VE280 Recitation Class Notes

VE280 SU19 TA Group

UM-SJTU-JI

May 22, 2019

Source code available at `https://github.com/ve280/VE280-Notes`
Slides modified from the version of YAO Yue's, with permission

RC Week 2

OK, VE280. What is this course?

How to write good code and linked lists

Writing Quality Code

Taste What kind of code are considered "good"?

Motivation What benefit would such code give us?

Techique What is the recipe for such code?

Tools What language features does C++ provide for achieving this goal? How to use them?

Good (Bad?) News: Exams will (mostly) test for the last point.

Elementary Data Structure

Singly linked lists, Doubly linked lists, Circular Arrays ...

OK, "Quality Code"?

It's very hard to give a definition.

Good Code

- Good variable names
- Consistent indentation
- Well tested, documented
- D-R-Y, Don't repeat yourself
- High Coherence / Low coupling
- Open for extension, but closed for modification

Bad Code

- Bad Style
- 200+ lines in a one function
- Functions of 20+ Args
- Magic numbers everywhere
- Seriously you know bad code when you see one (write one).

Many of them are related to *Abstraction*.

Head First *Abstraction*

No abstraction

The Requirement

CubedEnix (CE) is trying to develop a third person shooting game called *World of Armored Blizzard* (WOAB). In this game, AI will control a tank that can fire upon the player. A programmer *Archer* is asked to implement this feature.

The Solution

Archer propose to have a global variable `TANK` of class `Tank` to represent the tank. Archer than writes the following code.

```
...
Player target = TANK.aim(); TANK.fire(target);
...
```

Archer feels happy, so does his boss.

Head First *Abstraction*

No abstraction

Change of Requirement

Players are complaining that the game is too easy. WOAB dev team decided to add 2 more AI controlled tanks. Again Archer is asked to implement it.

The Solution

Archer decided to use 3 global variables `TANK0`, `TANK1`, `TANK2` of class `Tank`. He copies the original code into 3 different places and modify each of them.

```
... Player target = TANK0.aim(); TANK0.fire(target);
... Player target = TANK1.aim(); TANK1.fire(target);
... Player target = TANK2.aim(); TANK2.fire(target);
```

Archer feels happy, so does his boss.

Head First *Abstraction*

No abstraction

Change of Requirement

Within the next 2 months they increased number of tanks to 10. 1 year later, they (finally!) decided to play a sound effect when a tank fires.

The Twist

Archer now has 10 copies! He needs to add a function call `playSound()` to each copy. Unfortunately he missed one location. Archer also doesn't test his code. Now Buggy code is released. Players are unhappy. His boss is unhappy. Archer is fired. Archer is unhappy. Lancer takes his job.

Head First *Abstraction*

With Procedual Abstraction

The Solution

Lancer decided to write a function that takes a `Tank` object as an argument.

```
// Argument: A non-null pointer to a Tank object.
// Effect  : Fires such tank on the current player.
void tankFire(Tank* t) {
    playSound();
    Player target = t->takeAim();
    t->fire(target);
}
```

With this new design Lancer can easily change the number of `TANK`s in the game, or modify the how tanks fire by simply changing this single function.

Head First *Abstraction*

With just procedual abstraction

Change of Requirement

You know what? Those player just can't be satisfied. WOAB dev team now decides to involve not just tanks but also battle ships and planes (and witches and dragons ...). Lancer is asked to implement "fire" feature for all of them.

The Twist

Lancer is now in a tough situation. Clearly Tanks, Ships, Planes... are different things. Their attack would have different effects. There won't be a single function that works for all of them.
Since Lancer didn't take VE280 seriously when he is in JI, Lancer falls back to copying the code for each "character" once. After 2 months, the code is no longer maintainable, and thus he is fired. Saber takes his job.

Head First *Abstraction*

With Data Abstraction

The Solution

The key is to see the elephant in the room. What's the common characteristic of tanks, ships, dragons and witches? They all attack our player!

```
class IAttackPlayer {
    virtual void fire(Player p) = 0;
    virtual Player takeAim() = 0;
}
class Tank : public IAttackPlayer;
class Ship : public IAttackPlayer;
....
```

Head First *Abstraction*

With Data Abstraction

The Solution

Then we can modify the function:

```
// Argument: a object implements IAttackPlayer.
// Effect  : Fires such tank on the current player.
void NpcFire(IAttackPlayer* t) {
    playSound();
    Player target = t->takeAim();
    t->fire(target);
}
```

In the above design Saber abstracted out what the thing "physically" is. She defines things by defining what it can do.

