规范性修改的前后对比

案例1

修改前

```
44  Vector<int> to_vector(string n){
45      Vector<int> ans = {};
46
47      for(int i = n.size()-1 ; i >=0 ; i--){
48
49          int digit = n[i]- '0';
50          ans.add(digit);
51      }
52      return ans;
53
54  }
```

修改后

```
44  Vector<int> to_vector(const string n)
45  {
46      Vector<int> digits = {};
47
48      for(int i = n.size()-1 ; i >=0 ; i--)
49      {
50          int digit = n[i]- '0';
51          digits.add(digit);
52      }
53      return digits;
54  }
55
```

案例2

修改前

```
    if(command == "Add"){
        if(is_Error(1,state,eva_stack,pc)) {pc=-2;cout <<"Error";}
        else{
        int num1 = eva_stack.pop();
        int num2 = eva_stack.pop();
        eva_stack.push(num1+num2);
        }

    }

    else if(command == "Sub"){
        if(is_Error(1,state,eva_stack,pc)) {pc=-2;cout <<"Error";}
        else{
         int num1 = eva_stack.pop();
        int num2 = eva_stack.pop();
        eva_stack.push(num2-num1);
        }

    }
```

修改后

```
    if(is_Error(1,state,eva_stack,pc))
    {
        pc=-2;
        cout <<"Error";
    }
    else
    {
        int num1 = eva_stack.pop();
        int num2 = eva_stack.pop();

        if(command == "Add")
        {
            eva_stack.push(num1+num2);
        }
        else if(command == "Sub")
        {
            eva_stack.push(num2-num1);
        }
    }
```

案例3

修改前

```
    switch(dir){
        case 1:
            d = UP;
            break;
        case 2:
            d = DOWN;
            break;
        case 3:
            d = LEFT;
            break;
        case 4:
            d = RIGHT;
            break;
    }
```

修改后

```
    switch(dir)
    {
        case 1:
            d = UP;
            break;
        case 2:
            d = DOWN;
            break;
        case 3:
            d = LEFT;
            break;
        case 4:
            d = RIGHT;
            break;
        default:
            os << "fault!"<<endl;
    }
```

三个最有帮助的规范性建议

1. “}”是否独占一行并与“{”所在行对齐于同一列。对于清晰程序的结构非常有帮助，类似于`python`中的强制缩进，增强了程序的易读性，有利于检查`{`的`}`对应关系。
2. if、for、while、do等语句自占一行，不论执行语句多少都加“{}”。
3. 程序中是否出现相同的局部变量和全部变量。个人感觉这是一类非常难发现的bug，
4. 变量命名法（众所周知三个最有帮助的规范性建议有四个）。目前用的比较多的是下划线命名法，采用规范化的变量命名法可以有效的程序中规避出现两个相同名字的变量，尤其是出现相同的局部变量和全局变量，而这一类bug往往很难发现。