

EECS 22: Advanced C Programming

Lecture 1

Rainer Dömer

doemer@uci.edu

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

Lecture 1: Overview

- Course Administration
 - New courses in Computer Engineering
 - Course overview
 - Course web pages
- Getting started
 - Obtain an account on the EECS Linux server
 - Work in the Linux system environment
- Review of C Programming
 - History of C
 - The first C Program, `HelloWorld.c`
 - General program structure, `Addition.c`

Course Administration

- New programming courses in Computer Engineering
 - EECS22, “Advanced C Programming”
 - EECS22L, “Software Engineering Project in C”
- Replacing Physics 51A, 52A with EECS 22, 22L
 - Expected in 2012/13 catalogue
 - pending approval by the UCI Academic Senate
 - Current students can opt to choose the new plan of study
 - Automatic approval for taking 22/22L to fulfill the 52A/51A requirement
 - Of course, you can still choose to take 52A/51A...
- EECS 22 offered in Fall 2011
 - Instructor: Prof. Rainer Doemer
- EECS 22L to be offered in Winter 2012
 - Instructor: Prof. Pai Chou

EECS22: Advanced C Programming, Lecture 1

(c) 2011 R. Doemer

3

Course Administration

- Changes to Computer Engineering Program
 - Delete Physics 51A and 52A from Math and Basic Science
 - Add EECS 22 and EECS 22L to Core Courses
- Sample Program of Study

FALL	WINTER	SPRING
Freshman		
Mathematics 2A	Mathematics 2B	Mathematics 2D
EECS12	Physics 7C/7LC	Physics 7D, 7LD
General Education	General Education	EECS20
General Education		General Education
Sophomore		
Mathematics 2J	Mathematics 3D	Mathematics 6D
Physics 7E, 52A	Physics 51A , 52B	EECS40
<u>EECS22</u>	<u>EECS22L</u>	EECS70B, 70LB
EECS31	EECS31L	General Education
Junior	EECS70A	

EECS22: Advanced C Programming, Lecture 1

(c) 2011 R. Doemer

4

EECS 22: Advanced C Programming

- “All you want to know about C Programming”
 - Review and reinforce basic C programming concepts
 - Study advanced features in detail
 - Put concepts and tools to their best use
- Features
 - Dynamic data structures using malloc(), free()
 - Keywords 'static', 'register', 'auto', 'extern', 'volatile', ...
 - Advanced data types, Unicode, variable-length arguments
 - Libraries, Makefile, ...
- Tools
 - C preprocessor, compiler, and linker
 - Debugger 'gdb', profiler 'gprof', 'gcov', ...
 - Dynamic memory allocation checker 'valgrind'

EECS22: Advanced C Programming, Lecture 1

(c) 2011 R. Doemer

5

EECS 22: Advanced C Programming

- Catalogue Data
 - **EECS 22 Advanced C Programming (Credit Units: 3) F.**
 - C language programming concepts.
 - Control flow, function calls, recursion.
 - Basic and composite data types, static and dynamic data structures.
 - Program modules and compilation units.
 - Preprocessor macros.
 - C standard libraries.
 - Prerequisites: EECS 20
 - (Design Units: 1)

EECS22: Advanced C Programming, Lecture 1

(c) 2011 R. Doemer

6

EECS 22: Advanced C Programming

- Course Contents
 - Review of C expressions, statements, control flow
 - Primitive, composite, and user-defined data types
 - Functions and parameter passing semantics
 - Variable scope rules (global, static, auto, extern)
 - Pointers and pointer arithmetic
 - Dynamic memory allocation
 - Dynamic data structures: linked lists, stacks, queues, trees
 - Function pointers and callback
 - Preprocessor definitions, conditionals, and macros
 - Program modules, header files, compilation units
 - Compilation and linking process, Makefile
 - C standard library, external libraries

EECS22: Advanced C Programming, Lecture 1

(c) 2011 R. Doemer

7

EECS 22L: Software Eng. Project in C

- “Developing real C Programs in a Team”
 - Hands-on experience with larger software projects
 - Introduction to software engineering
 - Specification, documentation, implementation, testing
 - Team work
- Features
 - Design efficient data structures, APIs
 - Utilize programming modules, build libraries
 - Develop and optimize contemporary software applications
- Tools
 - Scripting ‘make’
 - Version control ‘cvs’
 - Testing and debugging with ‘gdb’, ‘gprof’, ‘valgrind’, ...

