EECS 22: Advanced C Programming

Lecture 10

Rainer Dömer

doemer@uci.edu

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

Lecture 10: Overview

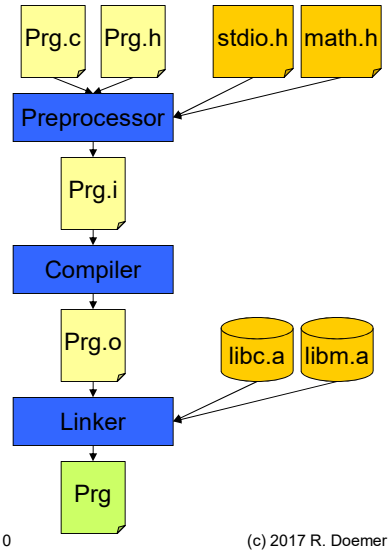
- Compiler Components
  - Preprocessor
  - Compiler
  - Linker
- Translation Units
  - Multiple modules
  - Compilation
- Application Example **`PhotoLab2`**
  - Decomposition into modules
    - Modules **`FileIO`**, **`Age`**, **`Main`**

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)

Compiler Components

- Compilation Phases and Intermediate Files
  - Source code
    - Program file
    - Header file(s)
  - Preprocessed file
  - Object file
  - Library file(s)
  - Executable file



Source Code

- Source Files
  - Header files: **`Program.h`**
    - Inclusion of other needed header files
    - Declaration and definition of exported constants and types
    - Declaration of exported global variables
    - Declaration of exported functions
  - Program files: **`Program.c`**
    - Inclusion of needed header files
    - Declaration and definition of internal constants and types
    - Declaration and definition of internal variables
    - Declaration and definition of internal functions
    - Definition of exported global variables
    - Definition of exported functions

Preprocessor

- C Preprocessor
  - tokenizes the source code and removes comments
  - handles preprocessing (#) directives
- Preprocessing Directives
  - Constant definition
    - Simple textual replacement
      - ➢ Definition may be undefined

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

  - Macro definition

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

    - Textual replacement with one or more arguments
      - ➢ Parameters may be "stringified"

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

      - ➢ Parameters may be concatenated

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

  - Header file inclusion

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

Preprocessor

- Preprocessing Directives (continued)
  - Conditional compilation

```
#define DEBUG     /* comment out to turn debugging off */
#define VERSION 2 /* select an algorithm to be used */

...

#ifdef DEBUG
printf("value of x is now %d\n", x);
#endif

...

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


Compiler

- GNU C Compiler Options
  - Language Dialect
    - **`-ansi`** selects ANSI-C language semantics
    - **`-std=c99`** selects C99 standard
  - 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

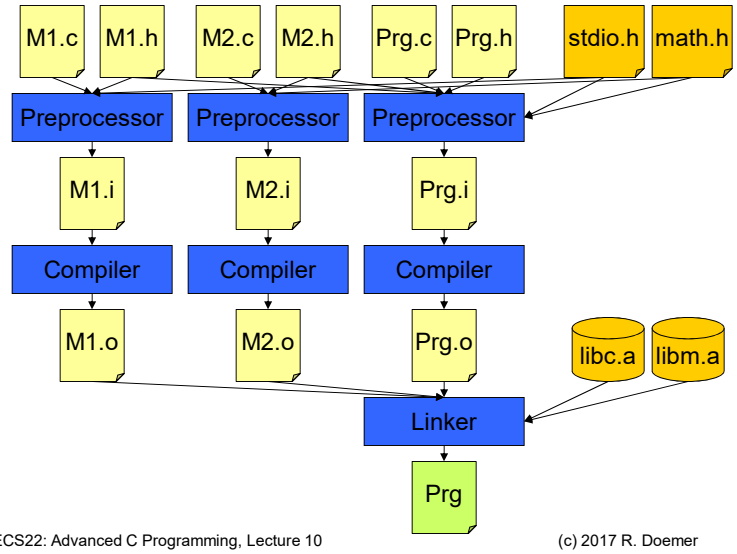
- Input: Object files
  - **`Program.o`**
    - Compiled object file of source file **`Program.c`**
  - **`libName.a`**
    - Archive of (several) compiled object files
  - **`libName.so`**
    - Shared object file (shared library) of compiled object files
- Output: Executable file
  - **`Program`**
    - Object files and libraries *linked* together into a complete file ready for loading and execution
    - GNU compiler recognizes object files by .o suffix
    - Library files can be referenced with the **`-lName`** option
    - ➢ **`gcc Program.o -lc -lm -o Program`**

Multiple Translation Units

- Set of Modules
  - C programs can be partitioned into multiple modules and compiled in separate translation units
  - A module typically consists 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`**

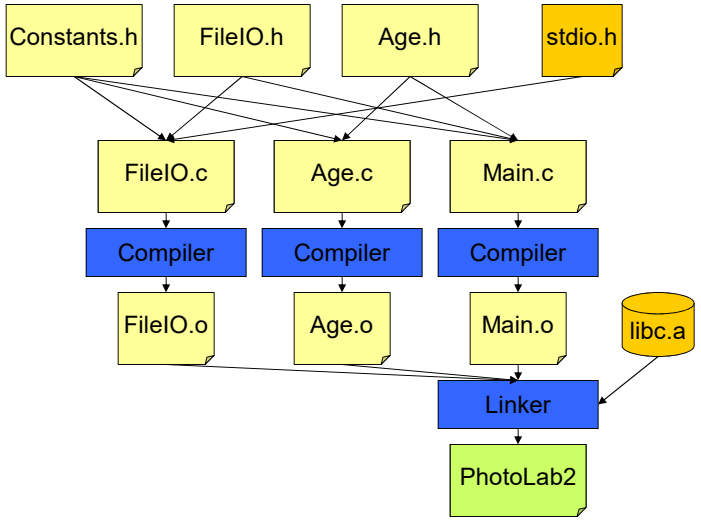
Multiple Translation Units

- Compilation Flow with Multiple Modules



Application Example

- Decomposition into Modules: **`PhotoLab2`**



Application Example

- Header file **`Constants.h`**

```
/*************************************************************/
/* Constants.h: header file for constant definitions         */
/* author: Rainer Doemer                                     */
/* modifications: (most recent first)                        */
/* 10/12/17 RD  version for EECS22, Fall 2017                */
/*************************************************************/

#ifndef CONSTANTS_H
#define CONSTANTS_H

/*** global definitions ***/

#define WIDTH  600      /* image width */
#define HEIGHT 400      /* image height */
#define SLEN    80      /* max. string length */

#endif /* CONSTANTS_H */

/* EOF Constants.h */
```

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

int SaveImage(      /* write image to file */
        const 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 */
```

EECS

Application Example

- Module file **`FileIO.c`**

```
/*********************************************************/
/* FileIO.c: program file for I/O module                 */
/*********************************************************/

#include <stdio.h>
#include "FileIO.h"

/*** function definitions ***/

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

EECS

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 (LoadImage("HSSOE.ppm", R, G, B) != 0)
       { return 10; }
    Age(R, G, B);
    if (SaveImage("HSSOE1965.ppm", R, G, B) != 0)
       { return 10; }

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


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
% PhotoLab2
%
```

HSSOE.ppm

HSSOE1965.ppm