraries, graphics libraries, etc.
- Can be reused in other programs
  - The rules of marshalling (covered in last lecture) ensure that the compiler knows how to call the object code contained in the library
    - Remember that object code is architecture-specific

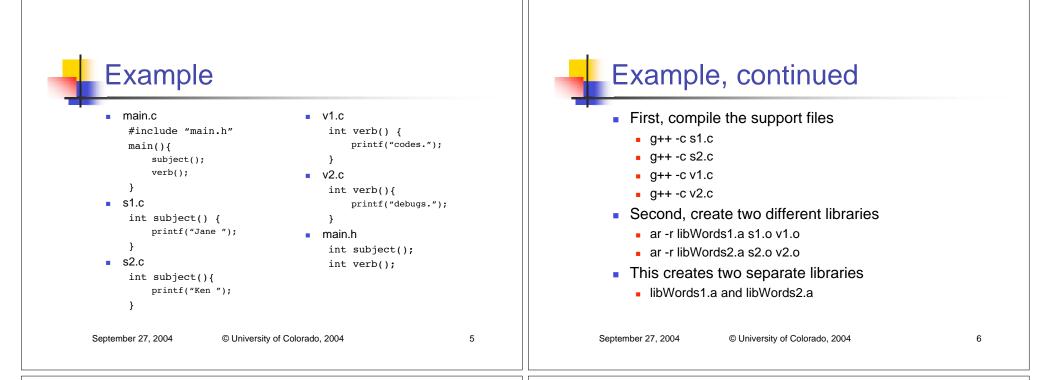
# Creating a Library

- Compile .c files to create .o files
- Use the ar command to create a library from the .o files
  - The .o files are stored in the archive such that they can be extracted at a later time
  - This allows a linker to be smart about using the object code in libraries
    - e.g. only those functions used are placed in the linked executable

3

September 27, 2004

4



### Checking library contents

- ar -t libWords1.a
  - s1.0
  - v1.o
- ar -t libWords2.a
  - s2.o
  - v2.o

- strings libWords1.a
   ...
  - Jane
  - codes.
  - ···
  - strings libWords2.a
    - ...
    - Ken
    - debugs.
  - ...

# Example, continued Third, compile main g++ -c main.c Fourth, link executable g++ main.o -o main1 -lWords1 g++ main.o -o main2 -lWords2 Fifth, run programs main1 -> Jane codes. main2 -> Ken debugs.

7

September 27, 2004

### More info on ar command

- ar is the ARchive command
- It is similar to tar: Tape Archive
  - Both store multiple files as a single collection
  - ar focuses on storing .o files to create libraries
- The similarity ends there

September 27, 2004

 the command flags and behavior of these commands are sometimes quite different

© University of Colorado, 2004

### ar command syntax ar (d|q|r|t) archive [files...] • r - Replace • replace .o files in archive with specified files • q - Quick append • append specified files to archive • d - Delete • delete specified files from archive • t - Table of Contents

- print table of contents of archive
- Note: This is just a sample of ar's functionality; see the ar man page for more details
- September 27, 2004

9

© University of Colorado, 2004

10

### Using Unix Libraries

- In order to use a Unix library, a compiler needs to know the location of the library, the location of its include file, and its name
- Unix compilers (g++, gcc, and cc) have command flags that let you specify this information
  - –I Directory for include files (uppercase i)
  - –L Directory for Libraries
  - –1 Name of library (lowercase L)

# More on include directories

- Any source file that wants to make use of a library, must include its header file
- The -I flag specifies a directory name for this purpose
- When a compiler encounters a "#include" statement, it looks in the current directory and the directory specified by the -I flag for the file

<ul> <li>The -L option specifies a directory where Unix libraries are stored</li> <li>When a linker needs to locate a library (in order to link it into an executable), the linker will look in the directory specified by the -L flag</li> <li>Note: you can have more than one -L and -I flags in a single command</li> </ul>	<ul> <li>The -1 flag (lowercase L) specifies the name of a Unix library</li> <li>The compiler assumes that all libraries begin with "lib" and end in ".a"</li> <li>As such, you write "-lmath" rather than "-llibmath.a"</li> <li>The latter would cause the compiler to look for a file called liblibmath.a.a!</li> </ul>
September 27, 2004 © University of Colorado, 2004 13	September 27, 2004 © University of Colorado, 2004 14
Note: Order is significant • The order of -1 flags is significant g++ main.c -o main -1Words1 -1Words2	Brooks' Corner: Why Did The Tower of Babel Fail? • Communication, (the lack of it) • This made it impossible to coordinate

### More on the Workbook Reducing communication paths Communication needs are reduced by OS/360 division of labor • Each programmer should see all the material specialization of function Each book was updated quickly (one-day) A tree structure often results from applying this Problem principle The workbook grew to 5 feet thick! However this serves power structures better than They switched to microfiche communication (since communication between siblings is We need to take advantage of on-line artifacts. often needed) information management techniques like open So communication structure is often a network hypermedia, information retrieval, and the WWW September 27, 2004 © University of Colorado, 2004 17 September 27, 2004 © University of Colorado, 2004 18

### **Organizational Structure**

- Brooks outlines
  - mission, producer, director, schedule, division of labor, and interfaces between the parts
- The new items are the producer and the director
  - producer: manages project and obtains resources
  - director: manages technical details
- Microsoft's program and product manager
  - former is director, latter does more marketing than Brooks specifies for producer but has some overlap