EECS 22: Advanced C Programming

Lecture 7 (TuTh)

Rainer Dömer

doemer@uci.edu

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

Recursion Revisited (supplemental slides from Tuesday lecture)

- Example: Fibonacci series
  - Mathematical properties:
    - The first two numbers are 0 and 1
    - Every subsequent number is the sum of the previous two
  - Recursive definition:
    - Base case: *fibonacci(0) = 0, fibonacci(1) = 1*
    - Recursion step: *fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)*

Recursion Revisited

- Program example: **`Fibonacci.c`** (part 1/2)

```
/* Fibonacci.c: example demonstrating recursion */
/* author: Rainer Doemer                        */
/* modifications:                               */
/* 11/14/04 RD  initial version                 */

#include <stdio.h>

/* function definition */

long fibonacci(long n)
{
   if (n <= 1)  /* base case */
      { return n;
       } /* fi */
   else         /* recursion step */
      { return fibonacci(n-1) + fibonacci(n-2);
       } /* esle */
} /* end of fibonacci */

/* main function */
...
```


Recursion Revisited

- Program example: **`Fibonacci.c`** (part 2/2)

```
...
int main(void)
{
   /* variable definitions */
   long int n, f;

   /* input section */
   printf("Please enter value n: ");
   scanf("%ld", &n);

   /* computation section */
   f = fibonacci(n);

   /* output section */
   printf("The %ld-th Fibonacci number is %ld.\n", n, f);

   /* exit */
   return 0;
} /* end of main */

/* EOF */
```

Recursion Revisited

- Timed example session: **`Fibonacci.c`**

```
% /usr/bin/time -f "%U seconds" ./Fibonacci
Please enter value n: 41
The 41-th Fibonacci number is 165580141.
2.37 seconds
% /usr/bin/time -f "%U seconds" ./Fibonacci
Please enter value n: 42
The 42-th Fibonacci number is 267914296.
3.71 seconds
% /usr/bin/time -f "%U seconds" ./Fibonacci
Please enter value n: 43
The 43-th Fibonacci number is 433494437.
5.62 seconds
% /usr/bin/time -f "%U seconds" ./Fibonacci
Please enter value n: 44
The 44-th Fibonacci number is 701408733.
8.97 seconds
% /usr/bin/time -f "%U seconds" ./Fibonacci
Please enter value n: 45
The 45-th Fibonacci number is 1134903170.
14.26 seconds
%
```


Recursion Revisited

- Example Revisited: Fibonacci series
  - Recursive definition:
    - Base case: $fibonacci(0) = 0,\ fibonacci(1) = 1$
    - Recursion step: $fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)$
  - Mathematical properties:
    - The first two numbers are 0 and 1
    - Every subsequent number is the sum of the previous two
  - ➢ Problem:
    - Program run-time grows exponentially! (factor 1.6, golden ratio)
  - ➢ Idea:
    - If we *remember the previously calculated numbers*, we can calculate the next number immediately!
    - ➢ Whenever a new number is calculated, keep it stored in a **`static`** array in memory
    - ➢ When a number is present in the memory, just look it up

Recursion Revisited

- Program example: **`Fibonacci2.c`** (part 1/3)

```
/* Fibonacci2.c: example demonstrating recursion */
/* author: Rainer Doemer                         */
/* modifications:                                */
/* 11/09/11 RD  version with 'static' memory     */
/* 11/14/04 RD  initial version                  */

#include <stdio.h>

#define MEM_SIZE 100

/* function definition */

...
```


Recursion Revisited

- Program example: **`Fibonacci2.c`** (part 2/3)

```
long fibonacci(long n)
{  static long fib[MEM_SIZE] = {0,1}; /* memory */
   if (n <= 1)        /* base case */
      { return n;
       } /* fi */
   else               /* previouly calculated results */
      { if (n < MEM_SIZE && fib[n])
           { return fib[n];
            } /* fi */
        else          /* recursion step */
           { long f;
             f = fibonacci(n-1) + fibonacci(n-2);
             if (n < MEM_SIZE)
                { fib[n] = f;   /* remember this */
                 } /* fi */
             return f;
           } /* esle */
       } /* esle */
} /* end of fibonacci */
...
```