A few more words

It's okay if You don't fully understand above story

That's gives you a good reason to learn it well.

Can you talk something about projects?

Well, projects are mostly very direct. Be careful in the process and they should be pretty easy. Reserve around 5 days for them.
Remember it's easy to simply finish them. But to finish them elegantly trust me when I say it's hard.
Yes, it's hard. But the outcome worth every minute spent.

RC Week 2

Your Linux Operating System

Linux as a operating system *kernel*

Operating systems as a resource manager

Operating systems manage your hardware. These include computation resource (CPU, GPU), storage (Hard drive), communication resource (network)...

The core of OS, a.k.a. *kernel*

At the very heart of the OS there lies the kernel. This piece of software talks directly to the hardware. It provides a unified way of accessing your storage devices (Filesystems), graphic devices, network...

"Linux" is first the name of the kernel

This implies you can actually change your desktop environment: fancier? go with Unity/KDE/Gnome. Light weight & fast? go with LXDE or XFCE.

Flavors of Linux: *Distributions*

Common Choices

This flexibility of choice allows to tailor the system towards our own need. Some commonly accepted configurations become "Distributions".

Ubuntu Series

Maintained by Canonical, it's actually a series of distributions with different choice of desktop environment. Ubuntu / *Unity*. Kubuntu / *KDE*. Lubuntu / *LXDE*. Xubuntu / *XFCE*. Use *apt*.

Debian

Once the most widely use distribution, the father of Ubuntu [that's right :=)]. Package management system is *apt*

Fedora and RedHat Linux Enterprise

Maintained by Red Hat. Fedora is new and cutting edge. RedHat now focus on enterprises. Package Management is *yum/dnf*.

Flavors of Linux: *Distributions*

Uncommon choices

Open SUSE

Maintained by german company SUSE. Focus on stability.

Arch Linux / Manjaro

ROLLING UPDATE! Absolutely cutting edge. Great community and wiki. User is assumed to be experienced. Installation of Arch Linux is done on command line while Manjaro makes easier. Package management system is *pacman*.

Linux From Scratch (LFS) / Gentoo

Try LFS if you are absolutely interested in figuring out how everything works. Think LFS is crazy enough? Well Gentoo compiles every piece of software you use FROM THE SOURCE!

Bash on Windows Subsystem for Linux

Think big, with abstraction

For Linux programs, technically, you don't need an "real" linux kernel to run them. As long as you have some thing that looks like a real one (one that follows the same abstraction).

Windows Subsystem for Linux

This year Microsoft actually release one. Installation guide is available here(click me).
This solution is easy to use, clean. It saves you time copying files back and forth from the virtual machine. You can simply develop in Windows using your favorite tool (IDE and test it easily under "Linux". It's a good solution for the purpose of this course.

Virtual Machines vs. Containers

Virtual Machines

A virtual machine (VM) is an emulation of a computer system. Virtual machines are based on computer architectures and provide functionality of a physical computer. The CPUs, memory, kernel, file systems and devices are all virtualized and need translation, which makes it slower than a physical computer.

Containers

A container shares the CPUs, memory, kernel, file systems and devices with the host machine, which makes it almost as fast as the host machine when executing machine code (no virtualization means no translation of machine code). Everything in a container is something mapped from the host system. Docker is a lighweighted, standalone and secure container daemon which is very popular nowadays.

Aspects of Linux: Users

MultiUser	Multiple person can operate the system at once.
Group	Each User belongs to one or more groups
Home Dir	Each user has his/her own directory under `/home`
~/	Shorthand for home directory. This one is user specific.
Premission	Owner/Group/Other Read/Write/Execute Flags
root	A special super user `root` overrides all security
su	Command that starts a temporary shell as `root`
sudo	A command that temporarily grant superuser privilege to current user. Namely, "execute the following command as 'root'".

