Рязанский станкостроительный колледж РГРТУ

МДК.01.01 Разработка программных модулей

Тема 2. Структурное программирование

Рязань 2021

Оглавление

Технология структурного программирования

Технологией программирования называют совокупность методов и средств, используемых в процессе разработки программного обеспечения.

Исторически в развитии программирования можно выделить несколько технологий программирования принципиально отличающихся методологий.

Изначально понятие технологии как таковой — это 60-е годы прошлого столетия — это период **"стихийного" программирования**. В этот период отсутствовало понятие структуры программы, типов данных и т.д. Вследствие этого код получался запутанным, противоречивым. Программирование тех лет считалось искусством. Для небольших программ это вполне годилось, но с увеличением размера программ это стало большим тормозом программирования.

Структурный подход к программированию представляет собой совокупность рекомендуемых технологических приемов, охватывающих выполнение всех этапов разработки программного обеспечения. В основе структурного подхода лежат три основных подхода:

- **декомпозиция** (разбиение на части) сложных систем с целью последующей реализации в виде отдельных небольших подпрограмм. С появлением других принципов декомпозиции (объектного, логического и т.д.) данный способ получил название процедурной декомпозиции.
- другим базовым принципом структурного программирования является использование при составлении программ только базовых алгоритмических структур (линейные, разветвляющиеся, циклические), запрет на использование оператора GOTO.
- повышения уровня типизации данных. Теория типов данных исходит из того, что каждое используемое в программе данное принадлежит одному и только одному типу данных. Тип данного определяет множество возможных значений данного и набор операций, допустимых над этим данным. Данное конкретного типа в ряде случаев может быть преобразовано в данное другого типа, но такое преобразование должно быть явно представлено в программе. Применение типизированного языка при написании программы позволяет еще при трансляции исходного текста выявить многие ошибки использования данных и этим повысить надежность программы

С применением структурного подхода появилась возможность читать и проверять программу как последовательный текст, что повысило производительность труда программистов при разработке и отладке программ. С целью повышения структурности программы были выдвинуты требования к большей независимости подпрограмм, подпрограммы должны связываться с вызывающими их программами только путем передачи им аргументов, использование в подпрограммах переменных, принадлежащих другим процедурам или главной программе, стало считаться нежелательным.

Поддержка принципов структурного программирования была заложена в основу так называемых процедурных языков программирования. Как правило, они включали основные "структурные" операторы передачи управления, поддерживали вложение подпрограмм, локализацию и ограничение области "видимости" данных. Среди наиболее известных языков этой группы стоит назвать Pascal, C.

Дальнейший рост сложности и размеров разрабатываемого программного обеспечения потребовал развития структурирования данных. Как следствие этого в языках появляется возможность определения пользовательских типов данных. Одновременно усилилось стремление разграничить доступ к глобальным данным

программы, чтобы уменьшить количество ошибок, возникающих при работе с глобальными данными. В результате появилась и стала развиваться технология модульного программирования.

Модульное программирование предполагает выделение групп подпрограмм, использующих одни и те же глобальные данные, в отдельно компилируемые модули (***библиотеки подпрограмм***). Связи между модулями при использовании данной технологии осуществляются через специальный интерфейс, в то время как доступ к реализации модуля (телам подпрограмм и некоторым "внутренним" переменным) запрещен. Эту технологию поддерживают современные версии языков Pascal и C (C++).

Инструментальные средства оформления и документирования алгоритмов программ

Для разработки алгоритма программы в виде блок-схемы могут использоваться различные инструментальные средства. В дисциплине «Основы алгоритмизации и программирования» подробно рассматривалась программа Maicrosoft Visio, поэтому подробно на инструментальных средствах останавливаться не будем. Также можно использовать и другие инструментальные средства для создания блок-схем алгоритмов программ.

Вспомним что из себя представляет алгоритм программы, оформленный в виде блок-схемы.

Блок-схема алгоритма представляет собой систему связанных геометрических фигур.

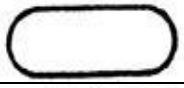
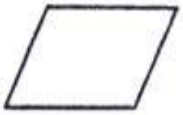
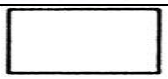
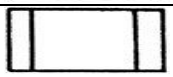
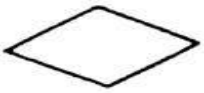
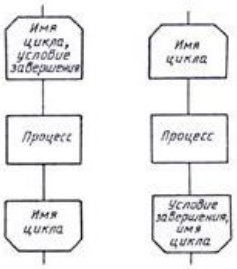
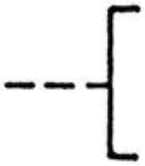
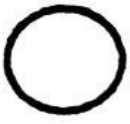
Каждая фигура обозначает один этап решения задачи и называется блоком. Порядок выполнения этапов указывается стрелками, соединяющими блоки. В схеме блоки стараются размещать сверху вниз, в порядке их выполнения. Для наглядности операции разного вида изображаются в схеме различными геометрическими фигурами.

Правила построения блок-схем (ГОСТ 19.701-90 (ИСО 5807-85)):

1. Символы в схеме должны быть расположены равномерно. Следует придерживаться разумной длины соединений и минимального числа длинных линий. Символы должны быть, по возможности, одного размера.
2. Минимальное количество текста, необходимого для понимания функции данного символа, следует помещать внутри данного символа. Текст для чтения должен записываться слева направо и сверху вниз независимо от направления потока. Если объем текста, помещаемого внутри символа, превышает его размеры, следует использовать символ комментария.
3. Блок-схема выстраивается в одном направлении либо сверху вниз, либо слева направо
4. Потоки данных или потоки управления в схемах показываются линиями. Направление потока слева направо и сверху вниз считается стандартным. В случаях, когда необходимо внести большую ясность в схему (например, при соединениях), на линиях, используются стрелки. Если поток имеет направление отличное от стандартного, стрелки должны указывать это направление.
5. Линии в схемах должны подходить к символу либо слева, либо сверху, а исходить либо справа, либо снизу. Линии должны быть направлены к центру символа.
6. При необходимости линии в схемах следует разрывать для избегания излишних пересечений или слишком длинных линий, а также, если схема состоит из нескольких страниц. Соединитель в начале разрыва называется внешним соединителем, а соединитель в конце разрыва — внутренним соединителем.

