

EECS 22: Advanced C Programming

Lecture 7

Rainer Dömer

doemer@uci.edu

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

Lecture 7: Overview

- Compiler Components
 - Preprocessor
 - Compiler
 - Linker
- Translation Units
 - Multiple Modules
 - Compilation
- Application example **PhotoLab**
 - Decomposition into Modules
 - Module **FileIO**
 - Module **Age**
 - Module **Main**

Compiler Components

- Introduction
 - C compilation process is a sequence of phases
 - Preprocessing (handle # directives)
 - Scanning and parsing (generate internal data structure)
 - Instruction generation (emit stream of CPU instructions)
 - Assembly (generate binary object file)
 - Linking (combine objects into executable file)
 - C compiler consists of separate components
 - Preprocessor (processes # directives)
 - Compiler (compiles and assembles code)
 - GNU compiler contains separate assembler
 - Linker (processes object files and libraries)

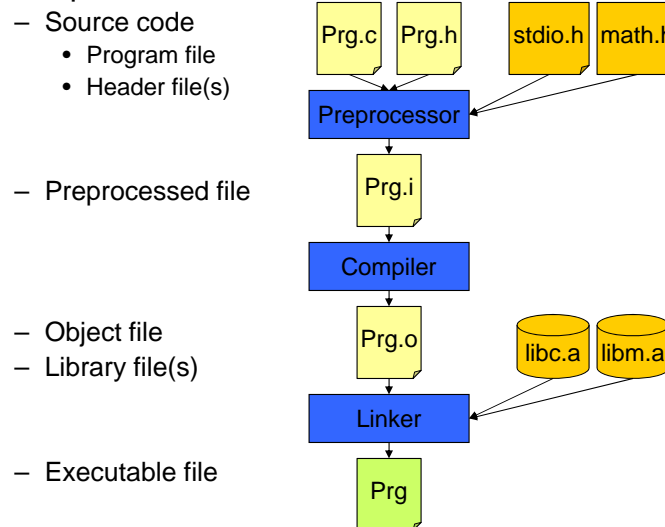
EECS22: Advanced C Programming, Lecture 7

(c) 2012 R. Doemer

3

Compiler Components

- Compilation Phases and Intermediate Files



EECS22: Advanced C Programming, Lecture 7

(c) 2012 R. Doemer

4

Source Code

- Source Files
 - Header files: **Program.h**
 - Inclusion of required header files
 - Definitions of exported constants
 - Declarations of exported global variables
 - Declarations of exported functions
 - Program files: **Program.c**
 - Inclusion of required header files
 - Declaration and definition of local variables
 - Declaration and definition of local functions
 - Definitions of exported global variables
 - Definitions of exported functions

EECS22: Advanced C Programming, Lecture 7

(c) 2012 R. Doemer

5

Preprocessor

- C Preprocessor
 - removes comments and tokenizes the source code
 - handles preprocessing (#) directives
- Preprocessing Directives
 - Constant definition
 - Simple textual replacement
 - Definition may be undefined
 - Macro definition
 - Textual replacement with one or more arguments
 - Parameters may be “stringified”
 - Parameters may be concatenated
 - Header file inclusion

```
#define MAX 100
int A[MAX];
...
#undef MAX
```

```
#define ABS(x) (x>0 ? x : -x)
```

```
#define string(x) #x
```

```
#define cat(x,y) x##y
```

```
#include <stdio.h>
#include "Constants.h"
```

EECS22: Advanced C Programming, Lecture 7

(c) 2012 R. Doemer

6

Preprocessor

- Preprocessing Directives (continued)
 - Conditional compilation

```

#define DEBUG /* comment out to turn debugging off */
...
#ifdef DEBUG
printf("value of x is now %d\n", x);
#endif
...

#if VERSION > 3
x = f_latest(y); /* run version 3 algorithm */
#elif VERSION > 2
x = f_advanced(y); /* run version 2 algorithm */
#else
x = f(y); /* run regular algorithm */
#endif

