456: Network-Centric Programming

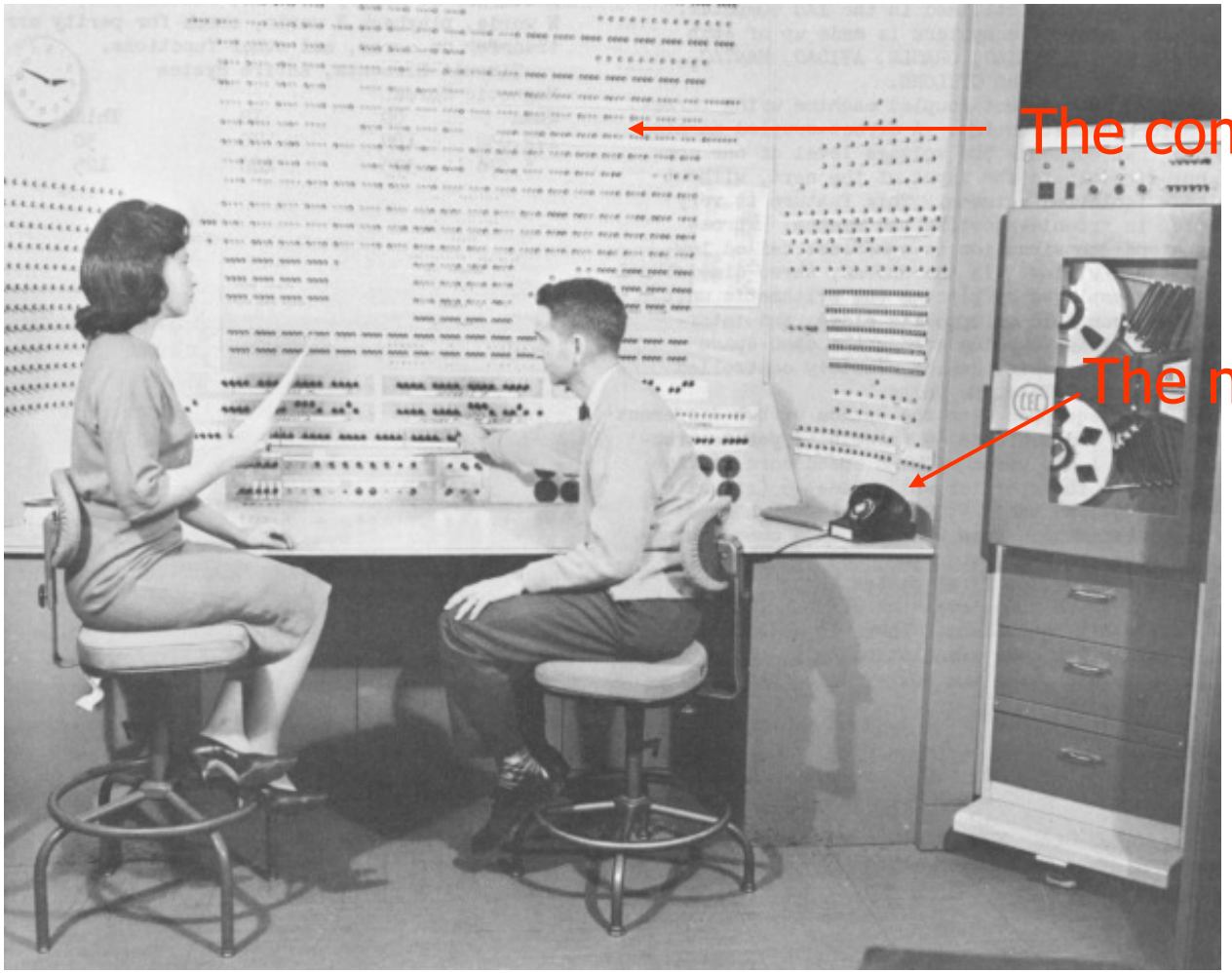
Yingying (Jennifer) Chen

Web: https://www.winlab.rutgers.edu/~yychen/
Email: yingche@scarletmail.rutgers.edu

Department of Electrical and Computer Engineering
Rutgers University

Why Network-Centric Programming?