Recursion Revisited

- Program example: **`Fibonacci2.c`** (part 3/3)

```
...
int main(void)
{
   /* variable definitions */
   long int n, f;

   /* input section */
   printf("Please enter value n: ");
   scanf("%ld", &n);

   /* computation section */
   f = fibonacci(n);

   /* output section */
   printf("The %ld-th Fibonacci number is %ld.\n", n, f);

   /* exit */
   return 0;
} /* end of main */

/* EOF */
```


Recursion Revisited

- Timed example session: **`Fibonacci2.c`**

```
% /usr/bin/time -f "%U seconds" ./Fibonacci2
Please enter value n: 41
The 41-th Fibonacci number is 165580141.
0.00 seconds
% /usr/bin/time -f "%U seconds" ./Fibonacci2
Please enter value n: 42
The 42-th Fibonacci number is 267914296.
0.00 seconds
% /usr/bin/time -f "%U seconds" ./Fibonacci2
Please enter value n: 43
The 43-th Fibonacci number is 433494437.
0.00 seconds
% /usr/bin/time -f "%U seconds" ./Fibonacci2
Please enter value n: 44
The 44-th Fibonacci number is 701408733.
0.00 seconds
% /usr/bin/time -f "%U seconds" ./Fibonacci2
Please enter value n: 45
The 45-th Fibonacci number is 1134903170.
0.00 seconds
%
```

Part 2: Overview

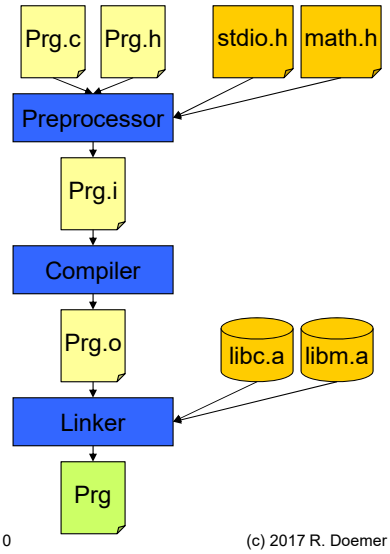
- Compiler Components
  - Preprocessor
  - Compiler
  - Linker
- Translation Units
  - Multiple modules
  - Compilation
- Application Example **`PhotoLab2`**
  - Decomposition into modules
    - Modules **`FileIO`**, **`Age`**, **`Main`**
- Shared Libraries
  - Static object file archives
  - Dynamically loaded shared objects

