

VE280 Mid RC Part 1

General tips for the exam

- Go over all the basic concepts covered in slides.
- Go over all the exercises and projects you've done, especially the implementation details.
- Grammar mistakes when writing codes **do** lead to deduction, e.g. "==" instead of "=".

Lecture 2 Linux Commands

2.1 Basic Commands

`pwd` : Print working directory.

`ls <path>` : List files and directories in the directory specified by `<path>`.

- `ls -a` : list all files (including hidden ones)
- `ls -l` : list in long format

Long Format example:

```
-rwxr-xr-- 1 user group 0 Jan 1 00:00 file
```

- First character → type of file
 - `-` : regular file
 - `d` : directory
- Next 9 characters → permissions
 - `r` (read), `w` (write), `x` (execute)
 - First 3 = owner, next 3 = group, last 3 = others

`cd <path>` : Change to the directory specified by `<path>`.

- `.` : current directory
- `..` : parent directory
- `~` : home directory
- `/` : root directory

`mkdir <directory>` : Make a new directory specified by `<directory>`.

`rmdir <directory>` : Remove an **empty** directory specified by `<directory>` .

`touch <file>` : Create an empty file specified by `<file>` .

`rm <file/dir>` : Remove a file or directory specified by `<file/dir>` .

- `rm -r <dir>` : remove a directory recursively (for non-empty folders).

`cp <src> <dest>` : Copy file or directory from `<source>` to `<destination>` .

- `cp -r <src> <dest>` : copy a directory recursively (for non-empty folders).

`mv` : Move / rename file or directory.

- `mv <file1> <file2>` : rename
- `mv <file> <directory>` : move file into directory
- `mv <dir1> <dir2>` : move directory into another

`cat <file>` : Print content of a file.

`less <file>` : View file in read-only mode (`q` to exit).

`diff <file1> <file2>` : Compare two files.

If same → no output; if different → `<` from file1, `>` from file2.

`nano` , `vim` : text editors.

Example for Nano: `Ctrl+S` save, `Ctrl+X` exit.

2.2 I/O Redirection

Redirect input / output of a command:

```
command < input > output
```

Examples:

```
cat < input.txt > output.txt
./my_program < input.in > output.out
```

- Overwrites output file if exists
- Fails if input file does not exist
- Creates output file if absent
- Use `>>` to append instead of overwrite : `./my_executable >> output.out`

Lecture 3 Develop and Compile Programs

3.1 Compilation Process

Stages

1. Preprocessing — remove comments, expand macros, include headers.
2. Compilation & Assembly — `.cpp → .o`.
3. Linking — combine objects into executable.

Example Suppose you have a program with `my_program1.cpp` , `my_program1.h` , and `my_program2.cpp` , and you want to compile them into an executable file `my_executable` , you can use the following commands to compile them:

```
g++ -o my_executable my_program1.cpp my_program2.cpp
```

Equivalent to:

```
g++ -c my_program1.cpp
g++ -c my_program2.cpp
g++ -o my_executable my_program1.o my_program2.o
```

Note that you don't need to manually do the preprocessing step by commands and don't need to add the header files to the compilation commands.

3.2 Makefile

Automate the compilation process.

Syntax

```
target: dependencies
      command
```

Example Makefile

```
all: my_executable

my_executable: my_program.o my_class.o
```

```

g++ -o my_executable my_program.o my_class.o

my_program.o: my_program.cpp
g++ -c my_program.cpp

my_class.o: my_class.cpp
g++ -c my_class.cpp

clean:
rm -f my_executable *.o

```

To use the Makefile to compile the program, type `make` in the terminal. To clean the compiled files, type `make clean`.

Compile Flags

- `-c` : compile into object files
- `-o` : specify output file name

3.3 Header File and Header Guard

Header file → contains declarations of functions, classes, and variables.

```

#ifndef MY_HEADER_H
#define MY_HEADER_H

// function declarations / class declarations

#endif

```

- Each function declared here should be **defined** in corresponding `.cpp` file.
- Header guards prevent **multiple definitions** and **reprocessing**.
- If the header file is included multiple times, the compiler will raise an error because the same function is declared multiple times.

Example:

```

#include "my_header.h"

void my_function() {
    // function body
}

```

Lecture 4 Review of C++ Basics

4.1 Basic Operators

- `i++` → returns old value.
- `++i` → returns new value.