Early Days: Computer and Network Separated



BRLESC US Army 1962

Trend 1: The disappearing computer

System/360 IBM Archives

Apple Macintosh 1984

Palm Pilot 1996

Trend 2: Convergence

- Networks and computers have merged

Trend 2: Convergence

Network Programming

- Most “computers” are networked
- It’s hard to find examples of software that is not affected by networks

Connected Home

Chumby

Weather clock

Raspberry Pi

Intel Galileo (Gen 2)

Intel Edison

WINLAB: ORBIT Wireless Network

Vehicle-to-Vehicle Communications (V2V)

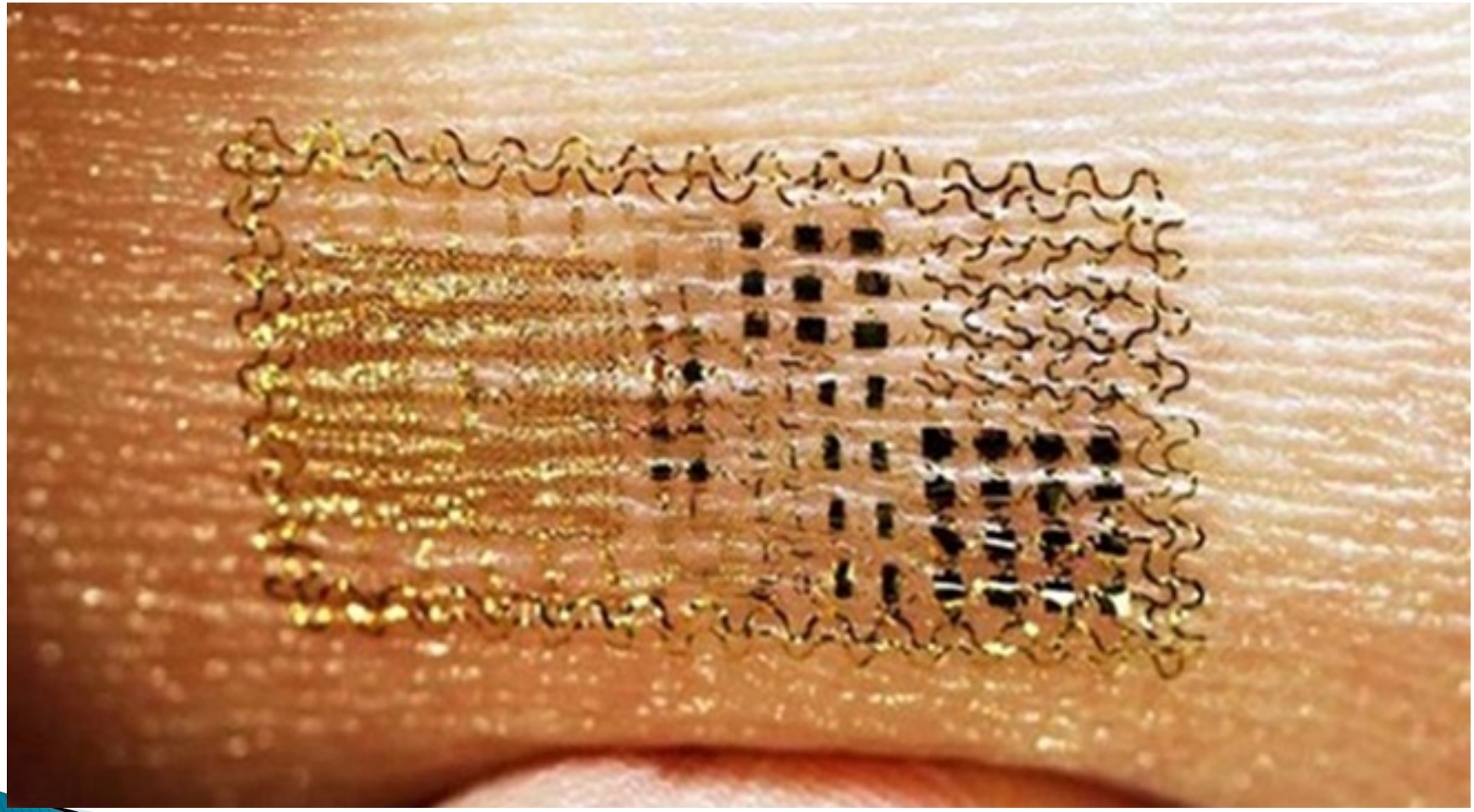
GPS/V2V Stalled vehicle warning

GPS/V2V Blind spot warning

Source: GM Press Release 2005

F015 Concept Car

E-Tattoo ?



