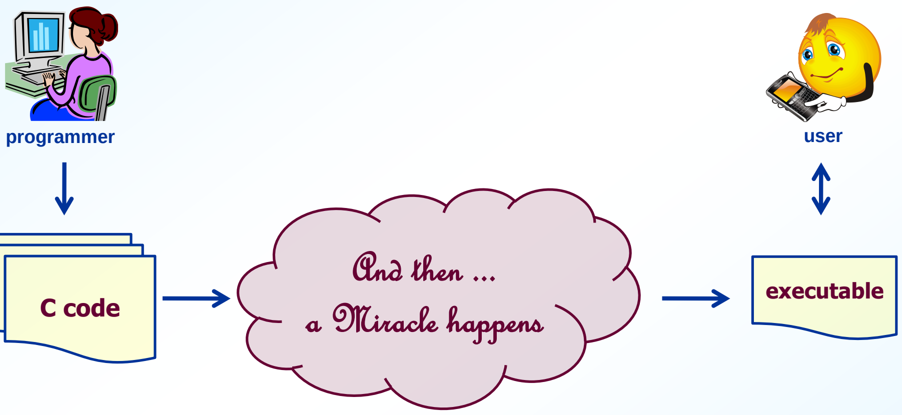
Section 4
Program Building

4.1 Overview



Overview (cont.)

- What is program building?
  - translation of source code into machine code
    - source code is written in a high-level programming language
      - cannot be executed directly by the CPU
    - machine code written in a low-level machine language
      - can be executed directly
  - creation of an executable file from one or more source files

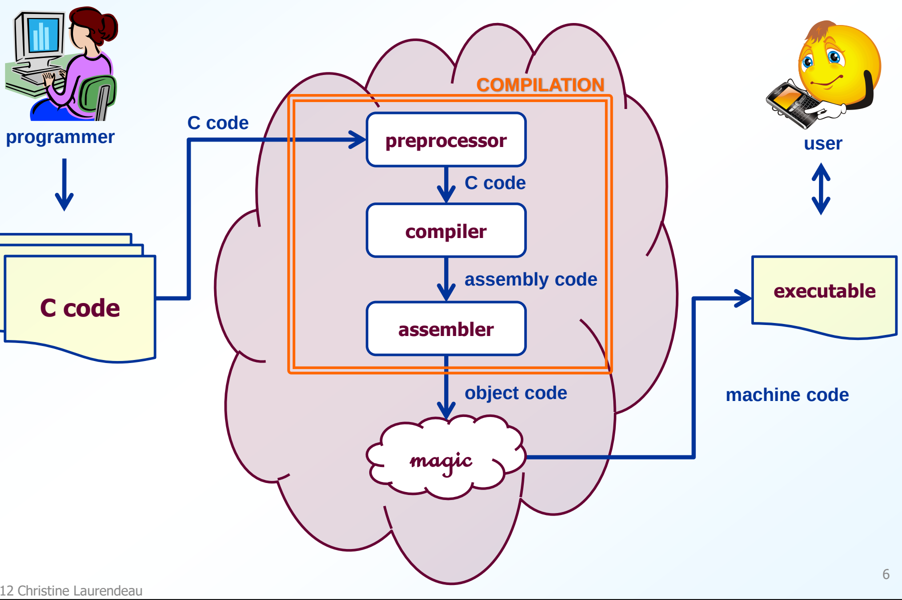
Overview (cont.)

- What is a program executable?
  - a file that contains machine code instructions
    - these instructions are OS and CPU dependent
    - you cannot compile on one platform and run on another
- Characteristics of an executable
  - consists of code from multiple source files
    - your code, other people's code, libraries
  - must have **one** **`main`** function

Overview (cont.)

- Transforming C code into an executable
  - compilation
    - transforms C code to object code
    - 1-to-1 correspondence between C files and object files
  - linking
    - transforms object code to an executable
    - one or more object files linked into one executable

4.2 Compilation



Preprocessing

- What does preprocessing do?
  - interprets all preprocessing directives of one source file
    - text substitution
      - including library header files, defining constants and aliases, etc.
    - conditional compilation
    - directives begin with the **#** symbol
  - input:
    - source code from one source file
  - output:
    - new source code with substitutions incorporated
    - use **`-E`** option to stop after preprocessing

Preprocessing Header Files

- What is a header file?
  - file containing information needed by multiple source files
    - data type definitions
    - function prototypes
    - ... more on this later ...
  - **never** contains function implementations or any statements!
- Characteristics
  - copied into source file during preprocessing
    - use angle brackets for header file from library
    - use double quotes for header file from current directory

Compiling

- What does compiling do?
  - translates source code to assembly code
    - performs optimizations
    - resolves internal function addresses
      - functions with implementations in the same source file
  - input:
    - source code from one source file
  - output:
    - corresponding assembly code
      - human readable version of machine code
    - use `-S` option to stop after compiling

