# Правила практикума

Данный практикум предназначен для обучения программированию. Делать это мы будем через решение практических задач. Для их решения вам потребуется применить определенные конструкции языка С++. Здесь будет дана базовая информация, минимально достаточная для написания кода. Для более глубокого изучения языка обратитесь к справочникам и учебникам.

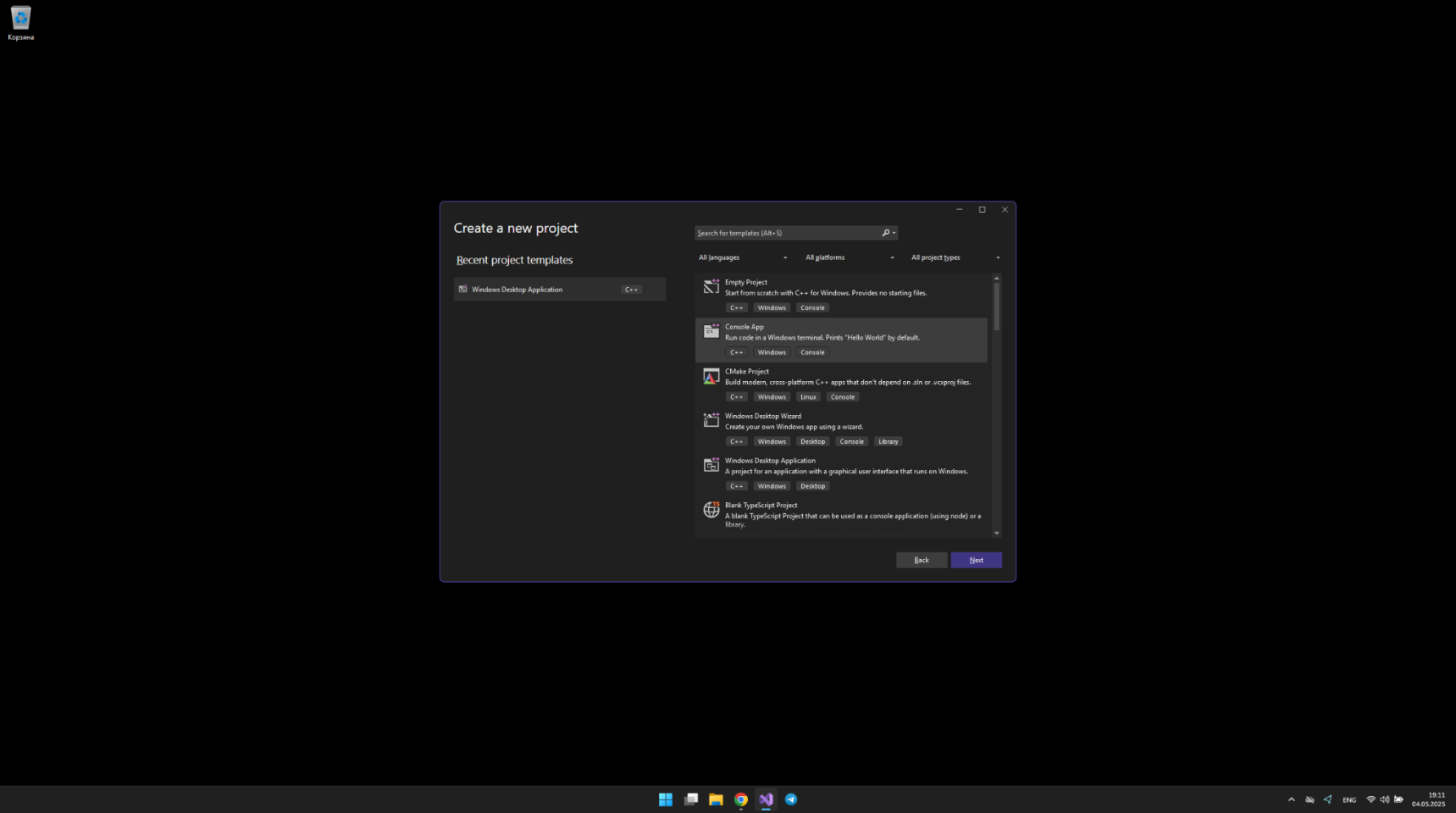