The Course …

This Course is a practical course about writing *real* small-medium network-centric programs

Advanced Programming

- Writing network-centric programs from scratch
  - Using operating system services effectively
  - Using development tools
  - Good development style
  - Performance tuning
  - Enhancing security

Topics

- Files, IO, Network Communication
- Socket Network Programming
- Network Server Design
  - Multi-threaded programs and synchronization
- Secure Programming
- Profiling and Performance Analysis
- Remote Procedure calls

How does it compare to other classes?

- Operating Systems – how do you write an operating system to make application programming easier
- Network-Centric Programming – writing real small-medium-sized network-centric programs
- Telecommunication Networks – how do network protocols work
- Software Engineering – designing large team-based projects

Course Work

- Four (major) Individual Programming Projects
  - Approx. one every 3 weeks
  - 10-14 days time to implement
  - All work must be done individually, unless otherwise stated
- In-class programming midterm exam
- Final Exam or Final Course Project

- Reading assignments
- Small homework assignments possible ungraded, but helpful to prepare for exams and projects

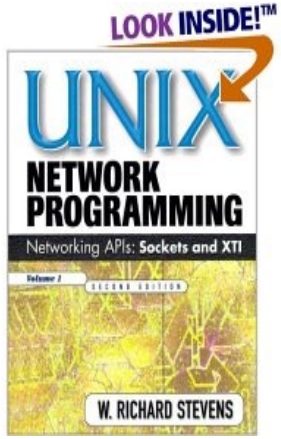
Lecture Style

- Analyze and discuss small sample programs
- In-class lab sessions

Recommended Textbook

- UNIX Network Programming Vol 1. (The Sockets Networking API),
  Stevens, Fenner, Rudoff, 3rd Edition, Addision Wesley

- Computer Systems: A Programmer's Perspective, Second Edition, Randal E. Bryant and David R. O'Hallaron, Prentice Hall, 2011



Also Useful

- Advanced Programming in the UNIX environment,
  Stevens, Rago, 3rd Edition Addison-Wesley.
- Advanced Linux Programming,
  Mark Mitchell, Jeffrey Oldham, and Alex Samuel
  New Riders Publishing

Grading

- Projects 45%
- Mid term 20%
- Final exam or final course project 20%
- Class participation and in-class lab sessions 15%
- Optional homework (bonus points up to 15 points)