7. Все повороты соединительных линий выполняются под углом 90 градусов.

Таблица 1. Элементы графических фигур, используемых при построении блок – схемы.

№	Наименование этапа	Изображение	Примечание
1	Терминатор		Начало и конец алгоритма
2	Передача данных		Ввод или вывод информации
3	Процесс		Арифметический блок, определяющий действие, которое нужно выполнить
4	Преопределенный процесс		Подпрограмма или функция
5	Принятие решения		Логический блок, проверяющий истинность или ложность некого условия
6	Подготовка		Модификацию команды или группы команд с целью воздействия на некоторую последующую функцию (установка переключателя, модификация индексного регистра или инициализация программы)
7	Цикл		Символ, состоящий из двух частей, отображает начало и конец цикла. Обе части символа имеют один и тот же идентификатор. Условия для инициализации, приращения, завершения и т.д. помещаются внутри символа в начале или в конце в зависимости от расположения операции, проверяющей условие.
8	Комментарий		Символ используют для добавления описательных комментариев или пояснительных записей в целях объяснения или примечаний. Пунктирные линии в символе комментария связаны с соответствующим символом или могут обводить группу символов. Текст комментариев или примечаний должен быть помещен около ограничивающей фигуры.
9	Соединитель		Символ отображает выход в часть схемы и вход из другой части этой схемы и используется для обрыва линии и продолжения ее в другом месте. Соответствующие, символы-соединители должны содержать одно и то же уникальное обозначение.

Пример 1.

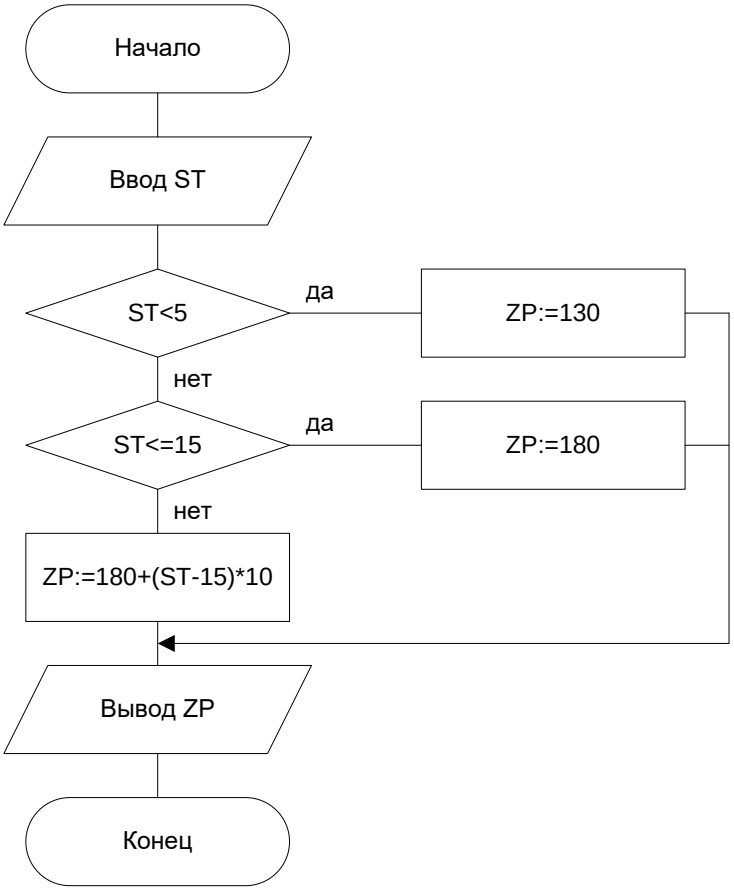
Составить алгоритм начисления зарплаты согласно следующему правилу: если стаж работы сотрудника менее 5 лет, то зарплата 130 руб., при стаже работы от 5 до 15 лет - 180 руб., при стаже свыше 15 лет зарплата повышается с каждым годом на 10 рублей.

Сформулируем задачу в математическом виде:

Вычислить

$$ZP = \begin{cases} 130, \text{если } ST < 5; \\ 180, \text{если } 5 \le ST \le 15 \\ 180 + (ST - 15)10, \text{если } 15 < ST \end{cases}$$

Вернемся к нашей задачи расчета заработной платы. Блок-схема описанного выше алгоритма будет иметь вид:



Концепции разработки программного модуля

Как правило программа имеет модульную структуру и состоит из множества модулей, которые могут разрабатывать команда проекта. Под модулями тут могут пониматься отдельные функции, классы, библиотеки функций или классов.

При разработке программного модуля, как правило, придерживаются следующего порядка:

- разработка или проверка спецификации модуля;
- выбор алгоритма и структуры данных;
- программирование (кодирование) модуля;

- шлифовка текста модуля;
- проверка модуля;
- компиляция модуля:
- тестирование и отладка модуля

Разработка или проверка спецификации модуля

На этом шаге происходит разработка или проверка спецификации модуля. Спецификация модуля содержит описание функции или функций, описание входных и выходных аргументов.

При разработке модулей, спецификацию модулей пишет разработчик.

При работе в команде, когда спецификация модуля уже разработана, разработчик изучает спецификацию модуля и должен убедиться, что она ему понятна и достаточна для разработки этого модуля.

Выбор алгоритма и структуры данных

На втором шаге разработки программного модуля необходимо выяснить, не известны ли уже какие-либо алгоритмы для решения поставленной и или близкой к ней задачи. И если найдется подходящий алгоритм, то целесообразно им воспользоваться. Выбор подходящих структур данных, которые будут использоваться при выполнении модулем своих функций, в значительной степени предопределяет логику и качественные показатели разрабатываемого модуля, поэтому его следует рассматривать как весьма ответственное решение.

Соответственно все описанное выше выполняет разработчик, либо при работе в команде архитектор приложения.

Программирование (кодирование) модуля

На третьем шаге осуществляется построение текста модуля на выбранном языке программирования. Обилие всевозможных деталей, которые должны быть учтены при реализации функций, указанных в спецификации модуля, легко могут привести к созданию весьма запутанного текста, содержащего массу ошибок и неточностей. Искать ошибки в таком модуле и вносить в него требуемые изменения может оказаться весьма трудоемкой задачей. Поэтому весьма важно для построения текста модуля пользоваться технологически обоснованными и практически проверенными стандартами оформления кода.

