



Программирование на C++

Практическая работа №3. Консольные мини-игры (Часть 1)

Цели:

- Научиться реализовывать функции для решения конкретных задач и повышения читаемости и переиспользуемости кода.
- Научиться генерировать случайные числа.
- Научиться работать с файлами.



Задания принимаются в формате ссылки на GitHub-репозиторий отправленной мне в dot с подписанием номера выполненной практической работы.

В репозитории должен находиться исходный код вашего задания, скриншоты или видео работы ваших мини-игр.

Пример оформления репозитория:

```
https://github.com/Meirbek-dev/SQLite_CourseProject
```

Мини-игра №1: «Угадай число» (Генерация случайных чисел)

Задание

Создать консольную игру **«Угадай число»**, в которой:

1. Программа загадывает случайное число в диапазоне от 1 до 100.
 2. Пользователь вводит своё предположение.
 3. Программа сравнивает ввод с загаданным числом и сообщает:
 - Если предположение меньше или больше, чем загаданное число.
 - Если ввод неверный, предложить повторить попытку.
 4. Игра продолжается до тех пор, пока пользователь не угадает число или пока не закончатся попытки.
 5. После угадывания программа выводит поздравительное сообщение, а также количество попыток, затраченных на угадывание.
 6. После окончания игры предложить возможность сыграть ещё раз.
-

Этапы выполнения

Основные компоненты программы:

а. Подключение необходимых заголовочных файлов

```
#include <iostream>
// Для классического подхода:
#include <cstdlib>    // Для функции rand() и srand() (если ис
пользуете данную библиотеку)
#include <ctime>      // Для получения времени для seed генера
тора
// Для современного подхода:
#include <random>
```

б. Инициализация генератора случайных чисел

Классический способ (rand/srand):

```
std::srand(static_cast<unsigned int>(std::time(nullptr)));  
int secretNumber = std::rand() % 100 + 1; // Диапазон от 1 до  
100
```

Современный способ (C++11 и выше):

```
std::random_device rd;  
std::mt19937 gen(rd());  
std::uniform_int_distribution<> distrib(1, 100);  
int secretNumber = distrib(gen);
```

с. Основной игровой цикл

- Использовать цикл `while`, который продолжается до тех пор, пока пользователь не угадает число.
- После каждой итерации сравнивать введённое число и выводить подсказки:
 - Если введённое значение меньше загаданного — выводим сообщение «Загаданное число больше».
 - Если больше — «Загаданное число меньше».
- Подсчёт количества попыток.

d. Обработка ввода пользователя

- Использовать `std::cin` для ввода.
- Проверить корректность ввода, при желании обработать неверные данные (например, если введён не числовой тип).

Пояснения к коду

1. Инициализация генератора случайных чисел:

Используем `std::random_device` и `std::mt19937` для получения случайного числа в диапазоне от 1 до 100.

2. Основной игровой цикл (`while`):

Цикл продолжается, пока пользователь не угадает число. Внутри цикла осуществляется:

- Запрос пользовательского ввода.
- Проверка корректности ввода.
- Сравнение введённого значения с загаданным числом и выдача соответствующих подсказок.

3. Обработка повторной игры:

После окончания текущей игры программа спрашивает пользователя, хочет ли он сыграть снова.

4. Обработка ошибки ввода:

Если пользователь вводит не число, программа сбрасывает состояние потока и игнорирует оставшийся ввод, предлагая ввести число заново.

Вопросы:

1. Что нужно исправить, если мы захотим, чтобы программа загадывала число от -430 до 70?
 2. Как программа обрабатывает некорректный ввод пользователя?
 3. Что происходит при вводе символов вместо числа?
 4. Как реализовано ограничение игрока попытками?
 5. Как можно модифицировать программу так, чтобы после каждой неудачной попытки выводилась информация об оставшемся числе попыток?
 6. Что произойдёт, если пользователь введёт число за границами диапазона (например, 0 или 101)?
 7. Как организована возможность повторной игры после завершения текущего раунда?
-

Мини-игра №2: «Генератор историй» (Работа с файлами)

Задание

Разработать консольную игру «Генератор историй», в которой:

1. Инициализация источников данных:

Программа содержит наборы строк (массивы или векторы), каждая из которых соответствует определённой категории. Например:

- **Герой:** (например, "смелый рыцарь", "хитрый вор", "волшебник", "отважный пират", "дерзкий исследователь").
- **Место действия:** (например, "в далёком королевстве", "на заброшенной фабрике", "в густом лесу", "на просторах космоса", "у подножия гор").
- **Событие:** (например, "победил дракона", "обнаружил сокровища", "выиграл битву", "устроил бал", "раскрыл древнюю тайну").
- **Обстоятельство или сопутствующая деталь:** (например, "с волшебным мечом", "на летающем ковре", "под звуки волшебной музыки", "с удивительной силой", "в сопровождении магического существа").

2. Генерация истории:

Программа случайным образом выбирает по одному элементу из каждого набора (каждый набор должен содержать хотя бы 5 элементов) и формирует историю, например:

«Смелый рыцарь в далёком королевстве, победил дракона с волшебным мечом.»

3. Отображение результата:

Сформированная история выводится на экран.