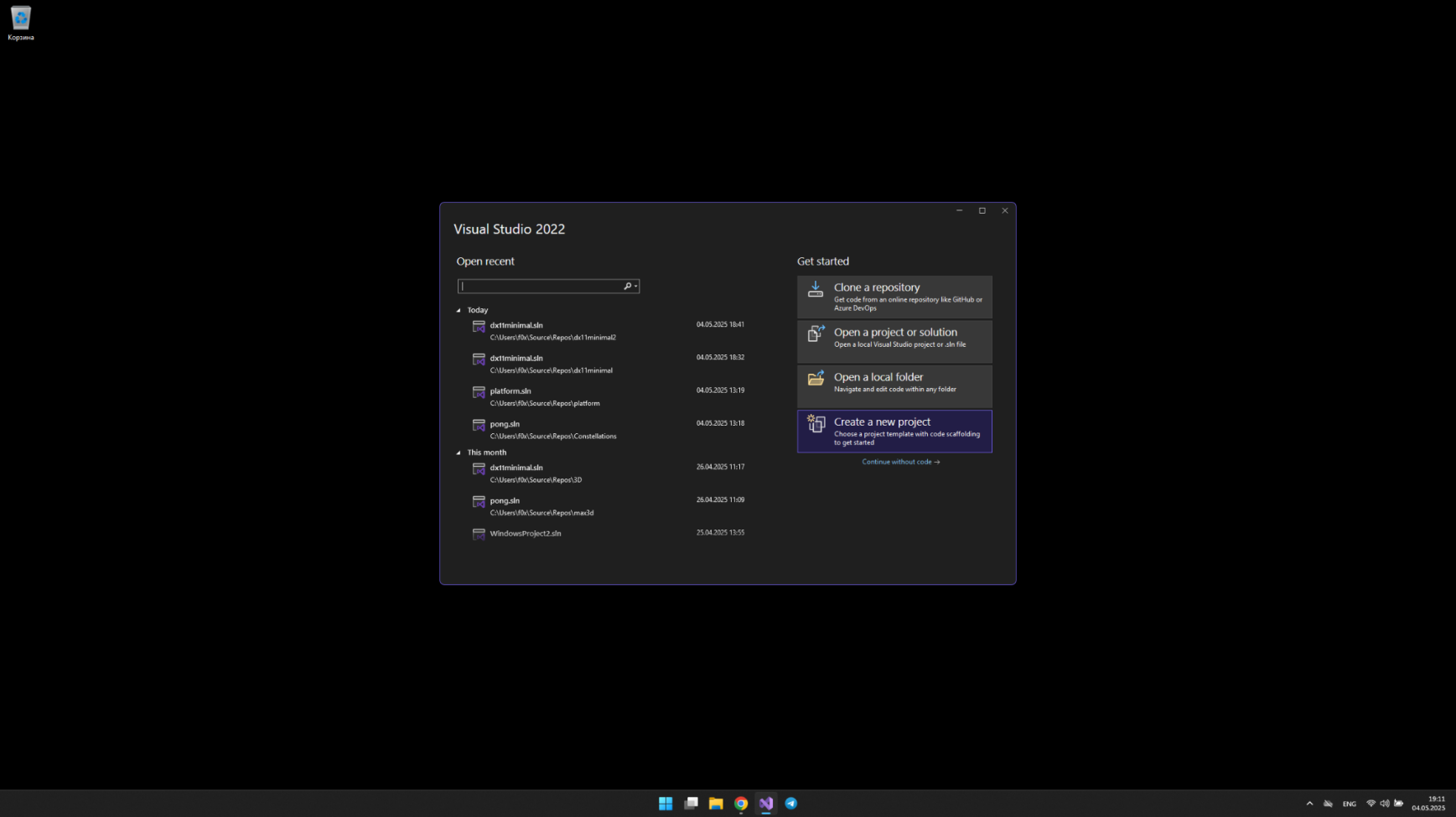
Решения всех задач можно посмотреть по **[ссылке](https://github.com/buffyemm/lesson)**. Однако, не рекомендую вам сразу заглядывать в код - ваше обучение состоит именно в *самостоятельном* решении задач. Используйте репозиторий с решениями только в случае, если долгое время не можете понять, как заставить вашу программу работать.

