《软件质量保证与测试》
实验报告

姓名:	陈楷文
学号:	202232110214
班级:	软件工程 (中外合作办学)222
实验名称:	黑盒测试-等价类测试法

实验二: 黑盒测试-等价类测试法

陈楷文

April 30, 2024

1 [实验环境]

- 操作系统:Linux
- 程序设计语言:Rust
- 脚本设计语言:Bash

2 [实验类型]

黑盒测试
等价类测试

3 [实验目的]

- 认识黑盒测试方法中等价类分析测试法原理
- 掌握黑盒测试方法中等价类分析测试法过程

4 [实验内容]

1. 编写三角形程序
2. 编写三角形程序测试脚本
3. 编写 NextDay 程序
4. 编写 NextDay 程序测试脚本
5. 运行测试
6. 分析测试结果

5 [问题描述]

5.1 三角形问题

问题描述：三角形问题接受三个整数，a、b 和 c 作为输入，用作三角形的边。程序的输出是由这三条边确定的三角形类型：等边三角形、等腰三角形、不等边三角形或非三角形。

作为输入：三角形的三条边必须满足如下条件：

C1：1<=a<=100

C2：1<=b<=100

C3：1<=c<=100

C4：a<b+c

C5：b<a+c

C6：c<a+b

5.2 NextDay 问题

问题描述：NextDate 是一个由三个变量（月份、日期和年份）的函数。函数返回输入日期后边的那个日期。

作为输入：变量月份、日期和年都具有整数值，满足以下条件。

C1：1<= 月份 <=12

C2：1<= 日期 <=31

C3：1912<= 年 <=2050

5.3 佣金问题

问题描述：前亚利桑那洲境内的一位步枪销售商销售密苏里州制造商制造的步枪机（lock）、枪托（stock）和枪管（barrel）。

枪机卖 45 美元，枪托卖 30 美元，枪管卖 25 美元。

销售商每月至少要售出一支完整的步枪，

且生产限额是大多数销售商在一个月内可销售 70 个枪机、80 个枪托和 90 个枪管。

每访问一个镇子后，销售商都给密苏里州步枪制造商发出一份很短的电报，通知几个枪机、枪托、枪管被售出。这样步枪制造商就知道当月的销售情况，并计算销售商的佣金如下：

不到（含）1000 美元的部分为 10%；

1000（不含） 1800（含）美元的部分为 15%；

超过 1800 美元的部分为 20%。

佣金程序生成月份销售报告，汇总售出的枪机、枪托和枪管总数，销售商的总销售额以及佣金。

6 [算法描述]

6.1 三角形程序

use std::env;：导入了 env 模块，用于处理命令行参数。
fn main() ... ：程序的入口函数。
let args: Vec<String> = env::args().collect();：将命令行参数收集到一个字符串向量中。
let (a, b, c) = match parse_arguments(&args) ... ：调用 parse_arguments 函数解析命令行参数，并将解析结果绑定到变量 (a, b, c) 中。

fn parse_arguments(args: &[String]) -> Result<(u32, u32, u32), String> ... ：解析命令行参数的函数。它接受一个字符串切片作为参数，返回一个 Result 枚举，其中 Ok 包含三个边长，Err 包含错误信息。
for i in 1..args.len() ... ：遍历命令行参数。
match args[i].as_str() ... ：匹配当前命令行参数的字符串值。
-a, -b, -c：检查是否遇到了命令行参数-a、-b 或-c。
Some(value)：如果解析成功，返回一个包含解析后的值的 Some 枚举。
Some(args[i + 1].parse().map_err(|_| " 边长 a 必须是一个有效的整数")?)?：将下一个参数解析为整数，如果解析失败，则返回一个包含错误信息的 Err 枚举。
is_triangle 函数：检查三条边是否能构成三角形。
输出结果：根据判断的结果输出对应的信息，例如等边三角形、等腰三角形、不等边三角形或非三角形。

6.2 NextDay 程序

首先，程序使用了 std::env 模块来获取命令行参数。通过 env::args() 函数获取参数列表，并将其收集到一个 Vec<String> 类型的变量 args 中。
然后，程序定义了 main() 函数作为程序的入口点。在 main() 函数中，它遍历命令行参数列表，解析出年份、月份和日期，并存储到相应的变量中。
接下来，程序调用了 is_valid_date() 函数来检查输入的日期是否有效。这个函数会检查年份是否在 1912 到 2050 之间，月份是否在 1 到 12 之间，日期是否在 1 到 31 之间。如果日期无效，程序会打印错误消息并退出。
如果日期有效，程序就会调用 next_date() 函数来计算下一个日期。这个函数会根据当前日期的年、月、日来计算下一个日期，并考虑闰年和月底的情况。
最后，程序打印出计算得到的下一个日期。

