VE280 2022FA RC1

Introduction:

Prerequisites:

- C++ editor
- GNU g++ installed
- Linux development environment (**REQUIRED!!**)
  - Linux in virtual machine
  - WSL
  - Dual Boot
  - MacOS (not recommended but acceptable)
- (Not for now) install valgrind
  - ```
    sudo apt-get install valgrind
    ```
- (Optional) know how to use git
    ```

Tips

- Lectures: Attend lectures and memorize details in the slides
- Projects: Start early and pay attention to code quality
- Exams: Get used to write piece of codes on paper

L2: Intro to Linux

Shell/Terminal

The program that interprets user commands and provides feedbacks is called a **shell**. Users interact with the computer through the shell. And **Terminal** provides an input and output environment for commands

- The general syntax for shell is `executable_file arg1 arg2 arg3 ...` .
- Arguments begin with `-` are called ”switches” or ”options”;
  - one dash `-` switches are called short switches, e. g. `-l` , `-a` . Short switch always uses a single letter and case matters. Multiple short switches can often be specified at once. e. g. `-al` = `-a -l` .
  - Two dashes `--` switches are called long switches, e. g. `--all` , `--block-size=M` . Long switches use whole words other than acronyms.
  - For many programs, long switches have its equivalent short form, e. g. `--help` = `-h`

Basic Command

The followings are really important!

- `man <command>` : display the **man**ual for a certain command (**very useful!**)
- `pwd` : **p**rint the path of current **w**orking **d**irectory
- `cd <directory>` : **c**hange **d**irectory
- `ls <directory>` : **ls**t the files under the directory
  - Arguments:
    - no argument, list the working directory (equivalent to `ls .`)
    - If the argument is a directory, list that directory (`ls <directory>`)
  - Optional:
    - `-a` : list all, including hidden ones
    - `-l` : list in long format
- `mkdir <directory-name>` : **m**a**k**e **dir**ectory
- `rmdir <directory-name>` : r**m**ove **dir**ectory
  - Only empty directories can be removed successfully.
- `touch <filename>` : create a new empty file
- `rm <files>` : r**m**ove the files
  - Optional:
    - `-i` : prompt user before removal, and put it into `~/.bashrc`
    - `-r` : Deletes files/folders recursively. **Folders requires this option**, e. g. (`rm -r testDir/`)
    - `-f` : Force remove. Ignores warnings.

- ⚠This is DANGEROUS. See the famous bumblebee accident.

- `cp <source> <destination>` : **c**o**p**y
  - `-r` Copy files/folders recursively. **Folders requires this option**.
  - Variations:

Source	Destination	Result
		copy the content of `file1` into `file2`
		copy the file into a directory
		copy two files into one directory
		Not allowed
	(exist)	copy `dir1` inside `dir2`
	(not exist)	copy `dir1` as `dir2`

- `mv <source> <destination>`: **m**o**v**e
  - **Exercise:** Can you make a similar table as above?
- `cat <file1> <file2> ...`: con**cat**enate

Advanced Command

Not saying that they are not important! Still possible to exist in exam! The followings serve as a documentation that you may refer to.

- `less <file1> <file2>` : display the content of the files
  - Less is a program similar to more, but it has many more features.
  - Less does not have to read the entire input file before starting, so with large input files it starts up faster than text editors like vi.
  - quit `less` : press `q`
  - go to the end of the file: press `G` (`shift+g` ? go to beginning?)
  - search: press `/` , then enter the thing to be searched, press `n` for the next match (`N` ?)
  - *multiple files: enter `:n` to view the next file, enter `:p` to view the previous one
- `diff <file1> <file2>`: Compare difference between two files
  - This command is important for you project!
  - If there are differences: lines after "<" are from the first file; lines after ">" are from the second file.
  - In a summary line: `c` : change; `a`: add; `d` : delete
  - `-y` Side by side view
  - `-w` Ignore white spaces (space, tab)
  - return value: 1 if the same, 0 if not
    - ```
      if !(diff out1 out2 > diff); then
          echo "\033[31m Wrong Answer! \033[0m"; exit;
      ```

- `nano` and `gedit` : basic command line file editor
      ```

    - Advanced editors like `vim` and `emacs` can be used also.
- Auto completion: type a few characters, then press `Tab`
    - single match: complete the remaining
    - multiple match: list all candidates
- `sudo apt-get install` : install a program
    - `sudo` command: execute command as a superuser, and requires you to type your password.
    - Editing read-only file: `sudo vim <file>`
    - `sudo apt-get autoremove` : remove a program

IO Redirection

Now, this is important!!

Most command line programs can accept inputs from standard input (keyboard by default) and display their results on the standard output (display by default).

- `executable < inputfile`: Use inputfile as `stdin` of executable
- `executable > outputfile` : Write the `stdout` of `executable` into outputfile
    - Note this command always truncates the file
    - Outputfile will be created if it is not already there
- `executable >> outputfile`: **Append** the `stdout` of executable into outputfile
- They can be used in one command line. Like `executable < inputfile > outputfile`.
- ⋆`exe1 | exe2` Pipe. Connects the `stdout` of exe1 to `stdin` of exe2.
    - e. g., `./add < squareofsum.in | ./square > squareofsum.out`

Need more help?

Try `man bash` and search by typing `/redirect`.

Linux File System

Directories in Linux are organized as a tree. Consider the following example:

```
/                                   //root
├── home/                           //users's files
    ├── username1
    ├── username2
    ├── username3
    └── ...
├── usr/                            //Unix system resources
    ├── lib
    └── ...
├── dev/                            //devices
├── bin/                            //binaries
├── etc/                            //configuration files
├── var/
└── ...
```

Remember the following:

- root: `/`
  - The top most directory in Linux filesystem
  - What will happen if you `cd ..` at root directory?
- home: `~`
  - Linux is multi-user. The home directory is where you can store all your personal information, files, login scripts
  - In Linux, it is equivalent to `/home/`
- current: `.`
- parent: `..`

More can be find here: https://ipcisco.com/lesson/linux-file-system

File Permissions

The general syntax for long format is:

`<permission> <link> <owner> <group> <size>(in bytes) <modified_time> <file_name>`

In total, 10 characters for permission syntax:

- char 1: Type. `-` for regular file and `d` for directory.
- char 2-4: Owner permission. `r` for read, `w` for write, `x` for execute.
- char 5-7: Group permission. `r` for read, `w` for write, `x` for execute.
- char 8-10: Permission for everyone else. `r` for read, `w` for write, `x` for execute.

As you can imagine, the permission goes down from owner -> group -> anyone else.

L3: Compile Program

Header Guard

```
//add.h
#ifnedf ADD_H // test whether ADD_H has not been defined before
#define ADD_H
int add(int a, int b);
#endif
```

Notes: If `ADD_H` has not been defined before, `#ifndef` succeeds and all lines up to #endif are processed. Otherwise, `#ifndef` fails and all lines between `#ifndef` and `#endif` are ignored.

What will happen for the following two header files, with/without header guard in add.h?

my_project1.h

