

EECS 10: Computational Methods in Electrical and Computer Engineering

Lecture 1

Rainer Dömer

doemer@uci.edu

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

Lecture 1.1: Overview

- Introduction
 - Course overview
- Introduction to Computers
 - What is a computer?
 - What is programming?
- Course administration
 - Course web pages

Introduction

- Course Contents
 - Introduction to computers
 - Introduction to structured programming
 - C, a high-level structured programming language
 - Binary data representation
 - Introduction to algorithm efficiency
 - Solving engineering problems
 - Applications of structured programming
 - Hands-on experience
 - Laboratory and discussion sessions

Introduction to Computers

- What is a computer?
 - Digital device capable of executing programs
 - performing computations
 - making logical decisions
- What is a program?
 - Set of instructions which process data
 - input data (e.g. from keyboard, mouse, disk)
 - output data (e.g. to monitor, printer, disk)
- What is programming?
 - Creation of computer programs by use of a programming language

Introduction to 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 all code)
 - 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)

EECS10: Computational Methods in ECE, Lecture 1

(c) 2015 R. Doemer

5

Course Administration

- Course web pages online at <http://eee.uci.edu/15y/18021/>
 - 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)

EECS10: Computational Methods in ECE, Lecture 1

(c) 2015 R. Doemer

6

Lecture 1.2: Overview

- Getting started
 - Obtain your UCI netID
 - Obtain an account on the EECS servers
 - Log into the server
- Linux system environment
 - System commands
 - Text editing

Getting Started

- Obtain your UCI netID
 - Your unique ID at UCI
 - Activation online at OIT (NACS) web pages:

`http://activate.uci.edu/activate/menu.html`
- Obtain an account on the EECS servers
 - Your working account in EECS
 - Activation online at EECS web pages:

`https://newport.eecs.uci.edu/account.py`

Getting Started

- Log into the server
 - Use a terminal with SSH protocol (secure shell)
 - Connect to the EECS Linux server
 - `crystalcove.eecs.uci.edu`
 - `zuma.eecs.uci.edu`
 - Authorize yourself with user name and password
- Work in the Linux system environment
 - Linux 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

EECS10: Computational Methods in ECE, Lecture 1

(c) 2015 R. Doemer

9

Linux System Environment

- Linux system 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

EECS10: Computational Methods in ECE, Lecture 1

(c) 2015 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!

Linux System Environment

- Example session (1/4):

```
login as: doemer
Password:
Last login: Mon Oct  1 08:20:09 2007 from beta.eecs.uci.e
...
If this system is busy, consider a less loaded one below:
vivian.eecs.uci up 30 days, 18:00, load average: 0.00, 0.00, 0.01
malibu.eecs.uci up 2826 days, 21:06, load average: 0.00, 0.00, 0.01
newport.eecs.uc up 23 days, 23:29, load average: 0.00, 0.00, 0.02
east.eecs.uci.e up 12 days, 4:56, load average: 1.46, 1.41, 1.68
% date
Mon Oct  1 08:24:47 PDT 2007
% echo "Hello EECS10!"
Hello EECS10!
% ls
eecs10/          Mail/           tmp/
% pwd
/users/faculty/doemer
% mkdir homework
% ls
eecs10/          homework/      Mail/          tmp/
...
```

Linux System Environment

- Example session (2/4):

```

...
% cd homework
% pwd
/users/faculty/doemer/homework
% ls
% mkdir hw1
% ls
hw1/
% cd hw1
% ls
% vi program.c
% ls
program.c
doemer@vivian% ls -l
total 2
-rw----- 1 doemer smmsp      51 Oct  1 08:32 program.c
% more program.c
This is my new program file.
I don't know C yet...
...

```

EECS10: Computational Methods in ECE, Lecture 1

(c) 2015 R. Doemer

13

Linux System Environment

- Example session (3/4):

```

...
% cp program.c mybackup.c
% ls
mybackup.c  program.c
% ls -l
-rw----- 1 doemer smmsp      51 Oct  1 08:34 mybackup.c
-rw----- 1 doemer smmsp      51 Oct  1 08:32 program.c
% cd ..
% pwd
/users/faculty/doemer/homework
% ls
hw1/
% /ecelib/bin/turnin
=====
EECS 10 Fall 2007:
Assignment "hw1" submission for doemer
Due date: Mon Oct  8 11:59:59 2007
=====
...

```

EECS10: Computational Methods in ECE, Lecture 1

(c) 2015 R. Doemer

14

Linux System Environment

- Example session (4/4):

```
...
Submit program.c [yes, no]? y
Cannot read file program.c
Submit mybackup.c [yes, no]? n
=====
Summary:
=====
You just submitted file(s):
  program.c
You have not submitted file(s):
  mybackup.c
doemer@vivian% ~eecs10/bin/listfiles.py
=====
EECS 10 Fall 2007: "hw1" listing for doemer
=====
Files submitted for assignment "hw1":
program.c
% logout
```

Lecture 1.3: Overview

- Introduction to Programming in C
 - History of C
 - Introduction to C
- Our first C Program
 - Example `HelloWorld.c`
 - Structure of a C program
 - `printf()` function
 - Program compilation and execution
 - String constants

Introduction to 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 all code**)
 - 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)

EECS10: Computational Methods in ECE, Lecture 1

(c) 2015 R. Doemer

17

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

EECS10: Computational Methods in ECE, Lecture 1

(c) 2015 R. Doemer

18

Introduction to C

- 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

Our 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 */
```

Our 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 */

```

Our 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 */

```

Our 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

Our 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!
```

Our first C Program

- Character string constants: "Strings"
 - start and end with a double quote character ("")
 - may not extend over a single line
 - subsequent string constants are combined
 - text formatting using escape sequences
 - `\n` new line
 - `\t` horizontal tab
 - `\r` carriage return
 - `\b` back space
 - `\a` alert / bell
 - `\\` backslash character
 - `\"` double quote character
- Experiments with the `HelloWorld` program...