The Miranda warning of `sudo`:

```
We trust you have received the usual lecture from the local
System Administrator. It usually boils down to these three things:
    #1) Respect the privacy of others.
    #2) Think before you type.
    #3) With great power comes great responsibility.
root's password:
```

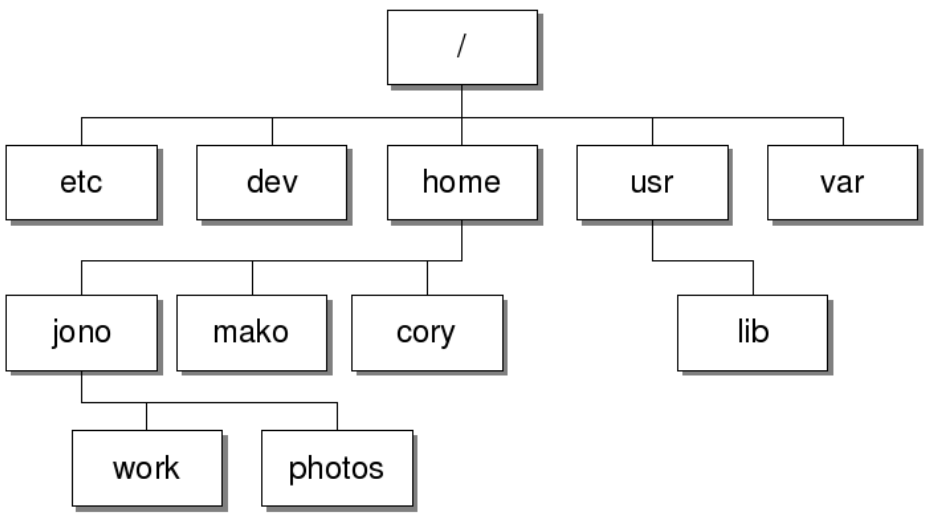
Aspects of Linux: Filesystem

Filesystem is an abstraction of storage

That's right, abstraction again. Filesystem hides the physical storage schema of your files. Instead it should reflect how your files are logically organized.

Linux Filesystem

In Linux files are organized in a tree-like structure.



Aspects of Linux: Filesystem

Caveats for the names

A word from Ken Thompson

> *Ken Thompson was once asked what he would do differently if he were redesigning the UNIX system. His reply: "I'd spell creat with an e."*

/bin	**Bin**aries	/mnt	**Mount**ed filesystem
/dev	**Dev**ices	/etc	Configuration files
/boot	**Boot**strap	/usr	**U**nix **s**ystem **r**esources
/sbin	**S**ecure **bin**aries	/home	Places for the user's files

More at https://wiki.debian.org/FilesystemHierarchyStandar

Aspects of Linux: Shell

CLI and Shell

Command **L**ine **I**nterface is fast, consumes minimum resources and effective. **G**raphical **U**ser **I**nterface comes with a much larger cost. Management tasks are generally easier on CLI (think about coding!).

The program that interprets user commands and provides feedbacks is called a *Shell*. When you login to the computer, the shell runs automatically. You interact with the computer through the shell.

Differenct choices of shell

The "standard" one on most linux is `bash`, as in "**B**ourne's **A**gain **Sh**ell". But choices like `zsh`, `fish`, `csh` are much more easier to use (cooler). Checkout the "oh-my-zsh" project on GitHub if you are interested.

Aspects of Linux: Shell

The working directory

How does `fopen("test.txt")` work? I mean, how does the computer know where to find `text.txt`?
Each program is associated with a special directory called "working directory". Normally this is the directory where you execute the program. (Not where the program is!).

Related commands

pwd **P**rint **w**orking **d**irectory

cd **C**hange **d**irectory. Each directory has 2 very special "sub-directory". "**./**" and "**../**". They are logically sub-directory. Meaning you can do `cd /home/john/./` and `cd /home/john/../`, but the in fact, the first command takes you to `/home/john/` and the second one takes you to `/home/`.

"`cd ./`" keeps where you are and "`cd ../`"" takes you 1 level up.

Aspects of Linux: Shell

Executing programs

The general syntax is

`executable_file arg1 arg2 arg3 ...`

Conventions are:
1) arguments begin with - are called "switches" or "options"
2) one dash (one -) are called short switches, e.g. `-l`, `-a`
3) short switch always uses single letter to specify. Case / lower letters can have very different meanings! Be very careful. 3) multiple short switches can often be specified at once. e.g. `ls -al`
4) two dashes (one -) are called long switches, e.g. `--all`, `--mode=linear`. Usually long switches use whole words other than acronyms.
5) **THERE ARE OUTLIERS!**. Unfortunately gcc and g++ are two of these naughty boys.