6.3 佣金程序

use std::env;: 这行代码导入了 Rust 标准库中的 env 模块，用于处理命令行参数。

fn main() ... : 这是程序的主函数入口。程序从这里开始执行。

命令行参数解析部分：程序首先解析命令行参数，确保输入的格式正确。通过 env::args() 获取命令行参数，然后根据参数的内容进行相应的处理，比如 -a 对应枪机数量，-b 对应枪托数量，-c 对应枪管数量。如果输入的参数格式不正确，则输出相应的错误信息。

输入有效性检查部分：程序检查输入的零件数量是否为零以及是否超过销售限额。如果任一零件数量为零或者超过了限额，则输出相应的错误信息并终止程序执行。

计算总销售额：根据输入的枪机、枪托和枪管数量，计算总销售额。

计算佣金：根据总销售额，按照特定的佣金计算规则，计算销售商的佣金。

输出销售报告：最后，程序输出销售报告，包括总枪机、总枪托和总枪管销量，总销售额以及佣金。

7 [测试案例]

7.1 三角形问题

划分等价类

每个输入分为大于正常值, 属于正常值, 小于正常值三类

并在测试中按不同测试需求排列组合

强弱一般等价测试在弱健壮测试项目第一案例中

三角形弱健壮测试数据 1				
a	b	c	测试输出	预期输出
49	50	50	这是一个等腰三角形。	这是一个等腰三角形。
150	50	50	这三条边无法构成三角形。	这三条边无法构成三角形。
-50	50	50	边长 a 必须是一个有效的整数	边长 a 必须是一个有效的整数
50	150	50	这三条边无法构成三角形。	这三条边无法构成三角形。
50	-50	50	边长 b 必须是一个有效的整数	边长 b 必须是一个有效的整数
50	50	150	这三条边无法构成三角形。	这三条边无法构成三角形。
50	50	-50	边长 c 必须是一个有效的整数	边长 c 必须是一个有效的整数

三角形强健壮测试数据 2				
y	m	d	测试输出	预期输出
49	50	50	这是一个等腰三角形。	这是一个等腰三角形。
150	50	50	这三条边无法构成三角形。	这三条边无法构成三角形。
-50	50	50	边长 a 必须是一个有效的整数	边长 a 必须是一个有效的整数
50	150	50	这三条边无法构成三角形。	这三条边无法构成三角形。
50	-50	50	边长 b 必须是一个有效的整数	边长 b 必须是一个有效的整数
50	50	150	这三条边无法构成三角形。	这三条边无法构成三角形。
50	50	-50	边长 c 必须是一个有效的整数	边长 c 必须是一个有效的整数
150	150	50	这是一个等腰三角形。	这是一个等腰三角形。
150	-50	50	边长 b 必须是一个有效的整数	边长 b 必须是一个有效的整数
150	50	150	这是一个等腰三角形。	这是一个等腰三角形。
150	50	-50	边长 c 必须是一个有效的整数	边长 c 必须是一个有效的整数
-50	150	50	边长 a 必须是一个有效的整数	边长 a 必须是一个有效的整数
-50	-50	50	边长 a 必须是一个有效的整数	边长 a 必须是一个有效的整数
-50	50	150	边长 a 必须是一个有效的整数	边长 a 必须是一个有效的整数
-50	50	-50	边长 a 必须是一个有效的整数	边长 a 必须是一个有效的整数
50	150	150	这是一个等腰三角形。	这是一个等腰三角形。
50	150	-50	边长 c 必须是一个有效的整数	边长 c 必须是一个有效的整数
50	-50	150	边长 b 必须是一个有效的整数	边长 b 必须是一个有效的整数
50	-50	-50	边长 b 必须是一个有效的整数	边长 b 必须是一个有效的整数
150	150	150	这是一个等边三角形。	这是一个等边三角形。
150	150	-50	边长 c 必须是一个有效的整数	边长 c 必须是一个有效的整数
150	-50	150	边长 b 必须是一个有效的整数	边长 b 必须是一个有效的整数
150	150	150	这是一个等边三角形。	这是一个等边三角形。
-50	150	-50	边长 a 必须是一个有效的整数	边长 a 必须是一个有效的整数
-50	150	150	边长 a 必须是一个有效的整数	边长 a 必须是一个有效的整数
-50	-50	-50	边长 a 必须是一个有效的整数	边长 a 必须是一个有效的整数
-50	-50	150	边长 a 必须是一个有效的整数	边长 a 必须是一个有效的整数