Следующий шаг разработки модуля связан с приведением текста модуля к завершенному виду в соответствии со спецификацией качества ПО. При программировании модуля разработчик основное внимание уделяет правильности реализации функций модуля, оставляя недоработанными комментарии и допуская некоторые нарушения требований к стилю программы. При шлифовке текста модуля он должен отредактировать имеющиеся в тексте комментарии и, возможно, включить в него дополнительные комментарии с целью обеспечить требуемые примитивы качества.

С этой же целью производится редактирование текста программы для выполнения стилистических требований, проводится рефакторинг кода. Цель таких действия сделать код программы по стилю таким, чтобы все модули имели один стиль написания, как будто их разрабатывал один человек, и чтобы код программы был понятен даже без комментариев.

Проверка модуля

Шаг проверки модуля представляет собой ручную проверку внутренней логики модуля до начала его отладки.

Можно использовать статическую проверку текста модуля, этот текст прочитывается с начала до конца с целью найти ошибки в модуле. Обычно для такой проверки привлекают, кроме разработчика модуля, еще одного или даже нескольких программистов. Рекомендуется ошибки, обнаруживаемые при такой проверке исправлять не сразу, а по завершению чтения текста модуля.

Можно использовать сквозное прослеживание, которое представляет собой один из видов динамического контроля модуля. В нем также участвуют несколько программистов, которые вручную прокручивают выполнение модуля (оператор за оператором в той последовательности, какая вытекает из логики работы модуля) на некотором наборе тестов.

Компиляция модуля

Сборка модуля или программы, включающая трансляцию всех модулей программы, написанных на одном или нескольких исходных языках программирования высокого уровня и/или языке ассемблера, в эквивалентные программные модули на низкоуровневом языке, близком машинному коду (байт код, объектный модуль и последующую сборку исполняемой машинной программы.

Если компилятор генерирует исполняемую машинную программу на машинном языке, то такая программа непосредственно исполняется физической программируемой машиной (например, компьютером).

В других случаях исполняемая машинная программа выполняется соответствующей виртуальной машиной. Входной информацией для компилятора (исходный код) является описание алгоритма или программы на предметно-ориентированном языке, а на выходе компилятора — эквивалентное описание алгоритма на машинно-ориентированном языке (объектный код, байт-код).

Тестирование и отладка модуля

Тестирование - это процесс выполнения модуля на некотором наборе данных, для которого заранее известен результат применения или известны правила поведения этих модулей. Указанный набор данных называется тестовым набором или просто тестом.

Отладка - это процесс, направленный на обнаружение и исправление ошибок в ПО.

Оформление документации на программные модули

Составление паспортов модулей (функций) программы

Для проектирования модуля исходной информацией (заданием на проектирование, постановкой задачи) является его паспорт.

Паспорт модуля может содержать:

1. Заголовок (Название модуля, параметры и возвращаемое значение).
2. Язык программирования, на котором модуль будет написан.
3. Описание входной информации модуля – описание переменных (имя, тип, описание) согласно указаниям, приведенным выше.
4. Описание выходной информации – описание переменных (имя, тип, описание) согласно указаниям, приведенным выше.
5. Модули, вызываемые из описываемого модуля. Указывается передаваемая информация из описываемого модуля в вызываемый и возвращаемая информация из вызываемого модуля в описываемый.
6. Основные подфункции модуля с указанием логики функционирования.
7. Сведения о литературных первоисточниках и изложение алгоритма решения.

Правильно написанный паспорт модуля даёт возможность разрабатывать и тестировать его автономно от остальных модулей.

Пример паспорта модуля (функций)

Разработчик студент группы П-31 Сидоров Сидор
Практическая работа №1, вариант №100

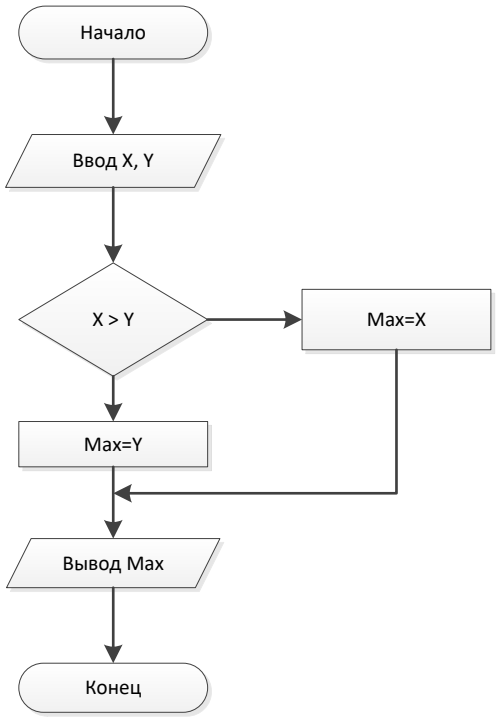
Спецификации модуля
Из двух чисел найти максимальное число
int MaxXY(int x, int y)
Параметры
int x - первое число
int y - второе число
Возвращаемое значение
int - максимальное число

Или можно так

Таблица спецификации модуля.

Имя	Реализуемая функция	Параметры
int MaxXY(int x, int y)	Из двух чисел найти максимальное число	*Входные данные:* x, y – исходные числа *Входные данные:* Результат функции максимальное число

Алгоритм функции int MaxXY(int x,int y)



Оценка сложности алгоритма

Понятие сложности алгоритма

Традиционно в программировании понятие сложности алгоритма связано с использованием ресурсов компьютера: сколько процессорного времени требует программа для своего выполнения, как много памяти машины при этом расходуется? Учет памяти обычно ведется по объему данных. Время рассчитывается в относительных единицах так, чтобы эта оценка, по возможности, была одинаковой для различных машин.

Постоянная задача для программистов – сделать программу, работающую хотя бы немного быстрее и попытаться заставить ее работать в условиях ограниченной памяти. Для ряда областей ресурсы настолько критичны, что может возникнуть проблема целесообразности всего проекта из-за неэффективной работы программы. К таким областям относятся системы реального времени (real-time systems).