EECS22: Advanced C Programming, Lecture 1

(c) 2011 R. Doemer

8

EECS 22L: Software Eng. Project in C

- Catalogue Data
 - **EECS 22L Software Engineering Project in C Language (Credit Units: 3) W.**
 - Hands-on experience with the ANSI-C programming language.
 - Medium-sized programming projects, team work.
 - Software specification, documentation, implementation, testing.
 - Definition of data structures and application programming interface.
 - Creation of program modules, linking with external libraries.
 - Rule-based compilation, version control.
 - Prerequisites: EECS 22
 - (Design Units: 3)

EECS22: Advanced C Programming, Lecture 1

(c) 2011 R. Doemer

9

EECS 22L: Software Eng. Project in C

- Course Contents
 - Software engineering topics, including specification, documentation, implementation, testing, debugging, project planning, organization, maintenance, version control, organization of source files, header files, modules
 - Compilation flow, Makefile, shell scripting
 - Definition of data structures and application programming interface
 - External libraries, system programming, POSIX API, interrupts
 - Introduction to C++ language, syntax and semantics, references, inline functions, default arguments, classes, members, and methods, object creation and deletion (constructors, destructors)

EECS22: Advanced C Programming, Lecture 1

(c) 2011 R. Doemer

10

Course Administration

- Course web pages online at <http://eee.uci.edu/11f/18056/>
 - Instructor information
 - Course description and contents
 - Course policies and resources
 - Course schedule
 - Homework assignments
 - Course communication
 - Message board (announcements and technical discussion)
 - Email (administrative issues)

EECS22: Advanced C Programming, Lecture 1

(c) 2011 R. Doemer

11

Getting Started

- Obtain an account on the EECS Linux server
 - Your working account in EECS
 - Activation online via EECS web server:
<https://newport.eecs.uci.edu/account.py>
- Log into the server
 - Use a terminal with SSH protocol (secure shell, port 22)
 - Connect to the EECS Linux server
 - `ladera.eecs.uci.edu`
 - Authorize yourself with user name and password
- Work in the Linux system environment
 - Shell prints command prompt, awaiting input
 - Type in system commands
`echo, date, ls, cat, man, more,`
`pwd, mkdir, cd, cp, mv, rm, rmdir, ...`
 - Refer to manual pages for help on commands

EECS22: Advanced C Programming, Lecture 1

(c) 2011 R. Doemer

12

Linux System Environment

- Linux shell commands
 - **echo** print a message
 - **date** print the current date and time
 - **ls** list the contents of the current directory
 - **cat** list the contents of files
 - **more** list the contents of files page by page
 - **pwd** print the path to the current working directory
 - **mkdir** create a new directory
 - **cd** change the current directory
 - **cp** copy a file
 - **mv** rename and/or move a file
 - **rm** remove (delete) a file
 - **rmdir** remove (delete) a directory
 - **man** view manual pages for system commands

Linux System Environment

- Text editing
 - **vi** standard Unix editor
 - **vim** vi-improved (supports syntax highlighting)
 - **pico** easy-to-use text editor
 - **emacs** very powerful editor
 - many others...
- Pick one editor and make yourself comfortable with it!

Review of C Programming

- Categories of programming languages
 - Machine languages (stream of 1's and 0's)
 - Assembly languages (low-level CPU instructions)
 - **High-level languages** (**high-level instructions**)
- Translation of high-level languages
 - Interpreter (translation for each instruction)
 - **Compiler** (**translation once for entire unit**)
 - Hybrid (combination of the above)
- Types of programming languages
 - Functional (e.g. Lisp)
 - **Structured** (e.g. Pascal, **C**, Ada)
 - Object-oriented (e.g. C++, Java, Python)

EECS22: Advanced C Programming, Lecture 1

(c) 2011 R. Doemer

15

History of C

- Evolved from BCPL and B
 - in the 60's and 70's
- Created in 1972 by Dennis Ritchie (Bell Labs)
 - first implementation on DEC PDP-11
 - added concept of *typing* (and other features)
 - development language of UNIX operating system
- “Traditional” C
 - 1978, “*The C Programming Language*”, by Brian W. Kernighan, Dennis M. Ritchie
 - ported to most platforms
- ANSI C
 - standardized in 1989 by ANSI and OSI
 - standard updated in 1999