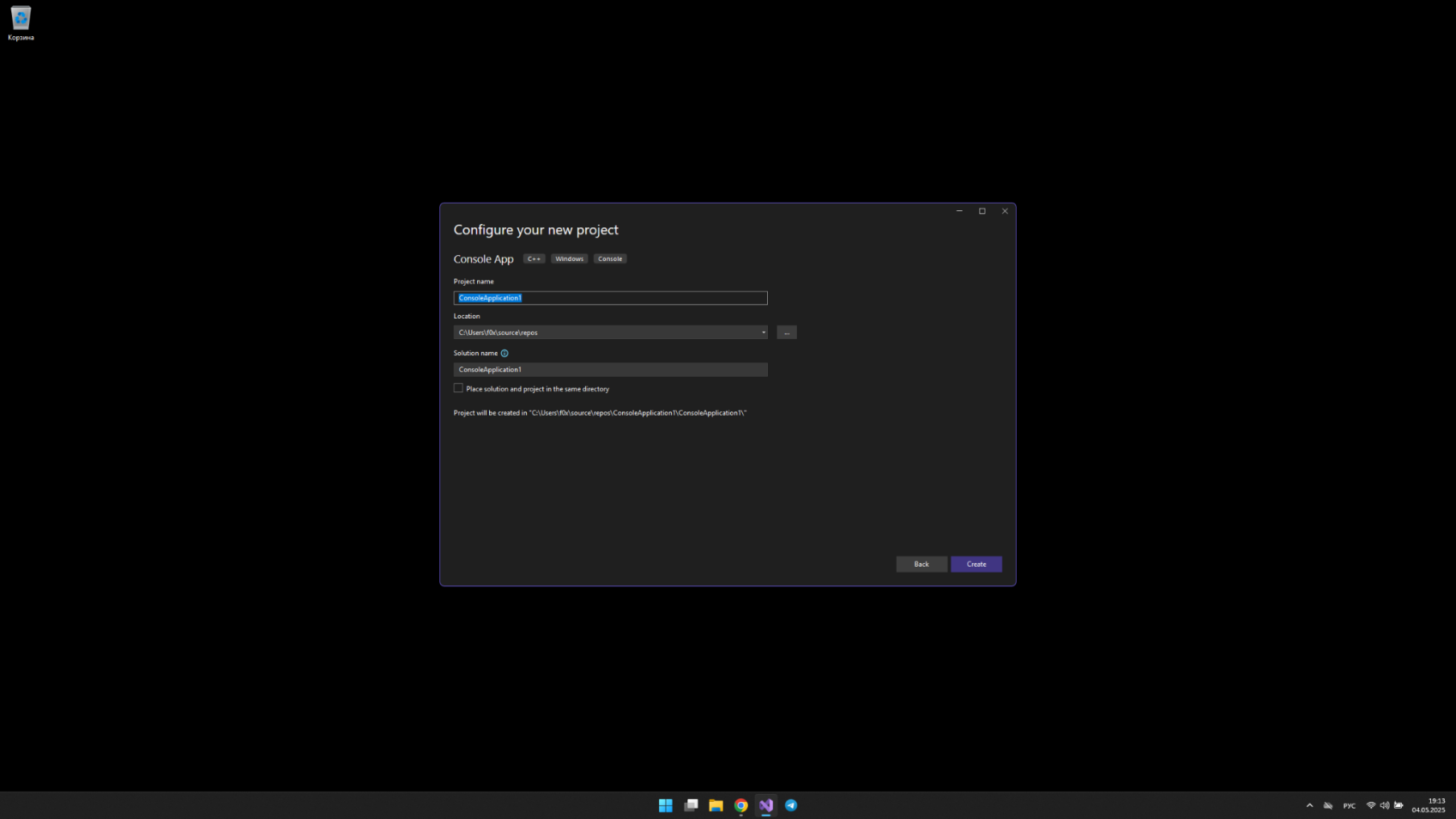
Для первых задач практикума объяснение будет максимально подробным. Далее, будет расти сложность задач, и, одновременно, уменьшаться количество сопроводительного текста. Сделано это с целью привить вам навык самостоятельного поиска и анализа информации. В студии, программист работает по техническому заданию. Умение читать это техзадание, а также оценивать его точность и полноту - очень важный навык.