Временная сложность

Временная сложность оценивается путём подсчёта числа элементарных операций, осуществляемых алгоритмом, где элементарная операция занимает для выполнения фиксированное время.

Линейная сложность

Говорят, что алгоритм работает за линейное время, или O(n), если его сложность равна O(n). Неформально, это означает, что для достаточно большого размера входных данных время работы увеличивается линейно от размера входа. Например, программа, суммирующая все элементы, требует время, пропорциональное длине списка. Это описание не вполне точно, поскольку время работы может существенно отличаться от точной пропорциональности, особенно для малых значений n.

Квадратичная сложность

Такую сложность $O(n^2)$ имеет, например, алгоритм сортировки вставками. В канонической реализации он представляет из себя два вложенных цикла: один, чтобы проходить по всему массиву, а второй, чтобы находить место очередному элементу в уже отсортированной части. Таким образом, количество операций будет зависеть от размера массива как n * n, т. е. n^2.

Кубическая сложность

Такую сложность $O(n^3)$ имеет, алгоритм содержащий три вложенных цикла. Таким образом, количество операций будет зависеть от размера массива как n * n * n, т. е. n^3.

Эффективность алгоритмов

Предположим, быстродействие компьютера и объем его памяти можно увеличивать до бесконечности. Была бы тогда необходимость в изучении алгоритмов? Да, но только для того, чтобы продемонстрировать, что метод развязку имеет конечное время работы и что он дает правильный ответ. Если бы компьютеры были неограниченно быстрыми, подошел бы произвольный корректный метод решения задачи. Конечно, тогда чаще всего избирался бы метод, который легче реализовать.

Сегодня очень мощные компьютеры, но их быстродействие не является бесконечно большой, как и память. Таким образом, при исчислении - это такой же ограниченный ресурс, как и объем требуемой памяти. Этими ресурсами следует пользоваться разумно, чем и способствует применение алгоритмов, которые эффективны в плане использования ресурсов времени и памяти.

Алгоритмы, разработанные для решения одной и той же задачи, часто могут очень сильно отличаться по эффективности. Эти различия могут быть намного больше заметными, чем те, которые вызваны применением различного аппаратного и программного обеспечения.

Пример 1.

Вычислить сумму элементов главной диагонали матрицы.

Код программы вариант №1

```
static void Main(string[] args)
{

    const int n = 10; //Размер массива
    int i, j,   //Счетчики
        sum;    //Сумма
    int[,] matr = new int[n, n]; //Массив

    // Инициализация случайными значениями в диапазоне -100-100
    Random rnd = new Random();
    for (i = 0; i < n; i++)
    for (j = 0; j < n; j++) matr[i, j] = rnd.Next(-100, 100);
    //Ищем сумму положительных чисел
    sum = 0;
    for (i = 0; i<n; i++)
        for (j = 0; j<n; j++)
            if (i == j) sum = sum +matr[i, j];
    Console.WriteLine("Сумма пложительных элементов= " + sum);
    Console.ReadKey();//Пауза
}
```

Определяем сложность работы алгоритма?

Код программы вариант №2

```
static void Main(string[] args)
{
    const int n = 10; //Размер массива
    int i, j,   //Счетчики
        sum;    //Сумма
    int[,] matr = new int[n, n]; //Массив

    // Инициализация случайными значениями в диапазоне -100-100
    Random rnd = new Random();
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++) matr[i, j] = rnd.Next(-100, 100);
    //Ищем сумму положительных чисел
    sum = 0;
    for (i = 0; i < n; i++)
        sum = sum + matr[i, i];
    Console.WriteLine("Сумма пложительных элементов= " + sum);
    Console.ReadKey();//Пауза
}
```

Определяем сложность работы алгоритма?

Пример 2.
Подсчитать сумму делителей элементов главной диагонали матрицы.

Код программы вариант №1

```
static void Main(string[] args)
{
    int n; //Размер массива
    int i, j, k;   //Счетчики
    int sum = 0,    //Сумма
        kol; // Количество

    Console.Write("Введите размер массива (100/1000/10000) - ");
    n = Convert.ToInt32(Console.ReadLine());
    int[,] matr = new int[n, n]; //Массив

    // Инициализация случайными значениями в диапазоне 0 - n
    Random rnd = new Random();
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++) matr[i, j] = rnd.Next(n)+1;

    //Ищем сумму делителей чисел
    for (i = 0; i < n; i++)
    {
        Console.WriteLine("Шаг - " + i);
        for (j = 0; j < n; j++)
        {
            kol = 0;
            for (k = 1; k <= matr[i, j]; k++)
                if (matr[i, j] % k == 0) kol++;
            if (i == j) sum = sum + kol;
        }
    }
    Console.WriteLine("Сумма делителей = " + sum);
    Console.ReadKey();//Пауза
}
```

Определяем сложность работы алгоритма?

Код программы вариант №2

```
//Ищем сумму делителей чисел
for (i = 0; i < n; i++)
{
    Console.WriteLine("Шаг - " + i);
    for (j = 0; j < n; j++)
    {
        if (i == j)
        {
            kol = 0;
            for (k = 1; k <= matr[i, j]; k++)
                if (matr[i, j] % k == 0) kol++;
            sum = sum + kol;
        }
    }
}
```

Определяем сложность работы алгоритма?

Задание

1. Проведите испытания программ из примера 2. Оцените время их работы. Для проведения испытаний в папке с лекциями приложено 2 файла: «***Лекция ОАиП №9. Пример2. Вариант1»*** и «***Лекция ОАиП №9. Пример2. Вариант1***».
2. Изучите пример 2 и найдите еще варианты для оптимизации программы.

Стандарты оформления кода

Основные положения

Стандарт является соглашением по оформлению и написанию кода на языке C#. В документе приведены основные правила оформления кода и приемы, используемые при написании программ.

Стандарт является обязательным для любых разработок программных систем, а также разработки учебных программ в рамках учебного процесса РССК РГРТУ.

Далее мы рассмотрим основные положения по оформлению кода на языке C#, более подробно смотрите файл «***Стандарт оформления кода C#***».

Стили наименования

PascalCase – первая буква каждого слова в имени идентификатора начинается с верхнего регистра.

Пример: TheCategory.