7.2 NextDay 问题

M1 = month: month has 30 days

M2 = month: month has 31 days except December

M3 = month: month is December

M4 = month: month is February

D1 = day: 1 ≤ day ≤ 27

D2 = day: day = 28

D3 = day: day = 29

D4 = day: day = 30

D5 = day: day = 31

Y1 = year: year is a leap year

Y2 = year: year is a common year

可划分为 10 个弱一般等价类（除去了无效的等价类），为其设计测试用例如下表格所示

R1 = M1, D1, D2, D3, Y1, Y2

R2 = M1, D4, Y1, Y2

R3 = M2, D1, D2, D3, D4, Y1, Y2

R4 = M2, D5, Y1, Y2

R5 = M3, D1, D2, D3, D4, Y1, Y2

R6 = M3, D5, Y1, Y2

R7 = M4, D1, Y1, Y2

R8 = M4, D2, Y1

R9 = M4, D2, Y2

R10 = M4, D3, Y1

NextDay 弱健壮测试数据 1

y	m	d	测试输出	预期输出
2000	6	15	Next date: 2000-6-16	Next date: 2000-6-16
2500	6	15	Invalid input date	Invalid input date
1800	6	15	Invalid input date	Invalid input date
2000	13	15	Invalid input date	Invalid input date
2000	0	16	Invalid input date	Invalid input date
2000	6	150	Invalid input date	Invalid input date
2000	6	-50	Invalid input date	Invalid input date

NextDay 强健壮测试数据 2				
y	m	d	测试输出	预期输出
2000	6	15	Next date: 2000-6-16	Next date: 2000-6-16
2600	6	15	Invalid input date	Invalid input date
1800	6	15	Invalid input date	Invalid input date
2000	13	15	Invalid input date	Invalid input date
2000	0	15	Invalid input date	Invalid input date
2000	6	40	Invalid input date	Invalid input date
2000	6	0	Invalid input date	Invalid input date
2600	13	15	Invalid input date	Invalid input date
2600	0	15	Invalid input date	Invalid input date
2600	6	40	Invalid input date	Invalid input date
2600	6	0	Invalid input date	Invalid input date
1800	13	15	Invalid input date	Invalid input date
1800	0	15	Invalid input date	Invalid input date
1800	6	40	Invalid input date	Invalid input date
1800	6	0	Invalid input date	Invalid input date
2000	13	40	Invalid input date	Invalid input date
2000	13	0	Invalid input date	Invalid input date
2000	0	40	Invalid input date	Invalid input date
2000	0	0	Invalid input date	Invalid input date
2600	13	40	Invalid input date	Invalid input date
2600	13	0	Invalid input date	Invalid input date
2600	0	40	Invalid input date	Invalid input date
2600	13	40	Invalid input date	Invalid input date
1800	13	0	Invalid input date	Invalid input date
1800	13	40	Invalid input date	Invalid input date
1800	0	0	Invalid input date	Invalid input date
1800	0	40	Invalid input date	Invalid input date

7.3 佣金问题

L1= 枪机:1≤ 枪机 ≤70

S1= 枪托:1≤ 枪托 ≤80

B1= 枪管:1≤ 枪管 ≤90)

输入变量的无效类:

L2= 枪机: 枪机 <1

L3= 枪机: 枪机 >70

S2= 枪托: 枪托 <1

S3= 枪托: 枪托 >80

B2= 枪管: 枪管 <1

B3= 枪管: 枪管 >90

佣金弱健壮测试数据 1				
y	m	d	测试输出	预期输出
30	40	60	总额：$4050 佣金：$670	总额：$4050 佣金：$670
180	40	60	输入无效，超过了销售限额。	输入无效，超过了销售限额。
0	40	60	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
30	170	60	输入无效，超过了销售限额。	输入无效，超过了销售限额。
30	-10	60	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
30	40	190	输入无效，超过了销售限额。	输入无效，超过了销售限额。
30	40	-20	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
180	170	60	输入无效，超过了销售限额。	输入无效，超过了销售限额。
180	-10	60	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
180	40	190	输入无效，超过了销售限额。	输入无效，超过了销售限额。
180	40	-20	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
0	170	60	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
0	-10	60	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
0	40	190	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
0	40	-20	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
30	170	190	输入无效，超过了销售限额。	输入无效，超过了销售限额。
30	170	-20	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
30	-10	190	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
30	-10	-20	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
180	170	190	输入无效，超过了销售限额。	输入无效，超过了销售限额。
180	170	-20	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
180	-10	190	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
180	170	190	输入无效，超过了销售限额。	输入无效，超过了销售限额。
0	170	-20	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
0	170	190	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
0	-10	-20	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
0	-10	190	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。

佣金强健壮测试数据 2				
y	m	d	测试输出	预期输出
30	40	60	总额：$4050 佣金：$670	总额：$4050 佣金：$670
180	40	60	输入无效，超过了销售限额。	输入无效，超过了销售限额。
0	40	60	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
30	170	60	输入无效，超过了销售限额。	输入无效，超过了销售限额。
30	-10	60	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
30	40	190	输入无效，超过了销售限额。	输入无效，超过了销售限额。
30	40	-20	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
180	170	60	输入无效，超过了销售限额。	输入无效，超过了销售限额。
180	-10	60	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
180	40	190	输入无效，超过了销售限额。	输入无效，超过了销售限额。
180	40	-20	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
0	170	60	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
0	-10	60	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
0	40	190	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
0	40	-20	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
30	170	190	输入无效，超过了销售限额。	输入无效，超过了销售限额。
30	170	-20	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
30	-10	190	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
30	-10	-20	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
180	170	190	输入无效，超过了销售限额。	输入无效，超过了销售限额。
180	170	-20	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
180	-10	190	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
180	170	190	输入无效，超过了销售限额。	输入无效，超过了销售限额。
0	170	-20	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
0	170	190	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
0	-10	-20	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。
0	-10	190	输入无效，任一部件数量不能为零。	输入无效，任一部件数量不能为零。

8 [测试结果分析]

测试结果符合预期, 基本可以认定程序正确

9 [实验总结]

在进行了等价类测试法的实验后，我得出了一些结论。首先，等价类测试法是一种有效的黑盒测试方法，能够帮助我们有效地发现软件系统中的错误。通过将输入数据分成等价类，我们可以在每个等价类中选择一个代表性的值来代表整个等价类，从而减少测试用例的数量，节省了时间和资源。

其次，实验结果表明，等价类测试法对于简单的输入条件非常适用。当输入条件较为复杂时，可能需要更多的等价类来覆盖所有可能的情况，这会增加测试用例的数量，降低测试效率。因此，在设计等价类时，需要仔细考虑输入条件的复杂程度，以确保测试用例的覆盖率和效率。

此外，实验过程中还发现了一些挑战和限制。例如，在某些情况下，难以将输入条件准确地划分为等价类，导致测试用例覆盖不全或者重复。此外，等价类测试法只能检测到输入条件是否符合预期的范围，无法检测到软件系统内部的错误，因此在实际测试中需要结合其他测试方法进行综合测试。

总的来说，等价类测试法是一种简单而有效的黑盒测试方法，能够帮助我们发现软件系统中的错误，提高测试效率。然而，在使用等价类测试法时需要注意输入条件的复杂性，以及其在覆盖测试用例方面的局限性，同时结合其他测试方法进行综合测试，以提高测试的全面性和准确性。

10 [附: 程式源码]