```
#include "add.h"
...
```

my_project2.h

```
#include "add.h"
#include "my_project1.h"
```

Including of a header file more than once may cause multiple definitions of the classes and functions defined in the header file.

With a header guard, we guarantee that the definition in the header is just seen once.

`g++`

This is a simple review for vg101/vg151.

Compile in one command: `g++ -o program source1.cpp source2.cpp`. (header files don't need to be included)

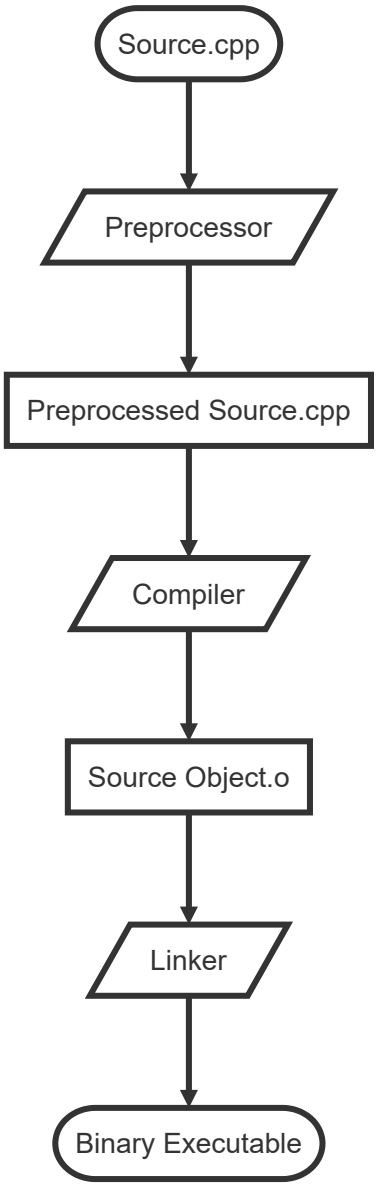
Run the program: `./program`

Some options for g++:

- `-o <out>` Name the output file as . Outputs `a.out` if not present.
- `-std=` Specify C++ standard.
- `-Wall` Report all warnings. **Do turn `-Wall` on during tests**

Compilation Process

More to come in next RC~



Appendix: Coding Style

Good coding style

- Meaningful variable names;
- Consistent indentation;
- Well tested, documented and commented;
- Rule of D-R-Y: Don't repeat yourself;

The following is a good function example.

```
class Student{
    // represents a JI student.
    string name;
    string major;
    int stud_id;
    bool graduated;

public:
    Student(string name="default", string major="ece", int stud_id=0, bool
graduated=false);
    // EFFECTS  : create a new student.

    bool compMajor(const Student &stud) const;
    // EFFECTS  : return true if "this" student has the same major as "stud",
    //            return false otherwise.

    bool hasGraduated() const;
    // EFFECTS  : return true if "this" student has graduated,
    //            return false otherwise.

    void changeMajor(string new_major);
    // MODIFIES : "major",
    // EFFECTS  : set "major" to "new_major".
};
```

Google C++ style guide

Link: https://google.github.io/styleguide/cppguide.html

Some rules:

- The `#define` Guard
- Inline Functions < 10 lines
- Forward declarations
- ……

Useful Tool

`cpplint` is a tool to test whether your code follow the google C++ style. And usually… it would say your code quality is really bad…

Installation (need python installed first)

```
pip install cpplint
```

Basic usage

```
cpplint p1.cpp
```

Need help?

```
cpplint --help
```

Bad coding style

- Vague variable names;
- Arbitrary indentation;
- Put all the implementation into main function.
- Repeat part of your code or have codes of similar function;
- Long function. Say 200+ lines in a one function;
- Too many arguments for one function. Say functions of 20+ arguments;

The following is a bad function example.

```
int poly_evaluation(int x, int *coef, unsigned int d)
{
    int r = 0, p = 1;
for(int i = 0; i<= d; i++){
    r += coef[i] * p;
        p *= x;}
    return r;}
```

Exam-Like Exercises:

The Linux Command Part takes 16 / 100 (1/6) in last year's midterm. The followings are typical exercises for the exam:

1. Rename the folder `VG101` into `VE280`. Assume `VE280/` does not exist at first.

   ```

   ```

2. List all the files(including hidden files) under home directory

   ```

   ```

3. Compile the source files `main.cpp` , `add.cpp`, `add.h` into executable file `main`

   ```

   ```

4. Copy all the content of `a.txt` to `b.txt`, **without** use `cp`

   ```

   ```

5. Remove the non-empty directory `VE280/` by force

```

```

6. Create an empty folder `VE280/` in the parent working directory

```

```

7. Compare two files `1.cpp` and `2.cpp` and save the result in a file called `result.txt`

```

```

Feel really easy? Then you should have no problem in the exam for this part!!! 😄

References:

[1] Weikang, Qian. VE280 Lecture 1-3.

[2] Pingchuan Ma. VE280 RC1. 2021 FA.