CamelCase – первая буква первого слова в идентификаторе в нижнем регистре, все первые буквы последующих слов – в верхнем.

Пример: theCategory.

Общие правила именования идентификаторов

Для названий любых элементов программного кода (модулей, классов и их методов, функций, переменных, констант, последовательностей, представлений), следует использовать содержательные имена, позволяющие сделать вывод о назначении данных элементов. При именовании идентификаторов не используются аббревиатуры или сокращения, если только они не являются общепринятыми.

Пример: GetWindow(), а не GetWin();

Если имя идентификатора включает в себя сокращение – сокращение пишется в PascalCase. Исключение - когда имя идентификатора должно быть указано в CamelCase и сокращение стоит в начале имени идентификатора. В этом случае сокращение пишется в нижнем регистре.

Пример:
SqlAccount для PascalCase, sqlAccount для CamelCase.

Использование верхнего и нижнего регистра в именах

Запрещается создавать два различных имени, функции или свойства с одинаковыми именами, отличающиеся только регистром. Запрещается создавать функции

с именами параметров, отличающимися только регистром. Ниже приведены примеры НЕправильных названий.

Пример:
KeywordManager и Keywordmanager;
findByID(int id) и FindByID(int id);

Правила именования переменных

Для именования переменных используется стиль CamelCase;

Пример: int userId = 20;

Правила именования констант

Для именования констант используется стиль PascalCase.

Правила именования функций

Для именования функций используется стиль PascalCase;

Имена функций должны давать четкое представление о том, какое действие эта функция выполняет. Имя функции начинается с глагола, указывающего на то, какое действие она выполняет.

Пример: public void CreateItem();

Правила именования параметров функций

Для именования параметров используется стиль CamelCase;

Имена параметров должны давать четкое представление о том для чего используется параметр, и какое значение следует передать при вызове функции.

Пример:
public void EncodeString(string sourceString, ref string encodedString),
а не public void EncodeString(string string1, ref string string2).

Форматирование кода программы

Используются стандартные настройки форматирования Visual Studio.

Для улучшения наглядности структуры программного кода следует использовать пробелы и отступы:

- Пробелы – для лучшего обозначения отдельных элементов кода в строке.
- Горизонтальные отступы – для лучшего обозначения вложенности функциональных блоков многострочного кода. При формировании горизонтальных отступов рекомендуется использовать tab - 4 пробела.
- Вертикальные отступы – для отделения логических блоков программного кода между собой.

Фигурные скобки размещаются всегда на отдельной строке.

В условии if-else всегда используются фигурные скобки.

Использование строк длиннее 100 символов не желательно. При необходимости инструкция переносится на другую строку. При переносе части кода на другую строку вторая и последующая строки сдвигаются вправо на один символ табуляции.

Каждая переменная объявляется на отдельной строке.

Комментирование кода программы

Все комментарии должны быть на русском языке.

В комментариях не допускается использование слов, не относящихся к техническому или деловому лексикону.

В комментариях не допускается наличие информации, не несущей смысловую нагрузку к пояснению сути алгоритма программного кода. Например, недопустимы комментарии вида: «это гениальный код», «таблица1», «!№;%:?*», «Это комментарий», «Эту фигню все равно никто не читает» и подобные им.

Если для языка программирования применяется система контроля версий, то комментарии вида: дата создания; дата модификации; автор создания; автор модификации - допускается вести в ней.

Комментарии к программному модулю, представленному в виде отдельного файла необходимо указывать всегда, при этом обязательны следующие комментарии:

- ***Описание назначения.***
- ***Дата создания.***
- ***Автор создания.***
- Дата модификации (при необходимости).
- Автор модификации (при необходимости).
- Цель модификации (при необходимости).

Комментарии к переменным, заголовкам свойств, полей, интерфейсов и прочего необходимо указывать всегда, при этом обязательны следующие комментарии:

- ***Описание назначения.***

Комментарии к заголовкам функций необходимо указывать всегда, при этом обязательны следующие комментарии:

- ***Описание назначения.***
- ***Описание входных и выходных параметров (при их наличии).***

Для функций, выполняющих сложные алгоритмы, не очевидные для восприятия, необходимо указывать подробные комментарии не только к заголовку функции, но и самому алгоритму с пояснением каждого шага выполнения алгоритма.

Для удобства работы с большой глубиной вложенности программных структур рекомендуется добавлять следующие комментарии:

- Метка окончания каждого ветвления алгоритма с указанием типа ветвления и краткого текста с условием.
- Метка окончания каждого цикла с указанием типа цикла и названия переменной-счетчика.
- Метка окончания модуля, класса, метода класса, события класса, функции, процедуры, триггера, пакета, с указанием названия закончившегося элемента программного кода, если указание названия не предусмотрено синтаксисом используемого языка.

Комментарии к тексту программного кода должны быть лаконичными и пояснять суть и назначение программного кода, а не его техническую составляющую. Например, вместо комментария «увеличиваем счетчик на 1» нужно писать «переходим к обработке следующего факультета». Текст комментариев не должен иметь сокращений, понимание которых может быть неоднозначным или затруднительным.

Пример программного кода с соблюдением правил оформления кода (язык С++):

```
// 14.01.2020  Сидоров С.С. ИСП-21
// Практическая работа №12 Сортировка
// Сортировка строк двумерного массива методом пузырька
#include <iostream>
#include <iomanip>
#include <stdlib.h>
#include <time.h>
using namespace std; //Установка стандартного пространства имен

//печать массива
void printMatrix
(int[][10], // Сортируемый массив
    const int // Размерность массива
);
int main()
{
    const int Size = 10; // Размерность массива
    int matrix[Size][Size]; // Определяем массив
    int temp; // Временная переменная
    srand(time(NULL)); // Определяем исходную точку для генератора случайных чисел

    // Заполняем массив случайным образом
    for (int i = 0; i < Size; i++)
        for (int j = 0; j < Size; j++)
            matrix[i][j] = 1 + rand() % 100;

    // Сортируем пузырьком
    for (int n = 1; n < Size * Size; n++)
    {
        for (int i = 0; i < Size; i++)
        {
            for (int j = 0; j < Size - 1; j++)
            {
                // Если нужно скорректировать порядок сортировки
                if (matrix[i][j + 1] < matrix[i][j])
                {
                    // Делаем обмен элементов массива местами
                    temp = matrix[i][j + 1];
                    matrix[i][j + 1] = matrix[i][j];
                    matrix[i][j] = temp;
                }// for (int j = 0; j < Size - 1; j++)
            }// for (int i = 0; i < Size; i++)
        }// for (int n = 1; n < Size * Size; n++)

        //Выводим массив на печать
        printMatrix(matrix, Size);

        return 0;
    }
}// end of main
```

```
// Печать массива
// Входные знвчения:
// varMX[][10] - сортируемый массив
// Size - размер массива
void printMatrix(int varMX[][10], const int size)
{
    for (int i = 0; i < size; i++)
    {
        cout << endl;
        for (int j = 0; j < size; j++) // Печать j-массива на одной строке
            cout << setw(4) << varMX[i][j];
        cout << endl; // Каждый j-массив на отдельной строке
    }
}// end of printMatrix
```