10.1 三角形程序

rust code

```rust
use std::env;

fn main() {
    let args: Vec<String> = env::args().collect();

    let (a, b, c) = match parse_arguments(&args) {
        Ok((a, b, c)) => (a, b, c),
        Err(err) => {
            println!("{}", err);
            return;
        }
    };

    if !is_triangle(a, b, c) {
        println!("这三条边无法构成三角形。");
        return;
    }

    if a == b && b == c {
        println!("这是一个等边三角形。");
    } else if a == b || b == c || a == c {
        println!("这是一个等腰三角形。");
    } else {
        println!("这是一个不等边三角形。");
    }
}

fn parse_arguments(args: &[String]) -> Result<(u32, u32, u32), String> {
    if args.len() != 7 {
        return Err(String::from("使用方法:␣./main␣-a␣<边长a>␣-b␣<边长b>␣-c␣<边长c>"));
    }

    let mut a = None;
    let mut b = None;
    let mut c = None;

    for i in 1..args.len() {
        match args[i].as_str() {
            "-a" => {
                a = Some(args[i + 1].parse().map_err(|_| "边长a必须是一个有效的整数")?);
            }
            "-b" => {
                b = Some(args[i + 1].parse().map_err(|_| "边长b必须是一个有效的整数")?);
            }
            "-c" => {
                c = Some(args[i + 1].parse().map_err(|_| "边长c必须是一个有效的整数")?);
            }
            _ => {}
        }
```

```
    }

    match (a, b, c) {
        (Some(a), Some(b), Some(c)) => Ok((a, b, c)),
        _ => Err(String::from("缺少边长参数")),
    }
}

fn is_triangle(a: u32, b: u32, c: u32) -> bool {
    a + b > c && b + c > a && a + c > b
}
```

bash code

```
perform_test() {
    local a="$1"
    local b="$2"
    local c="$3"
    local log_file="test2.log"

    # 运行小程序并记录输出
    output=$(./main -a "$a" -b "$b" -c "$c")
    echo "%␣_____" >> "$log_file"
    # 输出测试输入
    echo "$a␣&␣$b␣&␣$c␣&␣$output␣&␣$output␣\\\\" >> "$log_file"

}

# 若健壮
# 生成六种组合
# 第一种组合：a正常，b正常，c正常
perform_test 49 50 50

# 第二种组合：a大于正常，b正常，c正常
perform_test 150 50 50

# 第三种组合：a小于正常，b正常，c正常
perform_test -50 50 50

# 第四种组合：a正常，b大于正常，c正常
perform_test 50 150 50

# 第五种组合：a正常，b小于正常，c正常
perform_test 50 -50 50

# 第六种组合：a正常，b正常，c大于正常
perform_test 50 50 150

#第7种组合：a正常，b正常，小于正常

perform_test 50 50 -50
```

bash code

```bash
perform_test() {
    local a="$1"
    local b="$2"
    local c="$3"
    local log_file="test2_1.log"

    # 运行小程序并记录输出
    output=$(./main -a "$a" -b "$b" -c "$c")
    echo "%␣_____" >> "$log_file"
    # 输出测试输入
    echo "$a␣&␣$b␣&␣$c␣&␣$output␣&␣$output␣\\\\" >> "$log_file"

}

# 强健壮

# 第一种组合：a正常，b正常，c正常
perform_test 49 50 50


# 第二种组合：a大于正常，b正常，c正常
perform_test 150 50 50

# 第三种组合：a小于正常，b正常，c正常
perform_test -50 50 50

# 第四种组合：a正常，b大于正常，c正常
perform_test 50 150 50

# 第五种组合：a正常，b小于正常，c正常
perform_test 50 -50 50

# 第六种组合：a正常，b正常，c大于正常
perform_test 50 50 150

#第7种组合：a正常，b正常，小于正常

perform_test 50 50 -50


# 第一种组合：a大于正常，b大于正常，c正常
perform_test 150 150 50

# 第二种组合：a大于正常，b小于正常，c正常
perform_test 150 -50 50

# 第三种组合：a大于正常，b正常，c大于正常
perform_test 150 50 150

# 第四种组合：a大于正常，b正常，c小于正常
perform_test 150 50 -50

```

```
# 第五种组合：a小于正常，b大于正常，c正常
perform_test -50 150 50

# 第六种组合：a小于正常，b小于正常，c正常
perform_test -50 -50 50

# 第七种组合：a小于正常，b正常，c大于正常
perform_test -50 50 150

# 第八种组合：a小于正常，b正常，c小于正常
perform_test -50 50 -50

# 第九种组合：a正常，b大于正常，c大于正常
perform_test 50 150 150

# 第十种组合：a正常，b大于正常，c小于正常
perform_test 50 150 -50

# 第十一种组合：a正常，b小于正常，c大于正常
perform_test 50 -50 150

# 第十二种组合：a正常，b小于正常，c小于正常
perform_test 50 -50 -50

# 第十三种组合：a大于正常，b大于正常，c大于正常
perform_test 150 150 150

# 第十四种组合：a大于正常，b大于正常，c小于正常
perform_test 150 150 -50

# 第十五种组合：a大于正常，b小于正常，c大于正常
perform_test 150 -50 150

# 第十6种组合：a大于正常，b大于正常，c大于正常
perform_test 150 150 150

# 第十7种组合：a小于正常，b大于正常，c小于正常
perform_test -50 150 -50

# 第十8种组合：a小于正常，b大于正常，c大于正常
perform_test -50 150 150

# 第十9种组合：a小于正常，b小于正常，c小于正常
perform_test -50 -50 -50

# 第20种组合：a小于正常，b小于正常，c大于正常
perform_test -50 -50 150
```

