Introduction to Systems Programming
(234122)

http://webcourse.tecnion.ac.il/234122

Lecture Slides – March 2017

Yechiel Kimchi

With thanks to Yossi Gil,
Eliezer Kantorowitz, Ayal Itzkoviz,
Omer Strulovich, Ron Rubinstein
Course Objectives

• Learn to develop large-scale *software* (not programs).
  – This is *Software-Engineering*

• Extend knowledge of C - mostly extended C89 (ANSI), with a few new capabilities of C99 (no C11).

• Learn how to work in Unix environment.

• Learn about object-oriented programming.

• Meet C++ and its (not so) basic capabilities.
What This Course Is Not

- A C/C++ *language* course.
- A data structures course.
- A programming tricks / hacks course.

**Administration at recitation**
## Comparison

<table>
<thead>
<tr>
<th>Program</th>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Typically 10-1000 lines.</td>
<td>Typically at least 10,000 lines.</td>
</tr>
<tr>
<td>Developed by individual.</td>
<td>Developed by a team.</td>
</tr>
<tr>
<td>Used by programmer.</td>
<td>Used by “customer”.</td>
</tr>
<tr>
<td><strong>Might</strong> run first time.</td>
<td><strong>Never</strong> runs first time.</td>
</tr>
<tr>
<td>Can be coded immediately.</td>
<td>Requires analysis/design.</td>
</tr>
<tr>
<td>Requires debugging.</td>
<td>Requires long-term maintenance.</td>
</tr>
<tr>
<td>Easy !!</td>
<td>Difficult.</td>
</tr>
</tbody>
</table>

**Example:** A sorting program.  
**Example:** An accounting system.
An Abstract Discussion about Software Quality
Features that Quality Software Has

In this course, most qualities will be intuitive(*)

- **Correct**
  - Meets requirements.

- **Useful**
  - Meets customer expectations

- **Robust**
  - Resistant to user/environmental errors (fault tolerant)
  - Easily modified/enhanced

- **Friendly**
  - “Easy” to learn and use
  - Human engineering

- **Efficient**

Here: Mostly functional requirements

This will be done almost formally

(*) Can be understood after Inro2cs
Software is built of parts that are built of parts, that are built of parts, that are built of parts, that are built of parts, that are built of parts, that are built of parts, ...

Software is fractal except that it’s finite
Quality Software has Abstract Properties

The parts are carefully built and connected in order to provide the following properties:

- **Independence**
  - Otherwise, BUGs leak from one part to another
- **Separation**
  - Necessary for independence
  - Small scale “independence” (examples in 2min)
- **Controlled communication (Interface)**
  - Supports independence and separation
- **Simplicity**
  - Helps in reducing bug count

What structural rules are able to provide these properties?
A Few Practical Rules that Help in Achieving the Abstract Goals

- No modifiable global objects
- Minimal side-effects (of functions, expressions)
- Each function/object has a single task
- Explicit pre/post-conditions (controlled interface)
- Functions are short
- Functions structure is simple

We shall dwell into them later
The Single Responsibility Principle

```c
int main()
{
    double salaries[EMPLOYEES_NUM], sum = 0.0, average;
    int i, above_average = 0;
    for(i = 0; i < EMPLOYEES_NUM; ++i) {
        scanf("%lf", &salaries[i]);
        sum += salaries[i];
    }
    average = sum/EMPLOYEES_NUM;
}
```

```c
int main()
{
    double salaries[EMPLOYEES_NUM], sum = 0.0;
    for (int i = 0; i < EMPLOYEES_NUM; ++i)
    {
        scanf("%lf", &salaries[i]); // Must test success!

        for (int i = 0; i < EMPLOYEES_NUM; ++i)
            sum += salaries[i];

    double average = sum/EMPLOYEES_NUM;
```
The Single Responsibility Principle

```c
int main(void)
{
    double sal[EMPLOYEES_NUM], avg;
    int above_avg;

    read_array(sal, EMPLOYEES_NUM);
    avg = sum_array(sal, EMPLOYEES_NUM) / EMPLOYEES_NUM;
}
```

```c
int main(void)
{
    double slries[NUM];

    ArrayRead(slries, NUM); // Input is tested inside!
    double avrg = ArraySum(slries, NUM) / NUM;
}
```
Separation to the Rescue

```c
int main(void)
{
    double salaries[NUM], sum = 0.0;

    for (int i = 0; i < NUM; ++i)
        scanf("%lf", &salaries[i]); // Must test success!

    for (int i = 0; i < NUM; ++i)
        sum += salaries[i];

    double average = sum/NUM;
}
```

```c
int main()
{
    double salaries[NUM], sum = 0.0;

    for (int i = 0; i < NUM; ++i)
        scanf("%lf", &salaries[i]); // Must test success!

    for (int i = 0; i < NUM; ++i)
        sum += salaries[i];

    double average = sum/NUM;
}
```

Structure echoes functions before they are introduced.
Visualizing Separation

**Example:** If a function contains two loops, it is almost impossible to test one of them separated from the other.

What is yet another important benefit?
Why OOP?