# Базовые сведения

Любой компьютер исполняет программы только на машинном языке, который не очень удобен для человека. С++ - компилируемый язык. Это значит, вашу С++ программу для выполнения необходимо перевести на язык машины. Занимается этим специальная программа (компилятор) который входит в состав Visual Studio.

Visual Studio на старте предложит вам создать шаблонное приложение. Для блоков №1 и №2 нам понадобится шаблон Console App, для последующих - Windows Desktop Application.





В случае выбора консольного приложения будет сгенерирован проект со следующим текстом:

// ConsoleApplication1.cpp : This file contains the 'main' function. Program execution begins and ends there.

//

#include <iostream>

int main()

{

std::cout << "Hello World!\n";

}

// Run program: Ctrl + F5 or Debug > Start Without Debugging menu

// Debug program: F5 or Debug > Start Debugging menu

// Tips for Getting Started:

// 1. Use the Solution Explorer window to add/manage files

// 2. Use the Team Explorer window to connect to source control

// 3. Use the Output window to see build output and other messages

// 4. Use the Error List window to view errors

// 5. Go to Project > Add New Item to create new code files, or Project > Add Existing Item to add existing code files to the project

// 6. In the future, to open this project again, go to File > Open > Project and select the .sln file

Проанализируем его структуру:

1. Строки, начинающиеся с **//** - это комментарии. Они не являются частью программы и нужны, чтобы пояснить людям, которые будут работать с вашим кодом, те или иные заложенные в него концепции. Также комментарии могут использоваться как памятки - например комментарий начинающийся с TODO: используется, чтобы не забыть, что нужно добавить в код какую-либо функциональность
2. Директива **#include** включает в текст какой-либо другой файл. Это, как-бы, гиперссылка для компилятора, по которой он перейдет, обработает расположенный там текст программы, а затем вернется на строку, следующую за #include. В данном случае мы включаем в программу файл **iostream**. Это уже написанный, стандартный код, который позволит нам принимать пользовательский ввод (считывать числа, строки введенные пользователем) и осуществлять вывод (печатать что-либо в окне).
3. **int main()**. Любая программа должна начать выполняться с какого-то места. В С++, в силу структуры языка, точку входа располагают не в начале текста программы. Из-за этого, потребовалось некое соглашение, откуда же начинать выполнение программы. В консольном приложении это место - **функция** main.

Main, как и любая функция, имеет определенный синтаксис.

int main ()

тип\_возвращаемого\_значения имя\_функции (аргументы\_функции)

Функция может вернуть значение (с помощью команды return) - таким образом вызывающая программа может узнать результат работы нашей программы. Также она может принять параметры, заданные при запуске.

Таким образом int main() - это частный случай более общего синтаксиса функций. О них мы поговорим позднее.

1. **Фигурные скобки {}.** Они определяют тело функции - т.е. область текста, составляющего эту функцию. В нашем конкретном случае - весь текст программы содержится внутри этих скобок. В более общем случае - фигурные скобки определяют область видимости, и используются в языке как маркеры указания тела какого-либо блока.
2. **std::cout << "Hello World!\n"** - вывод информации в консоль. Std:: указывает компилятору использовать пространство имен std. Пространство имен (namespace) - специальная область, в которой хранятся функции и переменные. Они помещены туда для того, чтобы исключить конфликт имен. То есть, дать вам возможность задавать любые имена переменных, не боясь дублирования названий с переменными в других местах программы.   
   Строка в кавычках - это то, что выведется на экран. Символ “\n” - перевод вывода на новую строку.

# Блок №1

## Практикум №1. Первая игра

### 1.1 Простые расчеты

Перед тем, как приступить к первой игре, давайте немного посчитаем. Концепция математических расчетов реализована в С++ следующим образом:

1. вы объявляете переменные
2. инициализируете их - т.е. кладете в них конкретные значения
3. производите математические действия.

Для объявления переменных используется следующий синтаксис:

тип\_переменной имя\_переменной;

Примеры:

int i;

float alpha1;

std::string Text;

Типов данных существует довольно много, о них поговорим позже. В рамках наших задач, нам пока будет достаточно типа **int** - он позволяет хранить в переменных целые числа

Для инициализации переменной мы сначала пишем ее имя (без типа!) затем знак “=”, после чего - значение.