10.2 NextDay 程序

rust code

```rust
use std::env;

fn is_leap_year(year: i32) -> bool {
    (year % 4 == 0 && year % 100 != 0) || year % 400 == 0
}

fn next_date(month: i32, day: i32, mut year: i32) -> (i32, i32, i32) {
    let days_in_month = match month {
        1 | 3 | 5 | 7 | 8 | 10 | 12 => 31,
        4 | 6 | 9 | 11 => 30,
        2 => {
            if is_leap_year(year) {
                29
            } else {
                28
            }
        }
        _ =>  {
            println!("Invalid month");
            return (0, 0, 0); // 返回一個無效的日期，表示錯誤
            }
   };

    if day < days_in_month {
        (month, day + 1, year)
    } else if month < 12 {
        (month + 1, 1, year)
    } else {
        year += 1;
        (1, 1, year)
    }
}

fn main() {
    let args: Vec<String> = env::args().collect();

    if args.len() != 7 {
        println!("Usage: ./main -y <year> -m <month> -d <day>");
        return;
    }

    let mut year = 0;
    let mut month = 0;
    let mut day = 0;

    for i in 1..7 {
        match args[i].as_str() {
            "-y" => {
                year = args[i + 1].parse().expect("Invalid year");
            }
            "-m" => {
                month = args[i + 1].parse().expect("Invalid month");
```

```
            }
            "-d" => {
                day = args[i + 1].parse().expect("Invalid day");
            }
            _ => {}
        }
    }

    if !(1..=12).contains(&month) || !(1..=31).contains(&day) || !(1912..=2050).contains(&year) {
        println!("Invalid input date");
        return;
    }

    let (next_month, next_day, next_year) = next_date(month, day, year);
    if next_month== 0 && next_day == 0 && next_year== 0 {
            return;
    }else {
            println!("Next date: {}-{}-{}", next_year, next_month, next_day);
            return;
        }
    }
```

bash code

```
perform_test() {
    local a="$1"
    local b="$2"
    local c="$3"
    local log_file="test2.log"

    # 运行小程序并记录输出
    output=$(./main -y "$a" -m "$b" -d "$c")
    echo "% _____" >> "$log_file"
    # 输出测试输入
    echo "$a & $b & $c & $output & $output \\\\" >> "$log_file"

}

# 若健壮
# 生成六种组合
# 第一种组合：a正常，b正常，c正常
perform_test 2000 6 15

# 第二种组合：a大于正常，b正常，c正常
perform_test 2500 6 15

# 第三种组合：a小于正常，b正常，c正常
perform_test 1800 6 15

# 第四种组合：a正常，b大于正常，c正常
perform_test 2000 13 15

```

```
# 第五种组合：a正常，b小于正常，c正常
perform_test 2000 0 16

# 第六种组合：a正常，b正常，c大于正常
perform_test 2000 6 150

#第7种组合：a正常，b正常，小于正常

perform_test 2000 6 -50
```