Разработка библиотек программных модулей

Термины:

Библиотека – это отдельный программный компонент, который обычно содержит несколько модулей (функции) связанных общим назначением. В C# библиотека представляет собой класс или классы в который упакованы модули (функции). Каждая библиотека создается в своем пространстве имен ***namespace***, которое в основной программе указывается через директиву ***using***.

Проект - это отдельная программная единица, которая содержит множество элементов для реализации программного продукта определенного направления.

Решение – агрегирующая единица, содержащая несколько проектов, связанных между собой. Окно обозреватель решений показывает все проекты, собранные в данном решении.

Алгоритм создания и использования библиотеки модулей в одном решении с прикладной программой.

1. Создайте проект прикладной программы:
   - В меню ***Файл*** последовательно выберите ***Создать***, а затем ***Проект***
   - Если нужно установите фильтр ***C#, Windows, Рабочий стол***
   - Из списка шаблонов выберите ***Приложение WPF (.NET Framework)***
   - Укажите название проекта и место его размещения
2. Разработайте интерфейс пользователя или сделайте это позже.
3. Создайте проект библиотеки модулей:
   - В меню ***Файл*** последовательно выберите ***Добавить***, а затем ***Создать проект***
   - Если нужно установите фильтр ***C#, Windows, Библиотека***
   - Из списка шаблонов выберите ***Библиотека классов(.Net Framework)***
   - Укажите название проекта и место его размещения
4. Создайте библиотеку модулей:
   - Измените на более подходящее имя пространства имен ***namespace ClassLibrary1***
   - Измените на более подходящее имя класса ***Class1***
   - Напишите необходимые модули (функции). Не забудьте установить модификаторы функций ***public*** - для доступа к функции вне класса и ***static*** – для использования функции без создания объекта класса

- Выполните сборку (компиляцию) библиотеки ***Сборка – Построить ClassLibrary1*** или ***Собрать решение.*** Обратите внимание, что библиотеку запустить на выполнения нельзя и при компиляции в папке \bin\debug создается файл библиотеки *.dll.

```
namespace ClassLibrary1
{
    Ссылок: 0
    public class Class1
    {
        Ссылок: 0
        public static int Summ(int x, int y)
        {
            return x + y;
        }
    }
}
```

Рис. Пример библиотеки

5. Настройте проект прикладной программы для использования библиотеки.
   - В ***обозревателе решений*** найдите проект прикладной программы выберите ***Ссылки (References)***, затем выберите ***Добавить ссылку...*** из контекстного меню.
   - В диалоговом окне "***Менеджер ссылок***" разверните ***Проекты*** и выберите ссылку на библиотеку с вашим названием ***ClassLibrary1***
   - В программном коде формы прикладной программы через директиву ***Using*** добавьте ссылку на пространство имен в библиотеке для использования коротких имен модулей. Например, using ***ClassLibrary1***.
6. Используйте функции библиотеки в прикладной программе

```
int x, y, z;
x = Convert.ToInt32(textBox1.Text);
y = Convert.ToInt32(textBox1.Text);
z = Class1.summ(x,y);
textBox3.Text = z.ToString();
```

Рис. Пример использования функции библиотеки

Алгоритм использования библиотеки модулей.

В данном случае предполагается что библиотека уже создана ранее в другом проекте, и мы имеем проверенный и откомпилированный файл библиотеке *.dll.

1. Создайте проект прикладной программы:
   - В меню ***Файл*** последовательно выберите ***Создать***, а затем ***Проект***
   - Если нужно установите фильтр ***C#, Windows, Рабочий стол***
   - Из списка шаблонов выберите ***Приложение WPF (.NET Framework)***
   - Укажите название проекта и место его размещения
2. Разработайте интерфейс пользователя или сделайте это позже.
3. Настройте проект прикладной программы для использования библиотеки.

- Скопируйте файл библиотеки ***.dll*** в папку с проектом прикладной программы или запомните путь к файлу ***.dll*** в проекте библиотеке
- В ***обозревателе решений*** найдите проект прикладной программы выберите ***Ссылки (References)***, затем выберите ***Добавить ссылку...*** из контекстного меню.
- В диалоговом окне "***Менеджер ссылок***" разверните ***Обзор*** и через кнопку ***Обзор..*** выберите файл библиотеки *.dll.
- В программном коде формы прикладной программы через директиву ***Using*** добавьте ссылку на пространство имен в библиотеке для использования коротких имен модулей. Например, using ***ClassLibrary1***.