4. Сохранение истории в файл:

После генерации истории программа должна предложить пользователю сохранить её в файл. Если пользователь выбирает «да», история

записывается в текстовый файл (например, `stories.txt`). Если файл уже существует, новая история дописывается в конец файла.

5. Возможность повторного запуска:

После генерации истории программа предлагает пользователю возможность сгенерировать новую историю или завершить работу.

Этапы выполнения

Основные компоненты программы:

а. Подключение необходимых заголовочных файлов

Используются стандартные библиотеки для ввода/вывода, работы со строками, векторами, генерации случайных чисел и работы с файлами.

```
#include <iostream>
#include <string>
#include <vector>
#include <cstdlib>      // Для rand() и srand()
#include <ctime>        // Для инициализации генератора случай
ных чисел
#include <fstream>      // Для работы с файлами
```

б. Инициализация генератора случайных чисел

Классический способ (rand/srand):

```
std::srand(static_cast<unsigned int>(std::time(nullptr)));
```

Современный способ (C++11 и выше):

```
// Пример:
#include <random>
std::random_device rd;
```

```
std::mt19937 gen(rd());  
std::uniform_int_distribution<> distrib(0, VECTOR_SIZE - 1);
```

В данном примере используется классический способ.

с. Подготовка источников данных (наборы строк)

Создайте несколько векторов или массивов строк, где каждая категория соответствует отдельному набору. Пример:

```
// Набор героев (минимум 5 элементов)  
std::vector<std::string> heroes = {  
    "...", "...", ...  
};  
  
// Набор мест действия (минимум 5 элементов)  
std::vector<std::string> places = {  
    "...", "...", ...  
};  
  
// Набор событий (минимум 5 элементов)  
std::vector<std::string> actions = {  
    "...", "...", ...  
};  
  
// Набор деталей (минимум 5 элементов)  
std::vector<std::string> details = {  
    "...", "...", ...  
};
```

d. Генерация истории

Для генерации истории:

- Выберите случайный элемент из каждого набора.

Пример выбора случайного индекса:

```
int itemIndex = std::rand() % items.size();
```

- Сформируйте итоговую историю, объединяя выбранные части:

```
std::string story = heroes[heroIndex] + " " +  
                    places[placeIndex] + ", " +  
                    actions[actionIndex] + " " +  
                    details[detailIndex] + ".";
```

```
std::cout << "\\nСгенерированная история:\\n" << story << st  
d::endl;
```

е. Сохранение истории в файл

После генерации истории программа должна запросить у пользователя, хочет ли он сохранить её в файл. Для работы с файлами используется библиотека `<fstream>`. Пример сохранения:

```
char saveChoice;  
std::cout << "\\nСохранить историю в файл (stories.txt)? (Y/  
N): ";  
std::cin >> saveChoice;  
  
if (saveChoice == 'Y' || saveChoice == 'y') {  
    // Открытие файла для дозаписи  
    std::ofstream outFile("stories.txt", std::ios::app);  
    if (outFile.is_open()) {  
        outFile << story << "\\n";  
        outFile.close();  
        std::cout << "История сохранена в файл stories.txt" <  
< std::endl;  
    } else {  
        std::cout << "Ошибка открытия файла для записи!" << s  
td::endl;
```



```
}  
}
```

f. Основной игровой цикл

Организуем цикл, который:

- Генерирует и выводит историю;
- Предлагает сохранить историю;
- Спрашивает пользователя, хочет ли он сгенерировать новую историю;
- Завершает выполнение при ответе «нет».

g. Дополнительные рекомендации и улучшения

1. Использование современного генератора случайных чисел:

Попробуйте использовать `<random>` для более надёжного генератора случайных чисел.

2. Структурирование кода:

Реализуйте отдельные функции:

- `void initializeData(...)` – для инициализации всех наборов строк.
- `std::string generateStory(...)` – для генерации истории.
- `void saveStory(const std::string&)` – для сохранения истории в файл.
- `bool askPlayAgain()` – для проверки, хочет ли пользователь сыграть ещё раз.

3. Декоративные элементы:

Можно добавить оформление (например, рамку вокруг истории или цветной текст, если поддерживаются ANSI-escape последовательностями).

Пояснения к коду

1. Подключение заголовочных файлов:

Используются библиотеки для работы со строками, векторами, генерацией случайных чисел и файловым вводом/выводом.

2. Генерация случайных чисел:

`std::srand` и `std::time` инициализируют генератор таким образом, чтобы получать разные последовательности случайных чисел при каждом запуске.

3. Источники данных:

Четыре вектора содержат фрагменты истории. Из каждого вектора случайным образом выбирается один элемент.

4. Генерация и вывод истории:

Сформированная история выводится на экран, затем пользователю предлагается сохранить её в файл (дописывается в конец файла `stories.txt`).

5. Основной цикл:


После каждого вывода история предлагается сохранить и затем запрашивается, хочет ли пользователь сгенерировать новую историю.

6. Обработка ошибок:

Добавить обработку некорректного ввода и улучшить оформление консольного интерфейса.

Ресурсы:

Diego Assencio

 <https://dassencio.org/78>

Генерация случайных чисел

<https://metanit.com/cpp/tutorial/8.2.php>

Файловые потоки. Открытие и закрытие

<https://metanit.com/cpp/tutorial/8.3.php>

Чтение и запись текстовых файлов