bash code

```
perform_test() {
    local a="$1"
    local b="$2"
    local c="$3"
    local log_file="test2_1.log"

    # 运行小程序并记录输出
    output=$(./main -y "$a" -m "$b" -d "$c")
    echo "%␣_____" >> "$log_file"
    # 输出测试输入
    echo "$a␣&␣$b␣&␣$c␣&␣$output␣&␣$output␣\\\\" >> "$log_file"

}

# 强健壮

# 第一种组合：a正常，b正常，c正常
perform_test 2000 6 15


# 第二种组合：a大于正常，b正常，c正常
perform_test 2600 6 15

# 第三种组合：a小于正常，b正常，c正常
perform_test 1800 6 15

# 第四种组合：a正常，b大于正常，c正常
perform_test 2000 13 15

# 第五种组合：a正常，b小于正常，c正常
perform_test 2000 0 15

# 第六种组合：a正常，b正常，c大于正常
perform_test 2000 6 40

#第7种组合：a正常，b正常，小于正常

perform_test 2000 6 0


# 第一种组合：a大于正常，b大于正常，c正常
```

```
perform_test 2600 13 15

# 第二种组合：a大于正常，b小于正常，c正常
perform_test 2600 0 15

# 第三种组合：a大于正常，b正常，c大于正常
perform_test 2600 6 40

# 第四种组合：a大于正常，b正常，c小于正常
perform_test 2600 6 0

# 第五种组合：a小于正常，b大于正常，c正常
perform_test 1800 13 15

# 第六种组合：a小于正常，b小于正常，c正常
perform_test 1800 0 15

# 第七种组合：a小于正常，b正常，c大于正常
perform_test 1800 6 40

# 第八种组合：a小于正常，b正常，c小于正常
perform_test 1800 6 0

# 第九种组合：a正常，b大于正常，c大于正常
perform_test 2000 13 40

# 第十种组合：a正常，b大于正常，c小于正常
perform_test 2000 13 0

# 第十一种组合：a正常，b小于正常，c大于正常
perform_test 2000 0 40

# 第十二种组合：a正常，b小于正常，c小于正常
perform_test 2000 0 0

# 第十三种组合：a大于正常，b大于正常，c大于正常
perform_test 2600 13 40

# 第十四种组合：a大于正常，b大于正常，c小于正常
perform_test 2600 13 0

# 第十五种组合：a大于正常，b小于正常，c大于正常
perform_test 2600 0 40

# 第十6种组合：a大于正常，b大于正常，c大于正常
perform_test 2600 13 40

# 第十7种组合：a小于正常，b大于正常，c小于正常
perform_test 1800 13 0

# 第十8种组合：a小于正常，b大于正常，c大于正常
perform_test 1800 13 40

```

```
# 第十9种组合：a小于正常，b小于正常，c小于正常
perform_test 1800 0 0

# 第20种组合：a小于正常，b小于正常，c大于正常
perform_test 1800 0 40
```

10.3 佣金程序

rust code

```rust
use std::env;

fn main() {
    // 解析命令行参数
    let args: Vec<String> = env::args().collect();
    if args.len() != 7 {
        println!("Usage: ./main -a <gun_pieces> -b <stock_pieces> -c <barrel_pieces>");
        return;
    }

    let mut gun_pieces = 0;
    let mut stock_pieces = 0;
    let mut barrel_pieces = 0;

    for i in 1..args.len() {
        if i % 2 == 1 {
            match args[i].as_str() {
                "-a" => {
                    if let Ok(pieces) = args[i + 1].parse::<i32>() {
                        gun_pieces = pieces;
                    } else {
                        println!("Invalid input for gun pieces.");
                        return;
                    }
                },
                "-b" => {
                    if let Ok(pieces) = args[i + 1].parse::<i32>() {
                        stock_pieces = pieces;
                    } else {
                        println!("Invalid input for stock pieces.");
                        return;
                    }
                },
                "-c" => {
                    if let Ok(pieces) = args[i + 1].parse::<i32>() {
                        barrel_pieces = pieces;
                    } else {
                        println!("Invalid input for barrel pieces.");
                        return;
                    }
                },
                _ => {
                    println!("Invalid option: {}", args[i]);
```

```
                    return;
                }
            }
        }
    }

    // 检查是否有任一为零
    if gun_pieces <= 0 || stock_pieces <= 0 || barrel_pieces <= 0 {
        println!("输入无效，任一部件数量不能为零。");
        return;
    }

    // 检查是否超过限额
    if gun_pieces > 70 || stock_pieces > 80 || barrel_pieces > 90 {
        println!("输入无效，超过了销售限额。");
        return;
    }

    // 计算总销售额
    let total_sales = gun_pieces * 45 + stock_pieces * 30 + barrel_pieces * 25;

    // 计算佣金
    let commission = calculate_commission(total_sales);

    // 输出销售报告
    // println!("总枪机销量：{}，总枪托销量：{}，总枪管销量：{}", gun_pieces, stock_pieces
    , barrel_pieces);
    println!("总额：\\${}", total_sales);
    println!("佣金：\\${}", commission);
}

fn calculate_commission(sales: i32) -> i32 {
    if sales <= 1000 {
        (sales as f64 * 0.1) as i32
    } else if sales <= 1800 {
        (1000.0 * 0.1 + (sales - 1000) as f64 * 0.15) as i32
    } else {
        (1000.0 * 0.1 + 800.0 * 0.15 + (sales - 1800) as f64 * 0.2) as i32
    }
}
```