Aspects of Linux: Shell

Useful commands

ls **List** files & folders under a directory. This command takes zero or more arguments. If the argument is a directory, list that dir. If the argument is a file, show information of that specific file. If no arguments are given, list working directory.

- -a List hidden files as well. Leading dot means "hidden".
- -l Use **l**ong format. Each line for a single file.

mkdir **Mak**e **dir**ectory, self-explanatory.

rm **Rem**ove files / directory. It is **extremely dangerous** to run `rm` with administrative privilege! See the bumblebee accident. `https://github.com/MrMEEE/bumblebee-Old-and-abbandoned/issues/123`

- -r Deletes files/folders recursively. Folders requires this option.
- -f Force remove. Ignores warnings.

rmdir **Rem**ove **dir**ectory, only **empty** ones can be removed.

Aspects of Linux: Shell

Useful commands, Cont'd

touch Designed to change the time stamp of file. Commonly used to create an empty file.

cp **Co**p**y** files/folder. Takes 2 arguments `source` and `dest`. Be very careful if both `source` and `dest` are both existing folders. Try it yourself!

- **-r** Copy files/folders recursively. Folders requires this option.

mv **M**o**v**e files / directory. Takes 2 arguments `source` and `dest`. If `source` and `dest` is the same location, this command essentially does a rename. Pay attention to the situation where both arguments are already existed.

cat Con**cat**enate files. Takes multiple arguments and print their content one by one to `stdout`. When there is just one argument, it essentially displays the content.

Aspects of Linux: Shell

Long Format of File Information

```
root@liu-Lenovo-M41-70:/# ls -l
```

permission	link	user	group	file size	modified time	file name
drwxr-xr-x	2	root	root	4096	Apr 21 06:13	bin
drwxr-xr-x	3	root	root	4096	Apr 21 06:13	boot
drwxr-xr-x	2	root	root	4096	May 14 2018	cdrom
drwxr-xr-x	20	root	root	4060	May 22 14:31	dev
drwxr-xr-x	155	root	root	12288	May 14 06:21	etc
drwxr-xr-x	3	root	root	4096	May 14 2018	home
lrwxrwxrwx	1	root	root	33	Apr 5 06:29	initrd.img -> boot/initrd.img-4.15.0-47-generic
lrwxrwxrwx	1	root	root	33	Apr 5 06:29	initrd.img.old -> boot/initrd.img-4.15.0-46-generic
drwxr-xr-x	22	root	root	4096	Jun 30 2018	lib
drwxr-xr-x	2	root	root	4096	Jun 25 2018	lib32
drwxr-xr-x	2	root	root	4096	Apr 27 2018	lib64
drwxr-xr-x	2	root	root	4096	Jun 25 2018	libx32
drwx------	2	root	root	16384	May 14 2018	lost+found
drwxr-xr-x	3	root	root	4096	May 15 2018	media
drwxr-xr-x	2	root	root	4096	Apr 27 2018	mnt
drwxr-xr-x	10	root	root	4096	Dec 4 20:45	opt
dr-xr-xr-x	247	root	root	0	May 22 14:30	proc
drwx------	9	root	root	4096	Nov 22 15:06	root
drwxr-xr-x	32	root	root	940	May 22 14:33	run
drwxr-xr-x	2	root	root	12288	May 14 06:20	sbin
drwxr-xr-x	2	root	root	4096	May 14 2018	snap
drwxr-xr-x	3	root	root	4096	Aug 14 2018	srv
-rw-------	1	root	root	2147483648	May 14 2018	swapfile
dr-xr-xr-x	13	root	root	0	May 22 14:41	sys
drwxrwxrwt	18	root	root	4096	May 22 14:41	tmp
drwxr-xr-x	13	root	root	4096	Oct 3 2018	usr
drwxr-xr-x	14	root	root	4096	Apr 27 2018	var
lrwxrwxrwx	1	root	root	30	Apr 5 06:29	vmlinuz -> boot/vmlinuz-4.15.0-47-generic
lrwxrwxrwx	1	root	root	30	Apr 5 06:29	vmlinuz.old -> boot/vmlinuz-4.15.0-46-generic

Aspects of Linux: Shell

Long Format of File Information, Cont'd

File permission
- first character: '-' regular file; 'd' directory
- read, write, execution permission of the owner
- read, write, execution permission of the group
- read, write, execution permission of everyone else
- last character: '-' regular file; 'x' executable

Link
- the number of hard links of the file
- a regular file has only one hard link (of itself)
- a directory has $2 + n$ hard links where n is the number of its subdirectories because there are two extra links to . and ..

