VE280 2022FA Mid RC

Tips for Exam

- Be HONEST
- Be careful
- Be critical

L2: Linux Command Checking List

Commands	Meaning?	Options?	Familiar?
ls			
man			
pwd			
cd			
touch			
mkdir			
rmdir			

Commands	Meaning?	Options?	Familiar?
rm			
cp			
mv			
cat			
diff			
`>`, `<`, `>>`			
`/`, `~`, `.`, `..`			

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.

Example:

```
drwxr-xr-x 6 mary mary 1024 Oct 9 1999 web_page
```

L10: I/O Streams

`cin`, `cout` & `cerr`

>> and <<

In C++, streams are **unidirectional**, which means you could only `cin >>` and `cout <<`.

If we look into `cin`, it's an object of class `istream` (input stream). `operator>>` (the extraction operator) is one of it's member function.

Return value also a reference of `istream` -> it can be cascaded like `cin >> foo >> bar >> baz;`

Useful functions:

```
istream& getline (istream& is, string& str);
std::ios::operator bool // member of istream -> if(cin), while(cin) -> read
until eof
istream& get (char& c); // member of istream
```

Differences:

- `>>` will read until reaching the next space or `\n`, and the space and `\n` will still be left in the buffer. And space and `\n` won't be stored into the parameter.
- `getline` would read a whole line and discard the `\n` at the end of the line directly.
- `get()` reads a single character, whitespace or newlines.

File Stream

```
#include <fstream>

std::ifstream iFile; // inherit from istream
std::ofstream oFile; // inherit from ostream
iFile.open("myText.txt");  // if unsuccessful to open, iFile would be in failed state, if(iFile) returns false. But member function open() has void return type!!!

iFile >> bar;
while(getline(iFile, line)) // simple way to read in lines.

oFile << bar;
```

String Stream

```
#include <sstream>

istringstream iStream; // inherit from istream
iStream.str(line); // assigned a string it will read from, often used for getline
iStream >> foo >> bar;
iStream.clear(); // Sometimes you may find this useful for reusing iStream

ostringstream oStream; // inherit from ostream
oStream << foo << " " << bar;
string result = oStream.str(); // method: string str() const;
```

L8: Enum

Enum is a type whose values are restricted to a set of integer values. Advantages:

- Use less memory than `std::string`
- More readable than `const int` or `char`
- Limit valid value set, so compiler help you find spelling mistakes.

Example

```cpp
#include <iostream>
enum A {
    a, b, c=-1, d, e=5, f, g=a + e, h
};
int main() {
    std::cout << a << ' ' << b << ' ' << c << ' ' << d << ' '
        << e << ' ' << f << ' ' << g << ' ' << h << '\n';
    // Output is 0 1 -1 0 5 6 5 6
    return 0;
}
```

- By default the enum value starts from 0, and increments for each value
    - But you can also assign any integer value to them
- Values in enum (a, b, c,...) can be treated as global `const int`
    - Can be compared (<, >, ==, !=)

L9: Program Arguments

Write a main function that takes program arguments:

```cpp
int main(int argc, char *argv[]) {
    /* code here */
}
```

Or in a way easier to memorize:

```cpp
int main(int argc, char ** argv) {
    /* code here */
}
```

Question: If a executable program is named `ex1`, then `argv[0]` must be `./ex1`?

L11: Testing

Concepts

Five Steps in testing:

- Understand the specification (Design requirement)
- Identify the required behaviors (Specification boil down; abstract; Party A)
- Write specific tests (**Simple**+ **Normal**+ **Nonsense**)
- Know the answers for those tests (The correct output; concrete; Party B)
- Stress tests (**large** and **long running**)

Things you need know at the very least

- Determine a test case to be simple/normal/nonsense.
- Write simple/normal/nonsense test cases for a function/program.
- Explain why is a test case simple/normal/nonsense.

Example

- Specification

  ```
  Write a function to calculate factorial of non-negative integer, return -1
  if the input is negative.
  ```

- Behavior
  - Normal: return `_____` for input `_____`
  - Boundary: return `_____` for input `_____`
  - Nonsense: return `_____` for input `_____`

L13: Abstract Data Type

What is ADT?

ADTs provide an **abstract description** of **values** and **operations**. In short, to define an ADT, we only need to know:

- What values it represents: a mobile phone that can make and receive calls
- What can it do to these values (operations): turn on/off, make/receive call, text message, play games...

Why use ADT?

Abstraction hides implementation detail and makes users' life easier. ADTs provide **two** advantages for users:

- Information hiding: We don't need to know the details (how messages travel across the air to reach our phone?)
  1. The **user do not need to know** (and should not need to know) **how the object is represented**.
  2. The **user do not need to know** (and should not need to know) how the **operations** on the object are implemented.
- Encapsulation: the objects and their operations are defined in the same place (You don't need to buy the screen, the circuit board, the wifi module...You just buy a phone) **combine both data and operations in one entity**.

ADT also benefits the developers:

- ADTs are **local**: the implementation of other components of the program does not depend on the implementation of ADT. To realize other components, you only need to focus locally.
- ADTs are **substitutable**: you can change the implementation and no users of that type can tell.

ADT in C++: Class

Getters & Setters

```
class Student {
    int score; // default: private
public:
    // A getter of score, qualified as const
    int getScore() const {return this->score;};
```

```
    // A setter of score. New scores lower than 0 is regarded as illegal.
    void setScore(int newScore) {
        if (newScore < 0) {
            cout << "How is that possible?" << endl;
            return;
        }
        this->score = newScore; // this pointer
    };
};
```

Initialization List

```
ClsName::ClsName() : base(..), m1(..), m2(..) {
    // Code for the some other operations need to be done during construction
}
```

- The order of initialization is **the order they are defined in the class**
- The performance (both time and memory) can be better than assigning to each values.
- A member that don't have a default constructor must be initialized in the initialization list.
- const members and references can only be initialized in the initialization list.

Const Member Functions

- A `const` qualifier after member functions promises that this member function will not modify this object.

  ```
  class Sample {
      int val;
  public:
      void setVal() const { val = 0; } // Compile error
  };
  ```

- Also, inside a `const` member function, `non-const` member functions (as well as other functions that may modify the object) **cannot be called** (to ensure that the object will not be modified).

References:

[1] Weikang, Qian. VE280 Lecture 2, 8-11, 13.

[2] Yunuo, Chen. VE280 Mid RC part 2.