```

Compiler

- GNU C Compiler Options
 - Language Dialect
 - `-ansi` selects ANSI-C language semantics
 - Warnings
 - `-Wall` enables all warnings
 - Compilation phases
 - `-E` preprocessing only, result is preprocessed file (.i)
 - `-S` code generation only, result is assembly file (.s)
 - `-c` compilation only, result is an object file (.o)
 - (none) all compilation phases, result is executable
 - Output File
 - `-o name` selects output filename (default a.out)

Compiler

- GNU C Compiler Options (continued)
 - Preprocessor definitions
 - `-Dmacro` command-line equivalent of `#define macro`
 - `-Dm=def` command-line equivalent of `#define m def`
 - `-Umacro` command-line equivalent of `#undef macro`
 - Support for Debugging
 - `-g` generate symbol tables needed by debugger
 - `-DDEBUG` turn on conditional code for debugging
 - Optimization Options
 - `-O2` optimize the generated code for speed (level 2)
 - `-DNDEBUG` turn off debugging support (i.e. assertions)

Linker

- Object files
 - **Program.o**
 - Compiled object code of source file `Program.c`
 - Use option `-c` in GNU compiler call to create object files
`gcc -c Program.c -o Program.o -Wall -ansi`
 - **Library.a** or **Library.so**
 - Archive or shared library of compiled object files
- Executable file
 - **Program**
 - Object files and libraries *linked* together into a complete file ready for loading and execution
 - GNU compiler recognizes object files by `.o` suffix, so object files and libraries require no special option
`gcc Program.o -lc -lm -o Program`

Translation Units

- Set of Modules
 - C programs can be partitioned into multiple modules and compiled in separate translation units
 - Modules typically consist of
 - Module header file (file suffix `.h`)
 - Module program file (file suffix `.c`)
 - Module object or library file (file suffix `.o`, `.a`, or `.so`)
- Compiling the Application
 - Compiled modules are then *linked* together
 - Linker combines object files and required libraries into an executable file
 - `gcc Program.o Mod1.o Mod2.o -lc -lm -o Program`

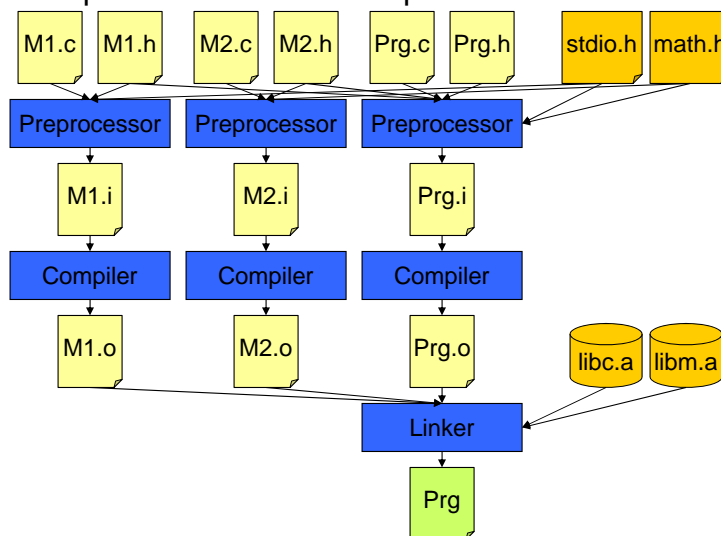
EECS22: Advanced C Programming, Lecture 7