Individual Projects

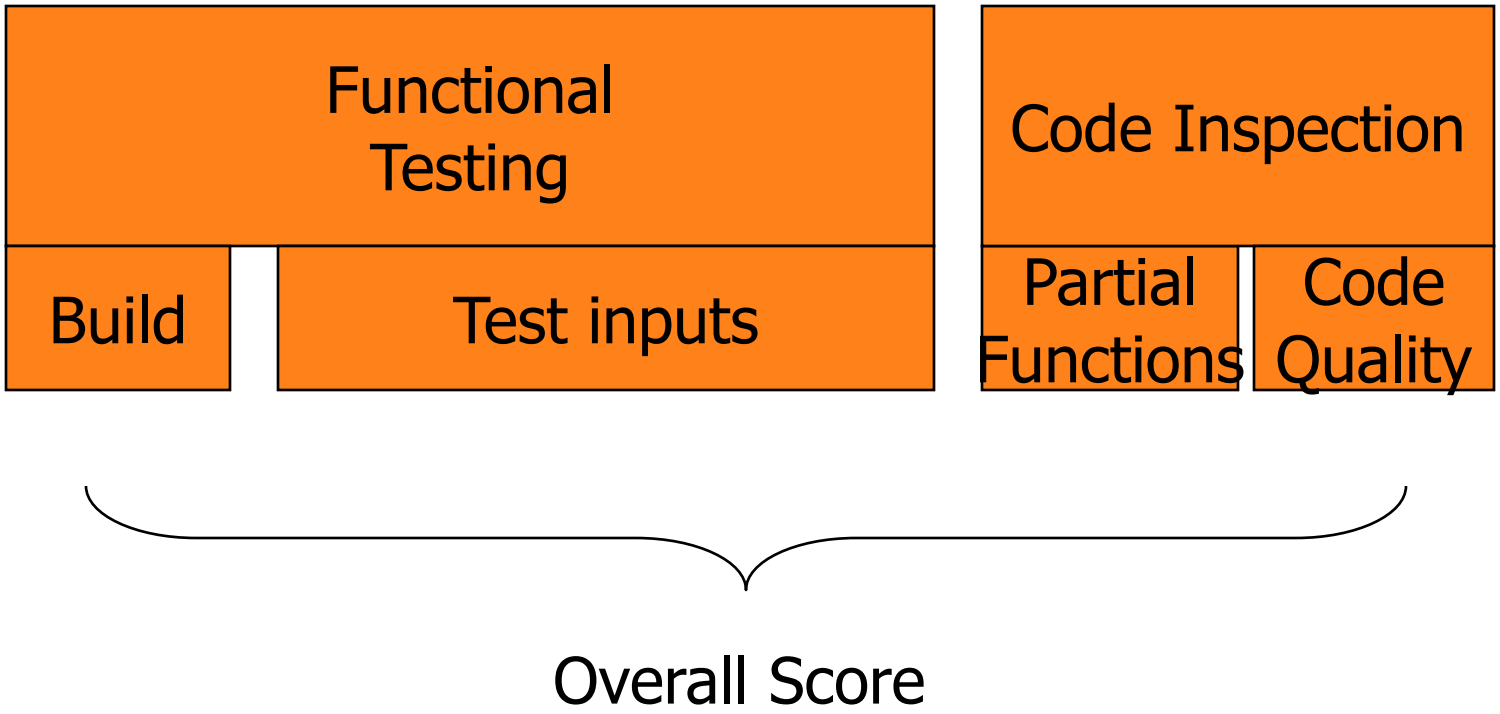
Course Projects

- Warming up, basic C++ programming
- Minimal webserver/proxy
- A (T)FTP server / client
  - Use threads and process control to improve performance
- Buffer overflow vulnerabilities (worms, viruses) in a network server

Linux

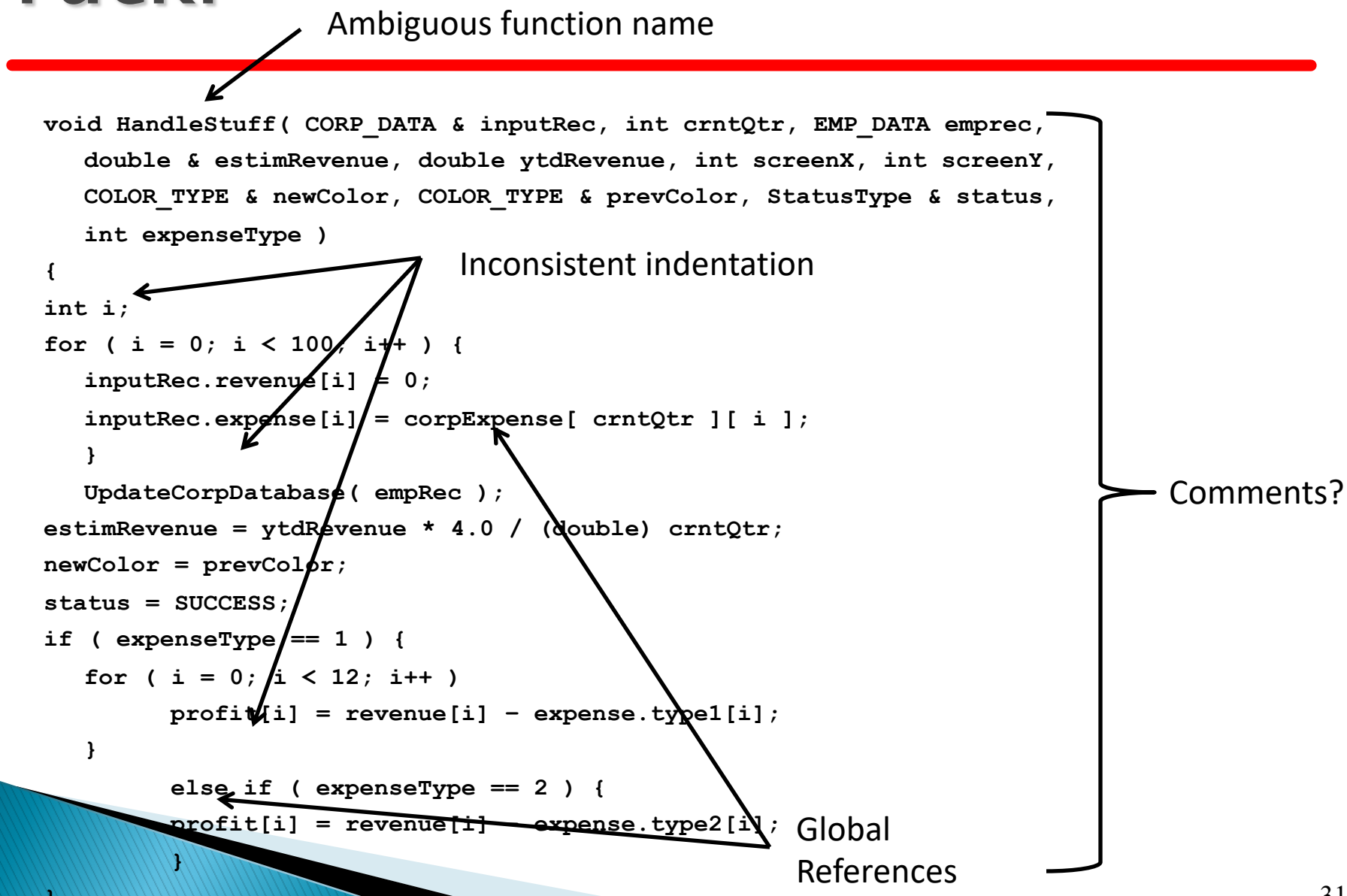
- **Your projects will be graded and must run on the EE 103/105 lab machines**
- It's possible to work on other machines to some extent but expect some incompatibilities and test your code in EE 103/105 before submission
  - Macs, Cygwin, other linux versions
  - DSV / EIT Lab (Engineering School)