4.2 lvalue and rvalue

- **lvalue** → can appear on both sides of `=`
- **rvalue** → only on the right side of `=`

Exercise: are the following elements lvalue/ rvalue?

```
int a = 1;
int b = a;
const int c = a + b;
int *p = &a;
```

4.3 Function Declaration and Definition

Declaration

```
// return_type function_name(parameter_list);
int add(int a, int b);
void print(string s);
```

Definition

```
int add(int a, int b) {
    return a + b;
}

void print(string s) {
    cout << s << endl;
}
```

Note that this can come before or after the function is called.

4.4 Pointers, References and Arrays

Pointers

A variable that stores the address of another variable. Changing the value of a pointer will change the value of the variable it points to.

```
int a = 280;
int *p = &a;
cout << *p << endl; // value of a
*p = 1000000;        // changes a
```

- `*` : dereference operator
- `&` : address-of operator (rvalue, like in the example above, you cannot change `&a` .)

References

An alias of a variable. Changing the value of a reference will change the value of the variable it refers to.

```
int a = 280;
int &r = a;
r = 1000000;
```

- Must initialize when declared
- Cannot change target after initialization

Exercise: What is the value of x, y, and r?

```
int x = 0;
int &r = x;
int y = 1;
r = y;
r = 2;
```

Exercise: What is the value of x, y, and *p?

```
int x = 0;
int *p = &x;
int y = 1;
p = &y;
*p = 2;
```

Arrays

```
int a[5] = {1, 2, 3, 4, 5};
cout << a[0] << endl; // 1
```

- Arrays are contiguous memory; `a` is a pointer to `a[0]`.
- Size fixed at compile time.

4.5 Function Call Mechanism

- **Pass by value** : the value of the parameter is copied to the function. Modifying the parameter inside the function will not affect the original variable.
- **Pass by reference** : the address of the parameter is passed to the function. Modifying the parameter inside the function will affect the original variable.

Arrays are passed by reference implicitly:

```
void f(int a[]) {
    a[0] = 1;          // passed by reference, the value of a[0] is changed
}
```

Exercise: Are x, y, and z passed by value/ reference? What's the output of the code?

```
void f(int x, int &y, int *z) {
    x = 1;
    y = 1;
    *z = 1;
}

void test() {
    int a = 0;
    int b = 0;
    int c = 0;
    f(a, b, &c);
}
```

```
    cout << "a: " << a << " b: " << b << " c: " << c << endl;
}
```

Advantage of pass by reference with pointers:

- You can directly change the value of the variable that the pointer points to. This avoids copying instances of large objects sometimes.

Advantage of pass by reference with references:

- More readable and intuitive than pointers.
- It keeps the advantage of avoiding unnecessary copying.

4.6 Structs

User-defined data types that can contain multiple variables of different types. The members of a struct can be common data types or other structs.

```
struct Student {
    string name;
    int id;
    double gpa;
}; // note semicolon
```

Initialization

```
Student s1; // s1 is a struct variable
s1.name = "Alice";
s1.id = 280;
s1.gpa = 4.0;
Student s2 = {"Bob", 281, 3.9}; // another way to initialize
Student *s3 = &s2; // s3 is a pointer to s2
s3->name = "Jack"; // access the members of struct pointer using
->
```

Answers for exercises

4.2

```

int a = 1;           // a: lvalue, 1: rvalue
int b = a;           // b, a both lvalue
const int c = a + b; // c, (a+b) both rvalue
int *p = &a;         // p lvalue, &a rvalue

```

4.4

```

int x = 0;           // x = 0
int &r = x;
int y = 1;           // y = 1
r = y;               // x = 1
r = 2;               // x = 2
// Finally, r = x = 2, y = 1

```

```

int x = 0;           // x = 0
int *p = &x;         // p points to x
int y = 1;           // y = 1
p = &y;              // p points to y
*p = 2;              // y = 2
// Finally, *p = y = 2, x = 0

```

4.5

```

void f(int x, int &y, int *z) {
    x = 1;           // passed by value
    y = 1;           // passed by reference
    *z = 1;          // passed by reference
}
void test() {
    int a = 0;
    int b = 0;
    int c = 0;
    f(a, b, &c);
    cout << "a: " << a << " b: " << b << " c: " << c << endl;
    // Finally, a = 0, b = 1, c = 1
}

```

Reference

- [1] Yunxiang Yao. *ECE2800J Mid RC Part 1*. 2025.
- [2] Weikang Qian. *ECE2800J Slides*. 2025.