Good Software needs to be:

- Easy to create and easy to maintain
- Both require easy to understand

For that it should be built using:

- Abstractions, e.g:
  - Generalizations
  - No duplications
- Encapsulation, e.g:
  - Information hiding
  - Separation of concerns

OOP achieves those in an intuitive way
Why Unix?

• Multi-user/multi-tasking operating system.
• De-facto standard across multiple platforms.
• Powerful procedural user-interface.
• Easy to learn.

• Linux is taking an increasing share as the chosen OS
  – Initially on PCs and alike, and later a bigger share on servers
  – Linux is a variant of UNIX, which supports GUI interface
  – Linux is open-source, mostly free, usually under GNU GPL license

See a few explanations and examples below
Why C / C++ ?

• De-facto industrial standard.
• “Mother tongue” of Unix systems.
• Structured language.
• C++ is object-oriented (actually, multi-paradigm) syntactic extension of C.

Search: Bad C++ books

Every book on C or C++, that I have seen in Hebrew, contained Horrible Errors.
It’s your call

Yechiel Kimchi
Bibliography  (very basic for C++ and OOP)

C:  
- The C Programming Language (2nd Edition, ANSI-C)  
  B. Kernighan and D. Ritchie,  
  Prentice-Hall,  
  1988
- C Programming FAQs  
  S. Summit,  
  Addison-Wesley Longman,  
  1996
- The Joy of C  
  (3rd Edition)  
  L. Miller and A. Quilici,  
  Wiley,  
  1997
- C A Reference Manual  
  (5th Edition)  
  S. Harbison and G. Steele,  
  Prentice Hall,  
  2002

C++:  
  B. Stroustrup,  
  Addison-Wesley Longman,  
  2011
- C++ Primer  
  (5th Edition)  
  S. Lippman and J. Lajoie,  
  Addison-Wesley Longman,  
  2011
- Effective C++  
  (3rd Edition)  
  S. Meyers,  
  Addison-Wesley Longman,  
  2005
- Effective Modern C++  
  S. Meyers,  
  O’Reilly,  
  2015
- Industrial Strength C++  
  M. Henricson and E. Nyquist,  
  Prentice Hall,  
  1997
- C++ FAQs  
  (2nd Edition)  
  M. Cline, G. Lomow, M. Girou  
  Addison-Wesley Longman,  
  1999

UNIX:  
- Unix in a Nutshell  
  (4th Edition)  
  A. Robbins,  
  O'Reilly,  
  2005

OOP:  
- Object-Oriented Software Construction  
  (2nd Edition)  
  B. Meyer,  
  Prentice Hall,  
  1997
UNIX in a Pill

• UNIX began as a textual OS (pre-GUI)
• For the last few decades it supplies GUI environment
  – Drag and drop
  – Highlight/copy/paste
  – Menus and click to activate/execute
• Yet, UNIX full power is still textual
  – Even in other window systems keyboard shortcuts, like Ctrl-C/Ctrl-V, are easier/faster than their pull-down menu counterparts – unfortunately, not all menu entries have a keyboard shortcut
• A UNIX window is in fact a terminal, operated textually
  – Create directories (folders) and move from one to another
  – Copy/move/remove/execute files/programs
  – Modify your environment
  – Get help
  – Much more
UNIX in a Pill (Cont.)

Examples:

- `ls` - View directory contents
- `cd` - Move around directories
- `man` - View a manual for a command
- `gcc` - Compile a C program
- `diff` - Find differences between two text files
- `wc` - Count characters/words/lines in a text file

Most commands have many options, so you can use `alias` - Create a short name for a long command, like

```
alias a=alias
```
```
a gcc='gcc -ansi -pedantic -Wall'
a g99='\gcc -std=c99 -pedantic -Wall'
```
UNIX in a Pill (Cont.)

The Shell:

• Communication with the UNIX OS is done using a special interface layer, called the shell.
• There are quite a few different shells available, we will use bash (some others are sh, ksh, csh, tcsh)
• The shell commands constitute a scripting language

• See environment files (for both csh and bash) at http://www.cs.technion.ac.il/users/yechiel/MTM/UNIXsetup/

Note: Copy/Paste in a UNIX terminal is done by highlight-with-left-mouse/click-right-mouse
[ctrl-C/ctrl-V have different meaning]
Combining commands:

csa ~ [120]  ls tmp

tmp:
Links.sav  Rek92.pdf  Rek92.ps.gz  x/
csa ~ [121]  ls temp

temp:
11.ps.gz  22.ps.gz  Links.sav
csa ~ [122]  ls temp > /tmp/ls.temp;  ls tmp | diff
    !#:3 - > /dev/null ||  ls temp | grep "\.gz"
ls temp > /tmp/ls.temp ; ls tmp | diff
    /tmp/ls.temp - > /dev/null ||  ls temp | grep "\.gz"
11.ps.gz
22.ps.gz
csa ~ [123]