Grading Projects



What does “Code Quality” mean?

Yuck!


Ambiguous function name
void HandleStuff(CORP_DATA & inputRec, int crntQtr, EMP_DATA emprec,
double & estimRevenue, double ytdRevenue, int screenX, int screenY,
COLOR_TYPE & newColor, COLOR_TYPE & prevColor, StatusType & status,
int expenseType)
{
int i;
for (i = 0; i < 100; i++) {
inputRec.revenue[i] = 0;
inputRec.expense[i] = corpExpense[crntQtr][i];
}
UpdateCorpDatabase(empRec);
estimRevenue = ytdRevenue * 4.0 / (double) crntQtr;
newColor = prevColor;
status = SUCCESS;
if (expenseType == 1) {
for (i = 0; i < 12; i++)
profit[i] = revenue[i] - expense.type1[i];
}
else if (expenseType == 2) {
profit[i] = revenue[i] - expense.type2[i];
}
}
Inconsistent indentation
Comments?
Global References

Some other problems…

- *11* parameters?! Yikes!
- Is there a single purpose to this routine?
- No defense against bad data (potential divide-by-zero error lurking in code)
- Some parameters passed into function are not used

… **Surely we can do better!**

Course Policies

Collaboration?

- Ungraded homework
- Graded Individual Projects
  - Discussion of general approach is allowed
  - But need to write your own code
  - Do not share code
  - Do not grab pieces of code from the web
- http://academicintegrity.rutgers.edu/
- Automatic code inspection for similarity checking will be conducted

Late Policy

- 20% penalty per *beginning* 24hr period.
- No assignments will be accepted after 72hrs.
- No exceptions.

Prerequisites

- Programming Methodology I/II
- C/C++ programming skills and familiarity with basic development tools: editors, compilers

Contacting us

- Instructor: Yingying (Jennifer) Chen
  - Office Hours: 4-5PM Wednesday
    https://rutgers.webex.com/meet/yingche
  - Email: yingche@scarletmail.rutgers.edu

- TA: Bin Hu
  - Office Hours: 4-5PM Wednesday
  - Email: bh439@scarletmail.rutgers.edu

Course Schedule and Communications

- Check the Canvas course page frequently
- Slides are posted in Modules
- If you are registered at the course, you should have access to the course site

Why C? Why Linux?

- To strike a balance between
  - Immediate practical relevance
  - Durability of knowledge
- C and UNIX were developed in the 60's and 70's and have withstood the test of time
- Recent revival through Linux

Isn't it all mobile apps and scripting these days?

