testing

Лучше полдня потерять, зато потом за 5 минут долететь.
Мотивация:

- избегать ошибок
- быстро находить ошибки при изменении кода
- улучшить структуру кода

testing

Лучше полдня потерять, зато потом за 5 минут долететь.
Мотивация:

- избегать ошибок
- быстро находить ошибки при изменении кода
- улучшить структуру кода

Требования:

- тест должен быть быстрым
- каждый тест тестирует одно свойство поведения

testing

```
class Reader {
    int fd;
public:
    Reader(int fd) : fd(fd) {}
    string GetString() {
        char buf[BUF_SIZE];
        int off = 0, readed;
        while (readed = read(fd, buf, CHUNK_SIZE) > 0) {
            off += readed;
        }
        return buf;
    }
}
```

testing

```
TEST(ReaderTests, read) {
    int fd =  open("input.txt", O_RDONLY);
    Reader r(fd);
    ASSERT_EQ(r.GetString(), "123");
}
```

FAIL

testing

```
class Reader {
    int fd;
public:
    Reader(int fd) : fd(fd) {}
    string GetString() {
        char buf[BUF_SIZE];
        int off = 0, readed;
        while (readed = read(fd, buf, CHUNK_SIZE) > 0) {
            off += readed;
        }
        buf[off] = '\0';
        return buf;
    }
}
```

FAIL

testing

```
class Reader {
    int fd;
public:
    Reader(int fd) : fd(fd) {}
    string GetString() {
        char buf[BUF_SIZE];
        int off = 0, readed;
        while ((readed = read(fd, buf, CHUNK_SIZE)) > 0) {
            off += readed;
        }
        buf[off] = '\0';
        return buf;
    }
}
```

OK, but hmm...

testing

```
class IStream {
public:
    virtual int read(char* buf, int cnt) = 0;
};
class FdStream : public IStream {
    int fd;
public:
    FdStream(std::string filename) { fd = open(filename.c_str(),
     O_RDONLY);    }
    virtual int read(char* buf, int cnt) { return ::read(fd, buf, cnt);
        }
};
class Reader {
    IStream &stream;
public:
    Reader(IStream &stream) : stream(stream) {}
    std::string GetString() {
        char buf[100];
        int off = 0, readed;
        while ((readed = stream.read(buf, 10)) > 0) { off += readed; }
        buf[off] = '\0';
        return buf;
    }
};
```

OK

testing

```
class MockStream : public IStream {
public:
    MOCK_METHOD2(read, int (char *buf, int cnt));
};

TEST(ReaderTests, mock) {
    std::string data = "123";
    MockStream stream;
    EXPECT_CALL(stream, read(_,
     _)).WillOnce(Return(10)).WillOnce(Return(0));
    Reader r(stream);
}
```

OK, but hmm...

testing

```
class MockStream : public IStream {
public:
    MOCK_METHOD2(read, int (char *buf, int cnt));
};

TEST(ReaderTests, mock) {
    std::string data = "123";
    MockStream stream;
    EXPECT_CALL(stream, read(_,
     Ge(5))).WillOnce(Return(10)).WillOnce(Return(0));
    Reader r(stream);
}
```

OK, but hmm...

testing

```
class MockStream : public IStream {
public:
    MOCK_METHOD2(read, int (char *buf, int cnt));
};

TEST(ReaderTests, mock) {
    std::string data = "123";
    MockStream stream;
    EXPECT_CALL(stream, read(_, Ge(5))).WillOnce(DoAll(CopyData(data),
     Return(10))).WillOnce(Return(0));
    Reader r(stream);
}
```

testing

```
class MockStream : public IStream {
public:
    MOCK_METHOD2(read, int (char *buf, int cnt));
};

ACTION_P(CopyData, data) {
   std::copy(data.data(), data.data() + data.size(), arg0);
   arg0[data.size()] = '\0';
}

TEST(ReaderTests, mock) {
    std::string data = "123";
    MockStream stream;
    EXPECT_CALL(stream, read(_, Ge(5))).WillOnce(DoAll(CopyData(data),
     Return(10))).WillOnce(Return(0));
    Reader r(stream);
    ASSERT_EQ(r.GetString(), "123");
}
```

testing

Состав gmock:

- Предикаты на аргументы: совпадение, любой (_), Gt(), NotNull(), группировка предикатов, предикаты для контейнеров, пользовательские

testing

Состав gmock:

- Предикаты на аргументы: совпадение, любой (_), Gt(), NotNull(), группировка предикатов, предикаты для контейнеров, пользовательские
- Ограничение на число вызовов Times(2), Times(Between(2,5))

testing

Состав gmock:

- Предикаты на аргументы: совпадение, любой (_), Gt(), NotNull(), группировка предикатов, предикаты для контейнеров, пользовательские
- Ограничение на число вызовов Times(2), Times(Between(2,5))
- Задание последовательности вызовов

testing

```
{
  InSequence s;
  EXPECT_CALL(mock, write(_, _));
  EXPECT_CALL(mock, flush());
}

Sequence s1, s2;
EXPECT_CALL(mock, init()).InSequence(s1,s2);
EXPECT_CALL(mock, put(_)).InSequence(s1);
EXPECT_CALL(mock, get()).InSequence(s2);
```

testing

Состав gmock:

- Предикаты на аргументы: совпадение, любой (_), Gt(), NotNull(), группировка предикатов, предикаты для контейнеров, пользовательские
- Ограничение на число вызовов Times(2), Times(Between(2,5))
- Задание последовательности вызовов
- Действия в WillOnce(), WillRepeatedly() — SetArg, Return, Invoke, DoAll, действия по умолчанию ON_CALL

testing

```
class Reader {
    IStream &stream;
public:
    Reader(IStream &stream) : stream(stream) {}
    std::string GetString() {
        char buf[100];
        int off = 0, readed;
        while ((readed = stream.read(buf, 10)) > 0) { off += readed; }
        buf[off] = '\0';
        return buf;
    }
};
```

testing

```
struct CopyBuf {
    std::string data = "11223344";
    int off = 0;
    int copy(void *buf, int cnt) {
        int ret = 0;
        for (int i = 0; i < 2 && off + i < data.size(); ++i) {
            ((char *)buf)[i] = data[off + i];
            ++ret;
        }
        off += 2;
        return ret;
    }
};


TEST(ReaderTests, mock) {
    std::string data = "11223344";
    MockStream stream;
    CopyBuf cb;
    EXPECT_CALL(stream, read(_, _)).WillRepeatedly(Invoke(&cb,
     &CopyBuf::copy));
    Reader r(stream);
    ASSERT_EQ(r.GetString(), "11223344");
}
```