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



EECS22: Advanced C Programming, Lecture 10 (c) 2017 R. Doemer 13

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

EECS22: Advanced C Programming, Lecture 10 (c) 2017 R. Doemer 14

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`**

EECS22: Advanced C Programming, Lecture 10 (c) 2017 R. Doemer 19

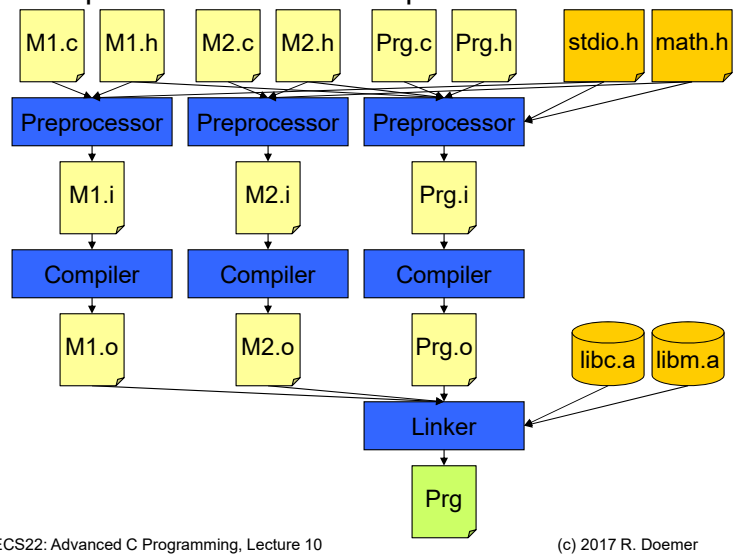
Multiple Translation Units

- 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`**

EECS22: Advanced C Programming, Lecture 10 (c) 2017 R. Doemer 20

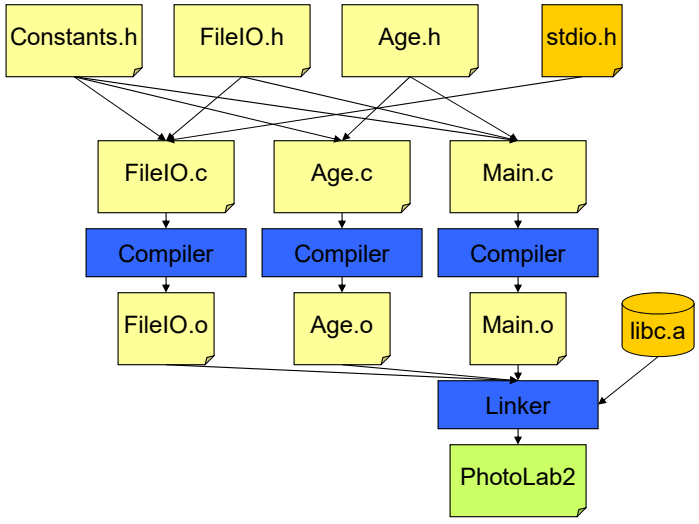
Multiple Translation Units

- 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 WriteImage(      /* 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 */
```

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

Shared Libraries

- When a set of modules is useful for multiple applications, a *shared library* can be built for reusing the functionality provided by the modules
  1) *Static library*: Archive of object files (linked at compile-time)
     - Examples: **`libC.a`**, **`libM.a`**
     - Archive tools **`ar`** and **`ranlib`**
       - can combine multiple object files into linker archives
  2) *Dynamic library*: Shared object file (loaded at run-time)
     - Examples: **`libC.so`**, **`libM.so`**
     - Compiler **`gcc`**
       - can create position-independent code (**`-fPIC`**)
       - can create shared libraries (**`-shared`**)
       - can create executable files that load shared object files at run-time (**`-L`*`path`* `-Xlinker -R -Xlinker` *`path`***)

Shared Libraries

- Example: Shared **`FileIO`** Library for **`PhotoLab3`**
- Option 1: Static shared library
  - Preparing the archive file:

```
% vi Constants.h
% vi FileIO.h
% vi FileIO.c
```

```
% gcc -c FileIO.c -o FileIO.o -Wall –ansi
% ar rc libFileIO.a FileIO.o
% ranlib libFileIO.a
%
```

  - Building and compiling the application:

```
% vi Age.h
% vi Age.c
% vi Main.c
```

```
% gcc -c Age.c -o Age.o -Wall –ansi
% gcc -c Main.c -o Main.o -Wall –ansi
% gcc Main.o Age.o –lFileIO –L. -o PhotoLab3
% PhotoLab3
%
```


Shared Libraries

- Example: Shared **`FileIO`** Library for **`PhotoLab3`**
- Option 2: Dynamically loaded shared library
  - Preparing the shared object file:

```
% gcc -c –fPIC FileIO.c -o FileIO.o -Wall –ansi
% gcc –shared –fPIC –o libFileIO.so FileIO.o
%
```

  - Building and compiling the application:

```
% gcc -c Age.c -o Age.o -Wall –ansi
% gcc -c Main.c -o Main.o -Wall –ansi
% gcc Main.o Age.o –lFileIO –L.
      –Xlinker –R –Xlinker . -o PhotoLab3
% PhotoLab3
% ldd PhotoLab3
        linux-vdso.so.1 =>  (0x00007fff07550000)
        libFileIO.so => ./libFileIO.so (0x00007f2ddf1ac000)
        libc.so.6 => /lib64/libc.so.6 (0x00000035f5000000)
        /lib64/ld-linux-x86-64.so.2 (0x00000035f4c00000)
%
```