4. Используйте функции библиотеки в прикладной программе

```
int x, y, z;
x = Convert.ToInt32(textBox1.Text);
y = Convert.ToInt32(textBox1.Text);
z = Class1.summ(x,y);
textBox3.Text = z.ToString();
```

Рис. Пример использования функции библиотеки

Пример части библиотеки модулей для выполнения действий с одномерным массивом (*универсальную библиотеку разработаете самостоятельно*).

```
namespace BiblMas
{
    ссылка: 1
    public class Massiv
    {
        //Заполнение массива случайными значениями
        //Col, Row - кол-во столбцов и строк
        //Rand- диапазон случ.значения 0-Rand
        //Выходные значения mas - Заполненный массив
        ссылка: 1
        public static void InitMas(out int [] mas, int column, int randMax)
        {
            Random Rnd; Rnd = new Random();
            mas = new Int32[column];
            for (int i = 0; i < column; i++)//заполнить массив случ.значениями
                mas[i] = Rnd.Next(randMax);
        }
        //Расчет суммы в массиве
        //Входные значения mas - массив
        //Выходные значения результат функции - сумма элементов массива
        Ссылок: 0
        public static int SumMas(int[] mas)
        {
            int sum = 0;//сумма
            for (int i = 0; i < mas.Length; i++)//заполнить массив случ.значениями
                sum = sum + Convert.ToInt32(mas[i]);
            return sum;
        }
    }
```

Пример использования модулей функций в основной программе

```
int[] mas;//Описываем массив как глобальный элемент

// Заполнить
Ссылок: 2
private void Заполнить_Click(object sender, RoutedEventArgs e)
{
    //Получаем диапазон случайных чисел
    int randMax = Convert.ToInt32(diapazon.Text);
    //Получаем количество ячеек в мвссиве
    int count = Convert.ToInt32(columnCount.Text);
    //Заполняем таблицу случайными значениями
    Massiv.InitMas(out mas,count,randMax);
    //Выводим массив на форму
    dataGrid.ItemsSource = VisualArray.ToDataTable(mas).DefaultView;
}
```

```
//Расчитать
Ссылок: 2
private void Рассчитать_Click(object sender, RoutedEventArgs e)
{
    //Суммируем ячейки таблицы
    int sum = Massiv.SumMas(mas); ;
    rez.Text = Convert.ToString(sum);//Ответ выводим на форму в поле результат
}
```

Замечание: Для вывода массива на форму в элемент DataGrid с помощью конструкции `dataGrid.ItemsSource = VisualArray.ToDataTable(mas).DefaultView;`. подключите приложенный класс VisualArray.

Добавить класс нужно командой ***Проект – Добавить класс*** и в разделе ***Visual C#*** выбрать элемент ***Класс***. Введите код класса, который можно взять из прилагаемого к лекции файла.

Пример сохранения одномерной таблицы в файл
(*универсальный модуль разработаете самостоятельно*)

```
//Создаем и настраиваем элемент SaveFileDialog
SaveFileDialog save = new SaveFileDialog();
save.DefaultExt = ".txt";
save.Filter = "Все файлы (*.*) | *.* |Текстовые файлы | *.txt";
save.FilterIndex = 2;
save.Title = "Сохранение таблицы";

//Открывем дмалоговое окно и при успехе работаем с файлом
if (save.ShowDialog() == true)
{
    //Создаем поток для работы с файлом и указываем ему имя файла
    StreamWriter file = new StreamWriter(save.FileName);

    //Записываем размер массива в файл
    file.WriteLine(mas.Length);
    //Записываем элементы массива в файл
    for (int i = 0; i < mas.Length; i++)
    {
        file.WriteLine(mas[i]);
    }
    file.Close();
}
```

Замечание: При использовании различных компонентов в создаваемой библиотеке модулей, например, для ***SaveFileDialog и StreamWriter*** нужно добавлять ссылку на системную библиотеку, где этот компонент прописан.

Пример чтения одномерной таблицы из файла
(*универсальный модуль разработаете самостоятельно*)

```
//Создаем и настраиваем элемент OpenFileDialog
OpenFileDialog open = new OpenFileDialog();
open.DefaultExt = ".txt";
open.Filter = "Все файлы (*.*) | *.* |Текстовые файлы | *.txt";
open.FilterIndex = 2;
open.Title = "Открытие таблицы";

//Откравем дмалоговое окно и при успехе работаем с файлом
if (open.ShowDialog() == true)
{
    //Создаем поток для работы с файлом и указываем ему имя файла
    StreamReader file = new StreamReader(open.FileName);

    //Читаем размер массива
    int len = Convert.ToInt32(file.ReadLine());
    //Создаем массив
    mas = new Int32[len];
    //Считываем массив из файла
    for (int i = 0; i < mas.Length; i++)
    {
        mas[i] = Convert.ToInt32(file.ReadLine());
    }
    file.Close();
```

Системы контроля версий, принципы организации работы

Основные понятия

Система контроля версий представляет собой программное обеспечение, которое позволяет отслеживать изменения в документах, при необходимости производить их откат, определять, кто и когда внес исправления и т.п.

Наверное, всем знакома ситуация, когда при работе над проектом, возникает необходимость внести изменения, но при этом нужно сохранить работоспособный вариант, в таком случае, как правило, создается новая папка, название которой скорее всего будет “Новая папка” с дополнением в виде даты или небольшой пометки, в нее копируется рабочая версия проекта и уже с ним производится работа. Со временем количество таких папок может значительно возрасти, что создает трудности в вопросе отката на предыдущие версии, отслеживании изменений и т.п. Эта ситуация значительно ухудшается, когда над проектом работает несколько человек.