EECS22: Advanced C Programming, Lecture 1

(c) 2011 R. Doemer

16

The C Programming Language

- What is C?
 - Programming language
 - high-level
 - structured
 - compiled
 - Standard library
 - rich collection of existing functions
- Why C?
 - de-facto standard in software development
 - code is portable to many different platforms
 - supports structured and functional programming
 - easy transition to object-oriented programming
 - C++ / Java
 - freely available for most platforms

EECS22: Advanced C Programming, Lecture 1

(c) 2011 R. Doemer

17

The first C Program

- Program example: `HelloWorld.c`

```
/* HelloWorld.c: our first C program */
/*
/* author: Rainer Doemer
/*
/* modifications:
/* 09/28/04 RD initial version
*/

#include <stdio.h>

/* main function */

int main(void)
{
    printf("Hello World!\n");
    return 0;
}

/* EOF */
```

EECS22: Advanced C Programming, Lecture 1

(c) 2011 R. Doemer

18

The first C Program

- Program comments
 - start with `/*` and end with `*/`
 - are ignored by the compiler
 - should be used to
 - document the program code
 - structure the program code
 - enhance the readability
- `#include` preprocessor directive
 - inserts a header file into the code
- standard header file `<stdio.h>`
 - part of the C standard library
 - contains declarations of standard types and functions for data input and output (e.g. function `printf()`)

```

/* HelloWorld.c: our first C program */
/* author: Rainer Doemer */
/* modifications: */
/* 09/28/04 RD initial version */
#include <stdio.h>
/* main function */
int main(void)
{
    printf("Hello World!\n");
    return 0;
}
/* EOF */

```

The first C Program

- `int main(void)`
 - main function of the C program
 - the program execution starts (and ends) here
 - `main` must return an integer (`int`) value to the operating system at the end of its execution
 - return value of 0 indicates successful completion
 - return value greater than 0 usually indicates an error condition
- function body
 - block of code (definitions and statements)
 - starts with an opening brace (`{`)
 - ends with a closing brace (`}`)
- `printf()` function
 - formatted output (to `stdout`)
- `return` statement
 - ends a function and returns its argument as result

```

...
/* main function */
int main(void)
{
    printf("Hello World!\n");
    return 0;
}
/* EOF */

```

The first C Program

- Program compilation
 - compiler translates the code into an executable program
 - `gcc HelloWorld.c`
 - compiler reads file `HelloWorld.c` and creates file `a.out`
 - options may be specified to direct the compilation
 - `-o HelloWorld` specifies output file name
 - `-ansi -Wall` specifies ANSI code with all warnings
- Program execution
 - use the generated executable as command
 - `HelloWorld`
 - the operating system loads the program (loader), then executes its instructions (program execution), and finally resumes when the program has terminated

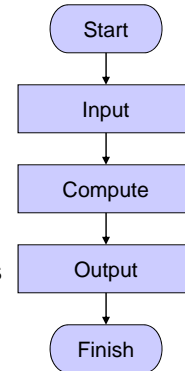
The first C Program

- Example session: HelloWorld.c

```
login as: doemer
doemer@ladera.eecs.uci.edu's password:
Last login: Tue Sep 20 16:26:18 2011 from beta.eecs.uci.edu
doemer@ladera.eecs.uci.edu:1 > cd eecs22/lecture1/
doemer@ladera.eecs.uci.edu:2 > vi HelloWorld.c
doemer@ladera.eecs.uci.edu:3 > ls
Addition.c HelloWorld.c
doemer@ladera.eecs.uci.edu:4 > ls -l
total 2
-r----- 1 doemer named 702 Oct 26 2009 Addition.c
-r----- 1 doemer named 263 Oct 26 2009 HelloWorld.c
doemer@ladera.eecs.uci.edu:5 > gcc HelloWorld.c
doemer@ladera.eecs.uci.edu:6 > ls
Addition.c a.out* HelloWorld.c
doemer@ladera.eecs.uci.edu:7 > a.out
Hello World!
doemer@ladera.eecs.uci.edu:8 > gcc -Wall -ansi HelloWorld.c -o
HelloWorld
doemer@ladera.eecs.uci.edu:9 > HelloWorld
Hello World!
doemer@ladera.eecs.uci.edu:10 > exit
```