(c) 2012 R. Doemer

11

Translation Units

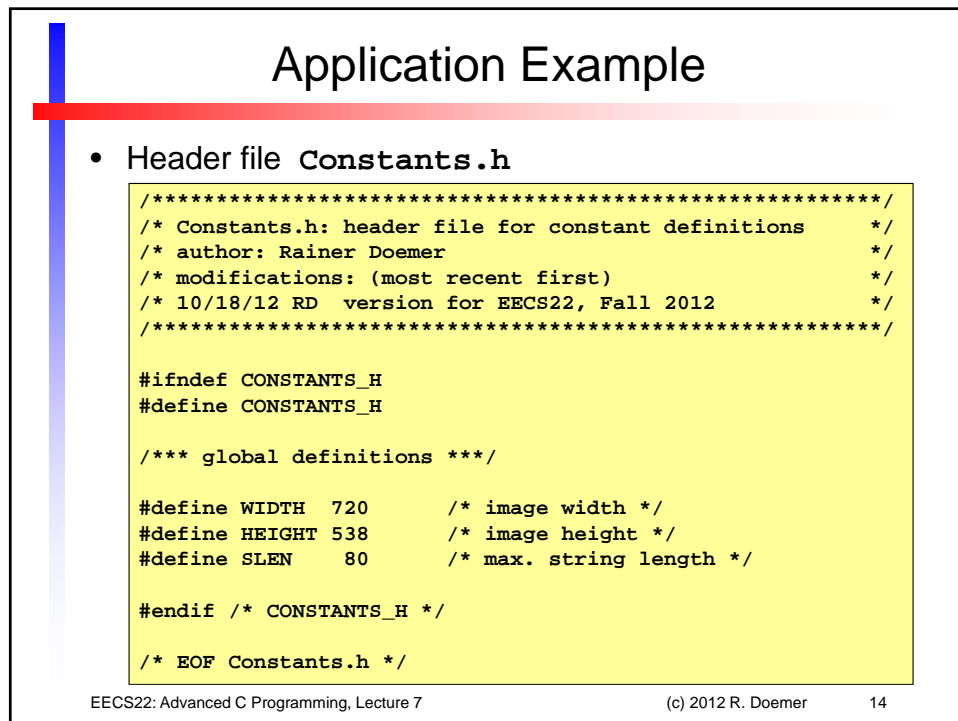
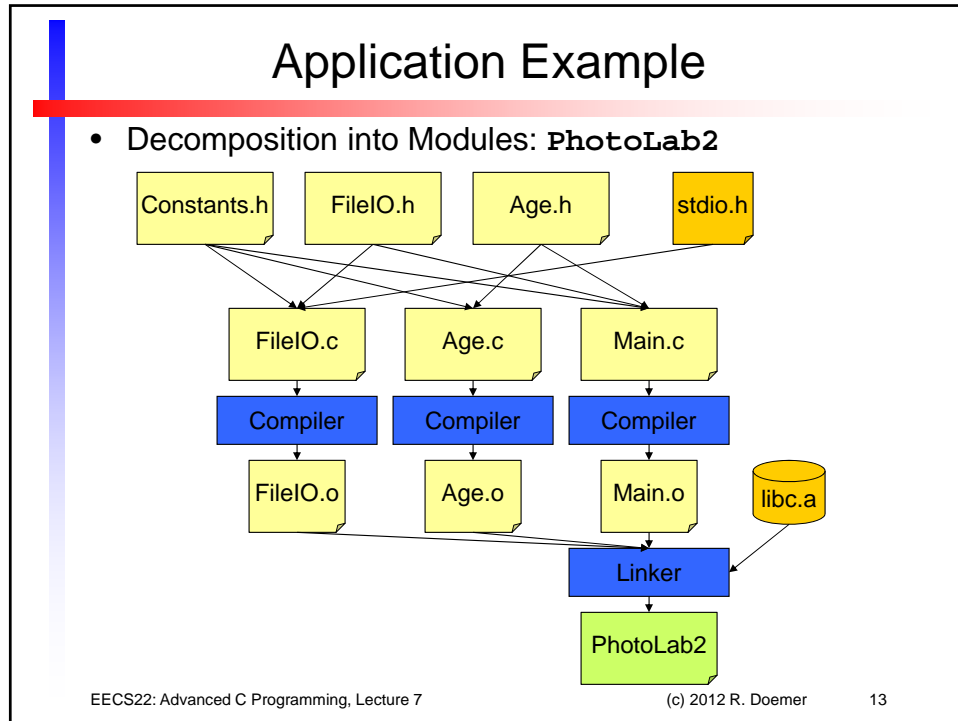
- Compilation Flow with Multiple Modules