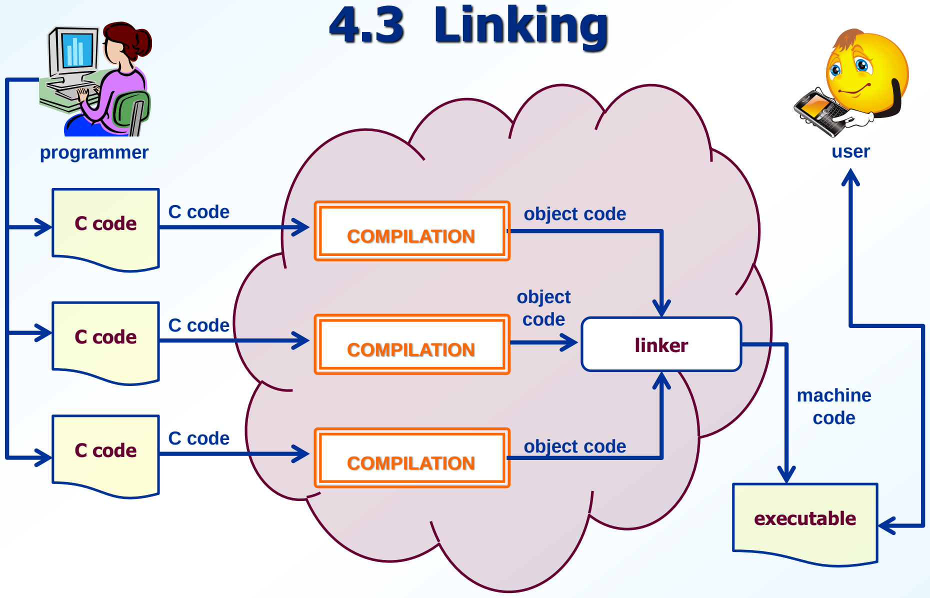
```
void sumIterative(int numElements,
                  int *intArray,
                  int *sum)
{
  int i;
  *sum = 0;

  for (i=0; i<numElements; ++i)
    *sum += intArray[i];
}
```

```
sumIterative:
.LFB2:
        ... start of proc stuff ...
        movl    16(%ebp), %eax
        movl    $0, (%eax)
        movl    $0, -4(%ebp)
        jmp     .L13
.L14:
        movl    16(%ebp), %eax
        movl    (%eax), %edx
        movl    -4(%ebp), %eax
        sall    $2, %eax
        addl    12(%ebp), %eax
        movl    (%eax), %eax
        addl    %eax, %edx
        movl    16(%ebp), %eax
        movl    %edx, (%eax)
        addl    $1, -4(%ebp)
.L13:
        movl    -4(%ebp), %eax
        cmpl    8(%ebp), %eax
        jl      .L14
        ... end of proc stuff ...
```

Assembling

- What does assembling do?
  - translates assembly code to object code
  - input:
    - assembly code from one source file
  - output:
    - corresponding object code
    - use `-c` option to stop after assembling
      - this is **essential** if using multiple source files

4.3 Linking



Linking (cont.)

- What does linking do?
  - combines code from multiple object files into one executable
  - resolves external function addresses
    - functions with implementations in different object file
    - library functions
  - input:
    - object code from multiple object files
  - output:
    - one executable file

Linking (cont.)

- Why separate compiling and linking?
  - object files may come from different source languages
  - you can link in object code from libraries
  - you only need to recompile the source files that have changed
    - compilation can be slow
    - unnecessary compilation must be avoided

Linking in Libraries

- What is a library file?
  - collection of related functions written by other programmers
- To use a library:
  - include the header file
  - link in the object file
- C standard library: `libc.a`
  - always linked in by default

Linking in Libraries (cont.)

- Types of linking
  - static linking
    - library object code is copied into executable
    - increases size of executable
    - faster execution time
  - dynamic linking
    - default setting
    - library object code is loaded at runtime, as needed
    - small executable, but slower execution time

4.4 Makefiles

- What is a Makefile?
  - a text file
  - a tool used to organize compiling and linking commands
  - manages dependencies between source and header files
    - only recompiles source files that have changed since last make

Makefiles (cont.)

- Characteristics of Makefiles
  - use special syntax
  - invoked from shell using **`make`** command
  - composed of two parts
    - dependencies
    - commands

Makefiles (cont.)

- Why use a Makefile?
  - keeps track of what needs to be recompiled
    - compares timestamp on source file to timestamp on object file
    - if source file is newer, it gets recompiled
  - decreases number of commands for programmer
    - with Makefile:
      - one make command
    - without Makefile:
      - one compilation command for each source file
      - one linking command

Makefiles (cont.)

- Makefile macros
  - similar to variables
  - can be used to
    - specify compilation options
    - define groups of files
  - common macros
    - **`all`**
      - define all final executables
    - **`clean`**
      - remove all intermediate files