Пример:

i =10;

В С++ можно совмещать объявление переменной и ее инициализацию.

Пример:

int i=10;

Инициализирующее значение не обязано быть простым числом - туда можно записать выражение, в том числе и с участием других переменных.

Пример:

int a = 2;

Int b = 5;

int i = 10 + 2;

i = a + b;

i = b + 2;

**Задача**: рассчитать сумму, разность, произведение, частное, остаток от деления двух переменных.

### 1.2 Секюрити в клубе

**Задача:** принять ввод - возраст пользователя. Если возраст меньше 18 лет - вывести сообщение “entrance denied”, если больше либо равно - “you're welcome”.

**Декомпозиция задачи:**

1. объявить переменную
2. принять в нее пользовательский ввод
3. обработать условие и вывести результат

С первым пунктом вы уже знакомы. Пользовательский ввод делается так:

std::cin >> имя\_переменной

После выполнения этой команды число попадет в указанную вами переменную.

Условия задаются так:

if (условие) {

действие\_при\_выполнении\_условия

} else {

действие\_при\_**не**выполнении\_условия

}

Пример:

If (x==3) {

a=5;

} else {

a=10;

}

Удвоенное равно (==) это **проверка** на равенство. Одинарное равно (=) это **присваивание**. Не путайте эти две конструкции.

Также, вы можете использовать условия для проверки, какая из двух переменных больше или меньше другой, с помощью соответствующих знаков (> и <).

### 1.3 Угадай число

**Задача:** Сгенерировать случайное число *в заданном диапазоне*, принять пользовательский ввод. Сравнить, и если числа совпадают - вывести сообщение “you win”, иначе - “you lose”.

Программа должна выполниться десять раз. Новое случайное число должно генерироваться только в случае, если пользователь угадал предыдущее.

Декомпозиция здесь напрямую следует из задачи и не должна вызвать у вас затруднений.

Для решения вам потребуется понимание случайных чисел и циклов.

**Случайные числа**

В контексте программирования мы говорим о **псевдослучайных** последовательностях - истинно случайные компьютеру взять особо не откуда. То есть, речь идет об алгоритме, результат работы которого воспринимается человеком как случайная последовательность.

В С++ для получения таких чисел используется функция **rand()**

Обратите внимание: эта функция при каждом вызове вернет *новое* случайное число.

В ряде задач требуется получение псевдослучайной последовательности, **одинаковой** при каждом вызове. Это реализуется с помощью команды **srand**(\_аргумент\_), где аргумент - это, условно, индекс смещения в последовательности. Так что пара команд

srand(\_аргумент\_);

int a = rand();

всегда присвоит переменной “а” одно и то же значение.

**Циклы**

Когда мы хотим выполнить некую последовательность команд несколько раз - мы используем циклы. Синтаксис:

for (<объявление счетчика - тип, имя>; <условие выполнения цикла>; <шаг цикла>)

{

<тело цикла>

}

Пример:

for (int i=0;i<10;i++)

{

}

Тело этого цикла выполнится десять раз, и на каждой итерации переменная i будет содержать новое значение. В данном случае - каждый раз к ней будет прибавляться единица.

### 1.4 Камень, ножницы, бумага

**Задача:** модифицировать программу “угадай число” таким образом, чтобы получилась игра “камень, ножницы, бумага”. Программа должна принимать на вход строку текста, и распознавать введенный вариант ответа. Используйте английский язык для текста. Также необходимо провести **валидацию** пользовательского ввода - если введенное слово не соответствует ни одному из вариантов, попросить пользователя повторить ввод.

Для работы со строками вам понадобится новый тип данных - std::string. Строки этого типа имеют множество возможностей, в частности, их можно сравнивать также, как числа.

Синтаксис такой же как у обычных переменных. Текстовое значение пишется в кавычках:

std::string a;

a = "text";

std::string b = “text2”;

В процессе разработки вам потребуется несколько раз использовать одни и те же текстовые значения - для вывода текста, валидации введенного текста, определения победы и поражения. Один из вариантов избежать этого - использовать **массивы** строк.

В этом случае, задав строки один раз, вы сможете обращаться к ним по индексам массива.

