

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

- Programming Courses in EECS
- Course Administration
 - 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`

Programming Courses in EECS

- Introductory Programming
 - EECS 10: uses C programming language (for EE)
 - EECS 12: uses Python programming language (for CpE)
- Programming from the Ground Up
 - EECS 20: starts with Assembly language (on bare CPU), then introduces C programming language
- Advanced Programming Courses
 - EECS 22: *“Advanced C Programming”* (in ANSI C)
 - EECS 22L: *“Software Engineering Project in C”* (ANSI C/C++)
- Object-Oriented Programming
 - EECS 40: introduces objects and classes, hierarchy, and higher object-oriented programming concepts using Java

EECS22: Advanced C Programming, Lecture 1

(c) 2014 R. Doemer

3

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.
 - Prerequisite: EECS 10 or EECS 20
 - (Design Units: 1)

EECS22: Advanced C Programming, Lecture 1

(c) 2014 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, variable-length arguments, ...
 - Libraries, Makefile, ...
- Tools
 - C preprocessor, compiler, and linker
 - Debugger ‘gdb’ and ‘ddd’
 - Dynamic memory allocation checker ‘valgrind’

EECS22: Advanced C Programming, Lecture 1

(c) 2014 R. Doemer

5

EECS 22: Advanced C Programming

- Course Topics
 - 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, ...
 - Function pointers and callback functions
 - 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) 2014 R. Doemer

6

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) 2014 R. Doemer

7

Course Administration

- Course web pages online at <http://eee.uci.edu/14f/18030/>
 - 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) 2014 R. Doemer

8

Getting Started

- Obtain an account on the EECS Linux servers
 - Activation online via EECS web server:
`https://newport.eecs.uci.edu/account.py`
 - Existing EECS accounts can be used
 - (contact OIT for password reset, if forgotten)
- Login to the server
 - Use a terminal with SSH protocol (secure shell, port 22)
 - Connect to one of the EECS Linux servers
 - `crystalcove.eecs.uci.edu`
 - `zuma.eecs.uci.edu`
 - Authorize yourself with UCInetID and EECS password
- Work in the Linux system environment
 - Shell prints command prompt, awaiting input
 - Use system commands: `ls`, `pwd`, `cd`, `cp`, `rm`, `mkdir`, ...
 - Refer to manual pages (`man`) for help on commands

EECS22: Advanced C Programming, Lecture 1

(c) 2014 R. Doemer

9

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

EECS22: Advanced C Programming, Lecture 1

(c) 2014 R. Doemer

10

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)

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 and 2011

EECS22: Advanced C Programming, Lecture 1

(c) 2014 R. Doemer

13

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) 2014 R. Doemer

14

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) 2014 R. Doemer

15

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 */

```

EECS22: Advanced C Programming, Lecture 1

(c) 2014 R. Doemer

16

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

```
% mkdir HelloWorld
% cd HelloWorld
% ls
% vi HelloWorld.c
% ls
HelloWorld.c
% ls -l
-rw-r--r--  1 doemer  faculty    263 Sep 28 22:11 HelloWorld.c
% gcc HelloWorld.c
% ls -l
-rw-r--r--  1 doemer  faculty    263 Sep 28 22:11 HelloWorld.c
-rwxr-xr-x  1 doemer  faculty   6352 Sep 28 22:12 a.out*
% a.out
Hello World!
% gcc -Wall -ansi HelloWorld.c -o HelloWorld
% ls -l
-rwxr-xr-x  1 doemer  faculty   6356 Sep 28 22:17 HelloWorld*
-rw-r--r--  1 doemer  faculty    263 Sep 28 22:17 HelloWorld.c
-rwxr-xr-x  1 doemer  faculty   6352 Sep 28 22:12 a.out*
% HelloWorld
Hello World!
```

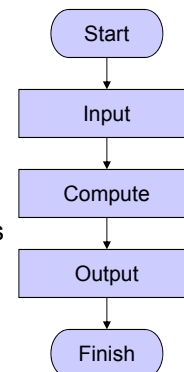
EECS22: Advanced C Programming, Lecture 1

(c) 2014 R. Doemer

19

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) 2014 R. Doemer

20

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 */
    ...

```

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 */

```

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**
 - 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)

General Program Structure

- Data input using **scanf ()** function

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

- Function **scanf ()** is defined in 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);
```

- Function `printf()` is defined in 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
 - text ("The sum...") is copied verbatim to the output
 - `"%d"` is replaced with a decimal integer value
- ... takes values from specified arguments (in order)
 - `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.
%
```