EECS22: Advanced C Programming, Lecture 7

(c) 2012 R. Doemer

12



Application Example

- Header file `FileIO.h`

```

/*****
/* FileIO.h: header file for I/O module */
/*****
#ifndef FILE_IO_H
#define FILE_IO_H

#include "Constants.h"

int ReadImage( /* read image from file */
    char Filename[SLEN],
    unsigned char R[WIDTH][HEIGHT],
    unsigned char G[WIDTH][HEIGHT],
    unsigned char B[WIDTH][HEIGHT]);

int SaveImage( /* write image to file */
    char Filename[SLEN],
    unsigned char R[WIDTH][HEIGHT],
    unsigned char G[WIDTH][HEIGHT],
    unsigned char B[WIDTH][HEIGHT]);

#endif /* FILE_IO_H */
/* EOF FileIO.h */

```

Application Example

- Module file `FileIO.c`

```

/*****
/* FileIO.c: program file for I/O module */
/*****
#include <stdio.h>
#include "FileIO.h"

/** function definitions */

int ReadImage(char Filename[SLEN],
    unsigned char R[WIDTH][HEIGHT],
    unsigned char G[WIDTH][HEIGHT],
    unsigned char B[WIDTH][HEIGHT])
{ /* ... function body ... */
}

int SaveImage(char Filename[SLEN],
    unsigned char R[WIDTH][HEIGHT],
    unsigned char G[WIDTH][HEIGHT],
    unsigned char B[WIDTH][HEIGHT])
{ /* ... function body ... */
}
/* EOF FileIO.c */

```


Application Example

- Header file `Age.h`

```

/*****
 * Age.h: header file for aging operation
 *****/

#ifndef AGE_H
#define AGE_H

/** header files */
#include "Constants.h"

/** function declarations */
void Age( /* age the image */
         unsigned char R[WIDTH][HEIGHT],
         unsigned char G[WIDTH][HEIGHT],
         unsigned char B[WIDTH][HEIGHT]);

#endif /* AGE_H */
/* EOF Age.h */

```

Application Example

- Module file `Age.c`

```

/*****
 * Age.c: program file for aging operation
 *****/

#include "Age.h"

/** function definitions */
/* age the image so that it looks like an old photo */
void Age(
    unsigned char R[WIDTH][HEIGHT],
    unsigned char G[WIDTH][HEIGHT],
    unsigned char B[WIDTH][HEIGHT])
{
    /* ... function body ... */
}
/* EOF Age.c */

```

Application Example

- Module file `Main.c`

```

/*****
/* Main.c: main program file */
/*****
#include "Constants.h"
#include "FileIO.h"
#include "Age.h"

int main(void)
{
    unsigned char R[WIDTH][HEIGHT];
    unsigned char G[WIDTH][HEIGHT];
    unsigned char B[WIDTH][HEIGHT];

    if (ReadImage("ucisailing.ppm", R, G, B) != 0)
        { return 10; }
    Age(R, G, B);
    if (SaveImage("ucisailing1965.ppm", R, G, B) != 0)
        { return 10; }

    return 0;
} /* end of main */
/* EOF Main.c */

```

EECS22: Advanced C Programming, Lecture 7

(c) 2012 R. Doemer

19

Application Example

- Build and Compilation session:

```

% vi Constants.h
% vi FileIO.h
% vi FileIO.c
% vi Age.h
% vi Age.c
% vi Main.c

```

```

% gcc -c FileIO.c -o FileIO.o -Wall -ansi
% gcc -c Age.c -o Age.o -Wall -ansi
% gcc -c Main.c -o Main.o -Wall -ansi
% gcc FileIO.o Age.o Main.o -o PhotoLab2
%

```

ucisailing.ppm



ucisailing1965.ppm



EECS22: Advanced C Programming, Lecture 7

(c) 2012 R. Doemer

20