General Program Structure

- Initialization section
 - Definition of variables (storage elements)
 - Name, type, and initial value
- Input section
 - read values from input devices into variables
 - standard input functions
- Computation section
 - perform the necessary computation on variables
 - assignment statements
- Output section
 - write results from variables to output devices
 - standard output functions
- Exit section
 - clean up and exit



EECS22: Advanced C Programming, Lecture 1

(c) 2011 R. Doemer

23

General Program Structure

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

```

/* Addition.c: adding two integer numbers */
/* */
/* author: Rainer Doemer */
/* */
/* modifications: */
/* 09/30/04 RD initial version */

#include <stdio.h>

/* main function */

int main(void)
{
    /* variable definitions */
    int i1 = 0; /* first integer */
    int i2 = 0; /* second integer */
    int sum; /* result */
    ...
  
```

EECS22: Advanced C Programming, Lecture 1

(c) 2011 R. Doemer

24

General Program Structure

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

```

...
/* input section */
printf("Please enter an integer:   ");
scanf("%d", &i1);
printf("Please enter another integer: ");
scanf("%d", &i2);

/* computation section */
sum = i1 + i2;

/* output section */
printf("The sum of %d and %d is %d.\n", i1, i2, sum);

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

/* EOF */

```

EECS22: Advanced C Programming, Lecture 1

(c) 2011 R. Doemer

25

General Program Structure

- Variable definition and initialization

```

/* variable definitions */
int i1 = 0;      /* first integer */
int i2 = 0;      /* second integer */
int sum;         /* result */

```

- Variable type: `int`
 - integer type, stores whole numbers (e.g. -5, 0, 42)
 - many other types exist (`float`, `double`, `char`, ...)
- Variable name: `i1`, `i2`, `sum`
 - valid identifier, i.e. name composed of letters, digits
 - variable name should be descriptive
- Initializer: `= 0`
 - specifies the initial value of the variable
 - optional (if omitted, initial value is undefined)

EECS22: Advanced C Programming, Lecture 1

(c) 2011 R. Doemer

26

General Program Structure

- Data input using `scanf()` function

```
/* input section */
printf("Please enter an integer:  ");
scanf("%d", &i1);
```

- part of standard I/O library
 - declared in header file `stdio.h`
- reads data from the standard input stream `stdin`
 - `stdin` usually means the keyboard
- converts input data according to format string
 - `"%d"` indicates that a decimal integer value is expected
- stores result in specified location
 - `&i1` indicates to store at the *address of variable i1*

General Program Structure

- Computation using assignment statements

```
/* computation section */
sum = i1 + i2;
```

- Operator `=` specifies an assignment
 - value of the right-hand side (`i1 + i2`) is assigned to the left-hand side (`sum`)
 - left-hand side is usually a variable
 - right-hand side is a simple or complex expression
- Operator `+` specifies addition
 - left and right arguments are added
 - result is the sum of the two arguments
- Many other operators exist
 - For example, `-`, `*`, `/`, `%`, `<`, `>`, `==`, `^`, `&`, `|`, ...

General Program Structure

- Data output using `printf()` function

```
/* output section */
printf("The sum of %d and %d is %d.\n", i1, i2, sum);
```

- part of standard I/O library
 - declared in header file `stdio.h`
- writes data to the standard output stream `stdout`
 - `stdout` usually means the monitor
- converts output data according to format string
 - standard text is copied verbatim to the output
 - `%d` is replaced with a decimal integer value
- takes values from specified arguments
 - `i1` indicates to use the value of the variable `i1`

General Program Structure

- Example session: `Addition.c`

```
% vi Addition.c
% ls -l
-rw----- 1 doemer faculty 702 Sep 30 14:17 Addition.c
% gcc -Wall -ansi Addition.c -o Addition
% ls -l
-rwx----- 1 doemer faculty 6628 Sep 30 16:44 Addition*
-rw----- 1 doemer faculty 702 Sep 30 14:17 Addition.c
% Addition
Please enter an integer: 27
Please enter another integer: 15
The sum of 27 and 15 is 42.
% Addition
Please enter an integer: 123
Please enter another integer: -456
The sum of 123 and -456 is -333.
%
```