- Check the PHP libraries – many similarities to C/UNIX libraries
  - Strcmp, fork, …
- Android runs on Linux stack

iPhones runs UNIX

The Linux Programming Environment and Stdio

Roadmap: The Basics

- Warm up
  - Linux programming environment, makefiles, error handling
- I/O
  - Files, directories, file descriptors
- Processes
  - Creating new processes, passing information
- Sockets
  - And their relation to file descriptors

First Reading – How to write a program on linux

- Editors
  - Mousepad
  - vi
  - Emacs
- Using gcc to compile
- Using make

Examples

- Reciprocal.
- Odd/even number
  - determine whether a number is odd or even
- Factors
  - show all the factors of a number

Reciprocal

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

double reciprocal (int i){
  assert(i !=0); //verify whether i is zero
  return 1.0/i;
}

int main(int argc, char **argv)
{
  int i;
  i = atoi(argv[1]); //convert string to integer
  printf("The reciprocal of %d is %g\n",i,reciprocal(i));
  return 0;
}
```

```
% make
% ./main 2
```

Odd/even number

```
//Show a number is odd or even.
#include <stdio.h>

int main()
{
    int n;
    printf("Enter an integer: ");
    scanf("%d",&n);
    if ( n % 2 == 0)
        printf("%d is even.\n ",n);
    else
        printf("%d is odd.\n ",n);
    return 0;
}
```

```
% make
% ./main
```

Factors

```
//Display all Factors of a Number
#include <stdio.h>

int main()
{
    int n, i;
    printf("Enter a positive integer: ");
    scanf("%d",&n);
    printf("Factors of %d are: ", n);
    for(i = 1; i <= n; ++i)
    {
        if(n % i == 0)
            printf("%d ",i);
    }
    return 0;
}
```

```
% make
% ./main
```

A Simple Makefile Tutorial

hellomake.c

```
#include <hellomake.h>

int main() {
  // call a function in another file
  myPrintHelloMake();

  return(0);
}
```

hellofunc.c

```
#include <stdio.h>
#include <hellomake.h>

void myPrintHelloMake(void) {

  printf("Hello makefiles!\n");

  return;
}
```

hellomake.h

```


/*
example include file
*/

void myPrintHelloMake(void);
```

Normally, you would compile this collection of code by executing the following command:
gcc -o hellomake hellomake.c hellofunc.c -I.

The -I. is included so that gcc will look in the current directory (.) for the include file hellomake.h.

A Simple Makefile Tutorial

Makefile 1

```
hellomake: hellomake.c hellofunc.c
        gcc -o hellomake hellomake.c hellofunc.c -I.
```

- Note that *make* with no arguments executes the first rule in the file.
- Furthermore, by putting the list of files on which the command depends on the first line after the :, make knows that the rule hellomake needs to be executed if any of those files change.

A Simple Makefile Tutorial

Makefile 2

```
CC=gcc
CFLAGS=-I.

hellomake: hellomake.o hellofunc.o
	$(CC) -o hellomake hellomake.o hellofunc.o
```

- The macro CC is the C compiler to use, and CFLAGS is the list of flags to pass to the compilation command.
- By putting the object files--hellomake.o and hellofunc.o--in the dependency list and in the rule, *make* knows it must first compile the .c versions individually, and then build the executable hellomake.

A Simple Makefile Tutorial

Makefile 3

```
CC=gcc
CFLAGS=-I.

hellomake: hellomake.o hellofunc.o
	$(CC) -o hellomake hellomake.o hellofunc.o
clean:
	rm *.o hellomake
```

- Make clean will delete all the object files and executable file

Object File vs. Executable File

OBJECT FILE	EXECUTABLE FILE
A file that contains an object code that has relocatable format machine code, which is not directly executable	A file that can be directly executed by the computer and is capable of performing the indicated tasks according to the encoded instructions
An intermediate file	A final file
A compiler converts the source code to an object file	A linker links the object files with the system library and combines the object files together to create an executable file
Cannot be directly executed by the CPU	Can be directly executed by the CPU

Visit www.PEDIAA.com

Reading Homework

- Advanced Linux Programming book
  - Reading Chapter 1
  - Following all the examples in Chapter 1 to get familiar with the computing environment