Задача 4 (Сервер комментариев)

Вы подключились к команде, которая разрабатывает web-сервер комментариев. Этот сервер позволяет создавать новых пользователей, публиковать новые объявления, а также читать все объявления выбранного пользователя. Кроме того команда недавно озаботилась борьбой со спамерами. Если какой-то пользователь признаётся спамером, он блокируется, после чего ему выдаётся страница капчи, на которой он должен подтвердить, что он человек. В случае успешного ввода капчи пользователь разблокируется и получает возможность снова оставлять комментарии.

Для выявления спамеров используется довольно простой алгоритм — спамером признаётся пользователь, отправивший три комментария подряд (см. реализацию сервера в заготовке решения).

Сервер работает по протоколу HTTP и обрабатывает следующие запросы:

- POST /add_user — добавляет нового пользователя и возвращает ответ 200 OK, в теле которого содержится идентификатор вновь добавленного пользователя (см. реализацию в заготовке решения)
- POST /add_comment — извлекает из тела запроса идентификатор пользователя и новый комментарий; если пользователь признаётся спамером, возвращает 302 Found с заголовком Location: /captcha, переводя пользователя на страницу капчи, в противном случае сохраняет комментарий и возвращает 200 OK
- GET /user_comments?user_id=[id] — возвращает ответ 200 OK, в теле которого содержатся все комментарии пользователя id, разделённые переводом строки
- GET /captcha — возвращает ответ 200 OK, в теле которого содержится страница капчи (для простоты в этой задаче мы просто возвращаем вопрос, на который надо ответить пользователю, на практике это может быть полноценная HTML-страница)
- POST /checkcaptcha — извлекает из тела запроса ответ на вопрос капчи; если он верен, разблокирует пользователя и возвращает 200 OK, если нет — возвращает 302 Found с заголовком Location: /captcha
- если метод запроса не POST и не GET или путь запроса не совпадает ни с одним из вышеперечисленных, сервер отвечает 404 Not found.

Web-сервер в коде реализован с помощью класса CommentServer:

```
1  struct HttpRequest {
2  string method, path, body;
3  map<string, string> get_params;
4  };
5
6  class CommentServer {
7  public:
8  void ServeRequest(const HttpRequest& req, ostream& response_output);
9
10 private:
11 ...
12 };
```

Его метод ServeRequest принимает HTTP-запрос, обрабатывает его и записывает HTTP-ответ в выходной поток response_output (этот поток может быть привязан к сетевому соединению). При записи HTTP-ответа в выходной поток используется следующий формат:

```
HTTP/1.1 [код ответа] [комментарий]
[Заголовок 1]: [Значение заголовка 1]
[Заголовок 2]: [Значение заголовка 2]
...
[Заголовок N]: [Значение заголовка N]
<пустая строка>
[Тело ответа]
```

2

- код ответа — это 200, 302, 404 и т.д.
- комментарий — "Found", "OK", "Not found" и т.д.
- Заголовок X — имя заголовка, например, "Location"
- тело ответа — например, это содержимое страницы капчи или идентификатор вновь добавленного пользователя; при этом, если тело ответа непустое, в ответе обязательно должен присутствовать заголовок Content-Length, значение которого равно длине ответа в байтах.

Пример ответа на запрос /add_user, в котором возвращается идентификатор нового пользователя, равный 12. Content-Length равен 2, потому что "12" — это два символа:

```
HTTP/1.1 200 OK
Content-Length: 2

12

```

С нашим сервером есть проблема — иногда он ничего не отвечает на запросы, а иногда возвращает некорректно сформированные ответы. Источник этих проблем в том, что ответы формируются вручную каждый раз (см. заготовку решения). Из-за этого мы то перевод строки забыли, то добавили лишний, то в коде ответа опечатались:

```
void ServeRequest(const HttpRequest& req, ostream& os) {
if (req.method == "POST") {
if (req.path == "/add_user") {
comments_.emplace_back();
auto response = to_string(comments_.size() - 1);
os << "HTTP/1.1 200 OK\n" << "Content-Length: " << response.size() << "\n" << "\n"
<< response;
} else if (req.path == "/checkcaptcha") {
      ...
os << "HTTP/1.1  20 OK\n\n";
}
} else {
os << "HTTP/1.1 404 Not found\n\n";
}
...
}
```

Вы решили избавиться от всех проблем разом и провести следующий рефакторинг:

- разработать класс HttpResponse, который будет представлять собой HTTP-ответ; в operator << вы решили инкапсулировать формат вывода HTTP-ответа в поток
- сделать новую сигнатуру метода ServeRequest — HttpResponse ServeRequest(const HttpRequest& req), — которая на этапе компиляции будет гарантировать, что наш сервер всегда возвращает хоть какой-то ответ (если мы забудем это сделать, компилятор выдаст предупреждение "control reaches end of non-void function")
- записывать ответ сервера в выходной поток в одном единственном месте, в котором вызывается метод ServeRequest

Интерфейс класса HttpResponse вы решили сделать таким:

```
enum class HttpCode {
Ok = 200,
NotFound = 404,
Found = 302,
};

class HttpResponse {
public:
explicit HttpResponse(HttpCode code);

HttpResponse& AddHeader(string name, string value);
HttpResponse& SetContent(string a_content);
HttpResponse& SetCode(HttpCode a_code);

friend ostream& operator << (ostream& output, const HttpResponse& resp);
};
```

Методы AddHeader, SetContent и SetCode должны возвращать ссылку на себя, чтобы иметь возможность сформировать ответ в одной строке с помощью chaining'a: return HttpResponse(HttpCode::Found).AddHeader("Location", "/captcha");. Перечисление HttpCode, передаваемое в конструктор класса HttpResponse, гарантирует, что мы не ошибёмся в коде ответа.

Этот рефакторинг вам и предстоит выполнить в этой задаче. Пришлите на проверку cpp-файл, который

- содержит реализацию класса HttpResponse,
- содержит реализацию класса CommentServer с публичным методом HttpResponse ServeRequest(const HttpRequest& req).

Сервер должен реализовывать описанный выше протокол.

Уточнения к реализации класса HttpResponse:

- Методы AddHeader, SetContent, SetCode должны возвращать ссылку на объект, для которого они вызваны
- Метод AddHeader всегда добавляет к ответу новый заголовок, даже если заголовок с таким именем уже есть
- operator<< для класса HttpResponse должен выводить HTTP-ответ в формате, описанном выше в описании метода ServerRequest; при этом заголовки можно выводить в произвольном порядке. Если у HTTP-ответа есть непустое содержимое, то необходимо вывести ровно один заголовок "Content-Length"(помимо заголовков, содержащихся в HTTP-ответе), значение которого равно размеру содержимого в байтах.

Заготовка решения

comment_server.cpp