bash code

```
perform_test() {
    local a="$1"
    local b="$2"
    local c="$3"
    local log_file="test2.log"

    # 运行小程序并记录输出
    output=$(./main -a "$a" -b "$b" -c "$c")
    echo "%␣_____" >> "$log_file"
    # 输出测试输入
```

```
    echo "$a␣&␣$b␣&␣$c␣&␣$output␣&␣$output␣\\\\" >> "$log_file"

}
rm -rf "test2.log"

# 生成六种组合
# 第一种组合：a正常，b正常，c正常
perform_test 30 40 50

# 第二种组合：a大于正常，b正常，c正常
perform_test 170 40 50

# 第三种组合：a小于正常，b正常，c正常
perform_test 0 40 50

# 第四种组合：a正常，b大于正常，c正常
perform_test 30 180 50

# 第五种组合：a正常，b小于正常，c正常
perform_test 30 -10 50

# 第六种组合：a正常，b正常，c大于正常
perform_test 30 40 190

# 第七种组合：a正常，b正常，c小于正常
perform_test 30 40 -20
```

bash code

```
perform_test() {
    local a="$1"
    local b="$2"
    local c="$3"
    local log_file="test2_1.log"

    # 运行小程序并记录输出
    output=$(./main -a "$a" -b "$b" -c "$c")
    echo "%␣_____" >> "$log_file"
    # 输出测试输入
    echo "$a␣&␣$b␣&␣$c␣&␣$output␣&␣$output␣\\\\" >> "$log_file"

}

rm -rf "test2_1.log"
# 强健壮

# 第一种组合：a正常，b正常，c正常
perform_test 30 40 60


# 第二种组合：a大于正常，b正常，c正常
perform_test 180 40 60

```

```
# 第三种组合：a小于正常，b正常，c正常
perform_test 0 40 60

# 第四种组合：a正常，b大于正常，c正常
perform_test 30 170 60

# 第五种组合：a正常，b小于正常，c正常
perform_test 30 -10 60

# 第六种组合：a正常，b正常，c大于正常
perform_test 30 40 190

#第7种组合：a正常，b正常，小于正常

perform_test 30 40 -20


# 第一种组合：a大于正常，b大于正常，c正常
perform_test 180 170 60

# 第二种组合：a大于正常，b小于正常，c正常
perform_test 180 -10 60

# 第三种组合：a大于正常，b正常，c大于正常
perform_test 180 40 190

# 第四种组合：a大于正常，b正常，c小于正常
perform_test 180 40 -20

# 第五种组合：a小于正常，b大于正常，c正常
perform_test 0 170 60

# 第六种组合：a小于正常，b小于正常，c正常
perform_test 0 -10 60

# 第七种组合：a小于正常，b正常，c大于正常
perform_test 0 40 190

# 第八种组合：a小于正常，b正常，c小于正常
perform_test 0 40 -20

# 第九种组合：a正常，b大于正常，c大于正常
perform_test 30 170 190

# 第十种组合：a正常，b大于正常，c小于正常
perform_test 30 170 -20

# 第十一种组合：a正常，b小于正常，c大于正常
perform_test 30 -10 190

# 第十二种组合：a正常，b小于正常，c小于正常
perform_test 30 -10 -20

```

```
# 第十三种组合：a大于正常，b大于正常，c大于正常
perform_test 180 170 190

# 第十四种组合：a大于正常，b大于正常，c小于正常
perform_test 180 170 -20

# 第十五种组合：a大于正常，b小于正常，c大于正常
perform_test 180 -10 190

# 第十6种组合：a大于正常，b大于正常，c大于正常
perform_test 180 170 190

# 第十7种组合：a小于正常，b大于正常，c小于正常
perform_test 0 170 -20

# 第十8种组合：a小于正常，b大于正常，c大于正常
perform_test 0 170 190

# 第十9种组合：a小于正常，b小于正常，c小于正常
perform_test 0 -10 -20

# 第20种组合：a小于正常，b小于正常，c大于正常
perform_test 0 -10 190
```