Пример:

std::string t[3] = { “text0”, “text1”, “text2”};

If (input == t[0]) {

userWeapon = 0;

}

### 1.5 Магазин

**Задача:** написать симулятор похода в магазин.

Есть список товаров имеющихся в магазине, с указанием цен и количества единиц. Известно, сколько у пользователя наличных денег.

Программа должна принимать список товаров, которые хочет купить пользователь, с указанием количества единиц каждого товара.

Требуется произвести следующие расчеты:

1. имеются ли в наличии товары которые хочет купить пользователь
2. хватит ли денег для приобретения товаров из списка
3. сколько денег останется после совершения покупок

Дополнительно, программа должна дать ответы на вопросы:

1. каким должен быть список покупок, чтобы потратить максимальное количество денег
2. каким должен быть список покупок, чтобы купить максимальное число товаров

Декомпозицию задачи проведите самостоятельно.

## Практикум №2. Текстовая адвенчура

### 2.1 Перемещение между локациями

**Задача:** написать программу, задающую несколько локаций, и обеспечить перемещение игрока по ним. Локация содержит название и текстовое описание. Игрок перемещается по ним вводя имя локации. Обеспечить валидацию пользовательского ввода.

Поскольку локация - комплексная сущность и ее нельзя представить одной переменной, рекомендуем использовать **структуры**.

Синтаксис:

struct имя\_структуры {

поля\_структуры

}

Пример:

struct box {

int x;

Int y;

int width;

int height;

}

Фактически, после объявления структуры мы получим новый тип данных. И сможем объявлять переменные этого типа:

box prettyBox;

Доступ к полям структуры делается через точку (.).

Пример:

int x = prettyBox.x;

Поскольку у нас несколько локаций, но их внутреннее устройство однотипно, мы можем объявить **массив структур**. Делается это, например, так:

box boxArray[5];

где box - тип, boxArray - имя, 5 - количество элементов в массиве.

Доступ:

int x = boxArray[3].x;

Смысл применения массивов - в возможности **итерироваться** по ним. В частности, мы можем написать цикл, в котором переберем все локации - например, если нам надо найти некий предмет, местонахождение которого нам неизвестно.

Местонахождение игрока логично хранить как переменную, содержащую номер локации. Описание локации, выводимое игроку, будет основано на этой переменной. Перемещение игрока осуществляется через изменение этой переменной.

### 2.2 Двери

**Задача:** переписать логику перемещения по локациям таким образом, чтобы из одной локации можно было попасть только в ограниченное количество других - например, из ванной можно переместиться в зал только через коридор. Игрок перемещается по локациям вводя их имя - из доступных по логике вариантов. Обеспечить валидацию пользовательского ввода.

Один из вариантов реализации - поместить переменные, отвечающие за двери, в струкутру локации. Каждая дверь будет содержать **номер** локации, в которую эта дверь ведет.

Поскольку дверей в локации может быть произвольное количество - удобно будет использовать конструкцию std::vector. Преимущество вектора над массивом в том, что вектор может хранить произвольное число переменных, причем добавлять и удалять их можно динамически.

Для использования вектора подключите соответствующий заголовочный файл:

#include <vector>

Саму переменную объявляем как обычно, но дополнительно указываем тип данных которые вектор будет хранить:

std::vector<int> door;

Положить данные в вектор можно командой push\_back(\_значение\_). Пример:

door.push\_back(6);

### 2.3 Предметы

**Задача:** обеспечить работу с предметами: создать полный список предметов в игре, расположить их в локациях, реализовать обработку команд:

List - показать все предметы, находящиеся в локации

Trunk - показать все предметы в “рюкзаке” игрока

Pick (название предмета) - поднять предмет и положить его в рюкзак игрока

Drop (название предмета) - сбросить предмет из рюкзака в локацию

Use (название предмета 1, название предмета 2) - использовать предмет (например - ключ на двери)

Из дополнительных возможностей стандартной библиотеки вам потребуется метод erase - он будет нужен для удаления предметов. Самостоятельно изучите его синтаксис.

### 2.4 Non-player characters (NPC)

**Задача:** добавить игроку характеристику количества жизненной энергии. Добавить NPC и прописать им логику нападения на игрока. Придумать правила боя и реализовать их в программе.