Owner/Group
- act with file permissions

File Size
- in bytes

Aspects of Linux: Shell

CLI Utilities

nano Command line file editor.

diff Compare the **diff**erence of two files.

- **-y** A side by side view, try it yourself.
- **-w** Ignore white spaces.

less Prints the content from its `stdin` in a readable way.

vi Advanced text editor

vim **v**i **im**proved, both `vi` and `vim` can be exited by first press `ESC` and type "`:q!`". You should know this in case your "friend" opens a `vi` window when you are away, so you won't get stuck inside.

grep Filters input and extracts lines that contains specific content. Very useful in debugging programs.

echo Prints its arguments to `stdout`.

Aspects of Linux: Shell

IO Redirection

When you are reading `cin` or writing to `cout`, you are essentially read/writing 2 special files. Again, abstraction, right?
It is possible to "switch" these two "virtual files" with real files before the actual execution of the program. This is called IO "redirection".
There are 4 most common redirections:

`exec < input`	Use `input` as `stdin` of `exec`
`exec > output`	Write the `stdout` of `exec` into `output`. Note this command always truncates the file. File will be created if it is not already there.
`exec >> output`	Similar to >, but it appends to ouput.
`prog1 | prog2`	Called a "pipe". Connects the `stdout` of `prog1` to `stdin` of `prog2`

Aspects of Linux: Shell

Globbing

Sometimes you don't care about one file, you care about **all** files. You can use * to represent any string in command arguments. This is called "globbing" and the * is called a wildcard.

- List every `.cpp` file in home dir: `$ ls -al ~/*.cpp`
- Delete everything in `/temp`: `$ rm -rf /temp/*`
- Combine all `.h` into one: `$ cat *.h > combined.h`

Looking for help

The "goto" location for help in Linux is the `man` command. This is a short hand of "**man**ual".

- Find out information about "vi": `$ man vi`
- Confused about cp: `$ man cp`

Example of shell commands

Setup

The following program `rc2xm` reads from it's standard input line by line and prepends `xm` at the beginning and print it out.

```
#include <iostream>
int main() {
    std::string str;
    while(std::getline(std::cin, str))
        std::cout << "xm" << str << std::endl;
    return 0;
}
```

We prepare an input file `xm.in` with the following content:

`xd↩ cg↩ hss ↩ xtt ↩ qs ↩ jcc ↩ xdtql ↩ zdnxd ↩`

We use ↩ to represent new-line to save space on slides.

Example of shell commands

Examples

For each of the following commands what is it's effect? What is the final output?

The commands

```
$ cat xm.in
$ cat xm.in > xm3.in
$ ./rc2xm < xm.in
$ ./rc2xm < xm.in > xm.out
$ cat xm* > xm2.in
$ cat xm*.in >> xm.out
$ cat xm* | ./rc2xm > out
$ ./rc2xm <out | grep "xd"
```

Output of last command

```
xmxmxmxd↩xmxmxmxdtql↩
xmxmxmzdnxd↩xmxmxd↩
xmxmxdtql↩xmxmzdnxd↩
xmxmxd↩xmxmxdtql↩
xmxmzdnxd↩xmxmxmxd↩
xmxmxmxdtql↩xmxmxmzdnxd↩
xmxmxd↩xmxmxdtql↩
xmxmzdnxd↩xmxmxd↩
xmxmxdtql↩xmxmzdnxd↩
```

An extra story: Package Managers

The need for a package manager

- Packages a fancy name for "a piece of software"
- Linux fs convention separates package into different locations.
- Packages reuse code from other packages, i.e. *dependencies*.

The apt package manager

- Standard choice for Debian (and its derivatives)
- `apt-get install` for installing new package.
- `apt-get upgrade` for upgrading existing package.
- `apt-get update` refreshes the list of available software.
- `apt-get autoremove` removes installed package.
- `apt moo` for a tiny surprise :)