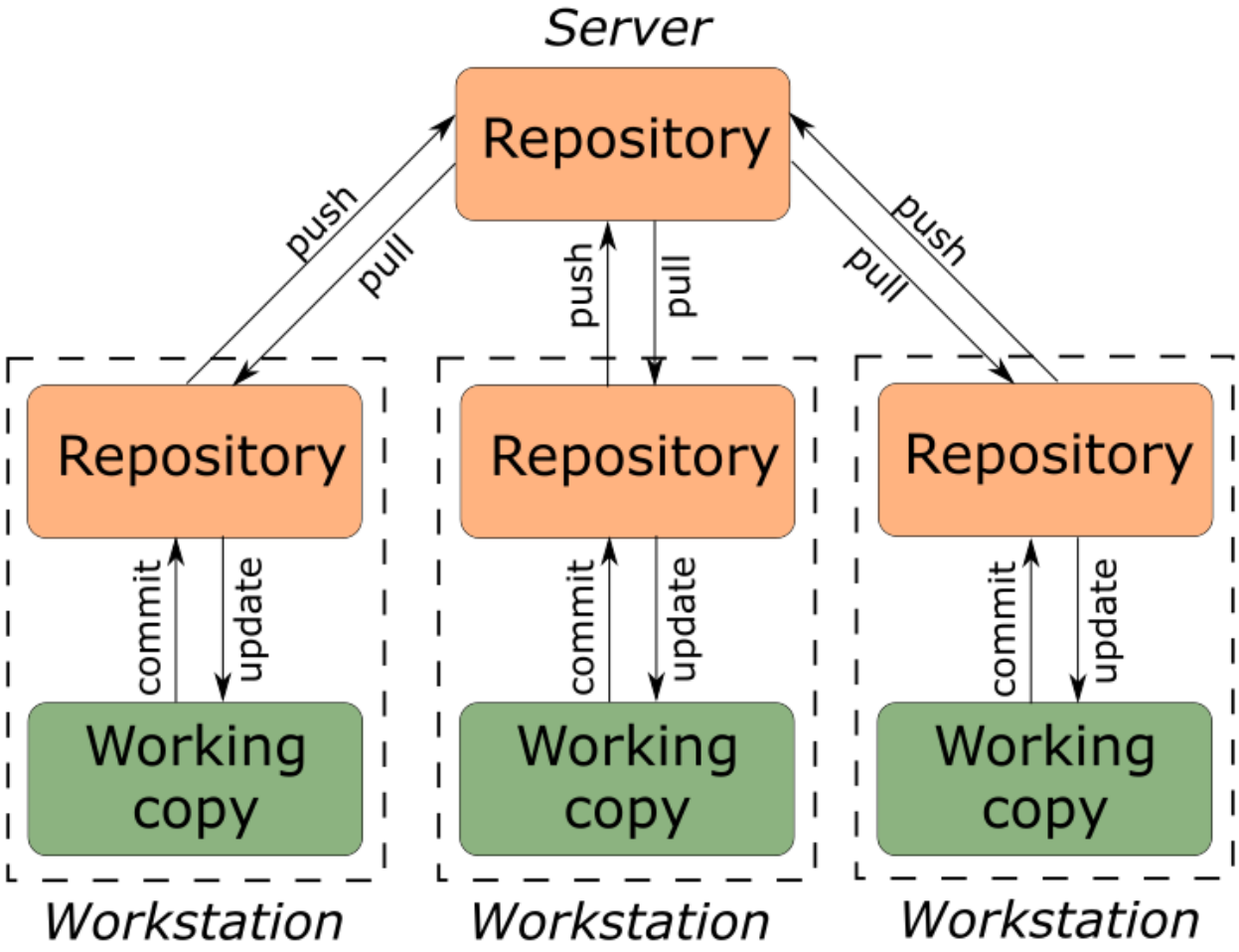
Для решения таких проблем как раз и используется система контроля версий, она позволяет комфортно работать над проектом как индивидуально, так в коллективе. Система контроля версий отслеживает изменения в файлах, предоставляет возможности для создания новых и слияние существующих ветвей проекта, производит контроль доступа пользователей к проекту, позволяет откатывать исправления и определять кто, когда и какие изменения вносил в проект.

Основным понятием системы контроля версий является репозиторий (***repository***) – специальное хранилище файлов и папок проекта, изменения в которых отслеживаются. В распоряжении разработчика имеется так называемая “рабочая копия” (*working copy*)

проекта, с которой он непосредственно работает. Рабочую копию необходимо периодически синхронизировать с репозиторием, эта операция предполагает отправку в него изменений, которые пользователь внес в свою рабочую копию (такая операция называется ***commit***) и актуализацию рабочей копии, в процессе которой к пользователю загружается последняя версия из репозитория (этот процесс носит название ***update***).

Распределенные системы контроля версий

Распределенные системы контроля версий позволяют хранить репозиторий (его копию) у каждого разработчика, работающего с данной системой. При этом можно выделить центральный репозиторий (условно), в который будут отправляться изменения из локальных и, с ним же эти локальные репозитории будут синхронизироваться. При работе с такой системой, пользователи периодически синхронизируют свои локальные репозитории с центральным репозиторием и работают непосредственно со своей локальной копией. После внесения достаточного количества изменений в локальную копию они (изменения) отправляются на сервер. При этом сервер, чаще всего, выбирается условно, т.к. в большинстве нет такого понятия как “выделенный сервер с центральным репозиторием”.



Большое преимущество такого подхода заключается в автономии разработчика при работе над проектом, гибкости общей системы и повышение надежности, благодаря тому, что каждый разработчик имеет локальную копию центрального репозитория. Наиболее известная распределенная система контроля версий - это ***Git***.

Git – распределенная система контроля версий, разработанная Линусом Торвальдсем для работы над ядром операционной системы *Linux*. Среди крупных

проектов, в рамках которых используется *git*, можно выделить ядро *Linux, Qt, Android*. *Git* свободен и распространяется под лицензией *GNU GPL* 2 и, доступен практически на всех операционных системах.

Благодаря ряду достоинств (высокая скорость работы, возможность интеграции с другими системами контроля версий, удобный интерфейс) и очень активному сообществу, сформировавшемуся вокруг этой системы, *git* вышел в лидеры рынка распределенных систем контроля версий.

Интеграция GIT, GitHub и Visual Studio

Git и ***GitHub*** – связаны, но они не одно и то же. ***Git*** – распределенная система контроля версий. Распределенная значит, что все данные хранятся не только у вас, но и на сервере (***GitHub***), и у других участников команды (если таковая имеется).

Что нужно для работы с ***Git*** с использованием сервера ***GitHub.com***.

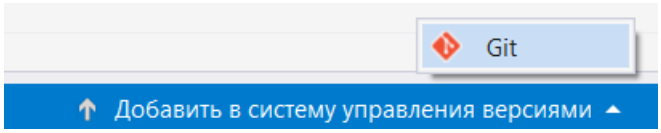
- ***Visual Studio 2019*** 16.8 или новее
- Подключение к интернету
- Несколько гигабайт памяти на жестком диске
- Регистрация на сайте ***GitHub*** где могут удаленно размещаться репозитории проектов.

Начиная с версии 16.8 Visual Studio 2019 включает полностью интегрированный интерфейс для работы с учетными записями GitHub. Теперь вы можете добавить в цепочку ключей учетные записи GitHub и GitHub Enterprise. Вы сможете добавлять и использовать их так же, как и учетные записи Майкрософт. Это позволит упростить доступ к ресурсам GitHub в Visual Studio. Дополнительные сведения см. на странице Работа с учетными записями GitHub в Visual Studio.

Создание репозитория GIT на GitHub или другом сервере Git

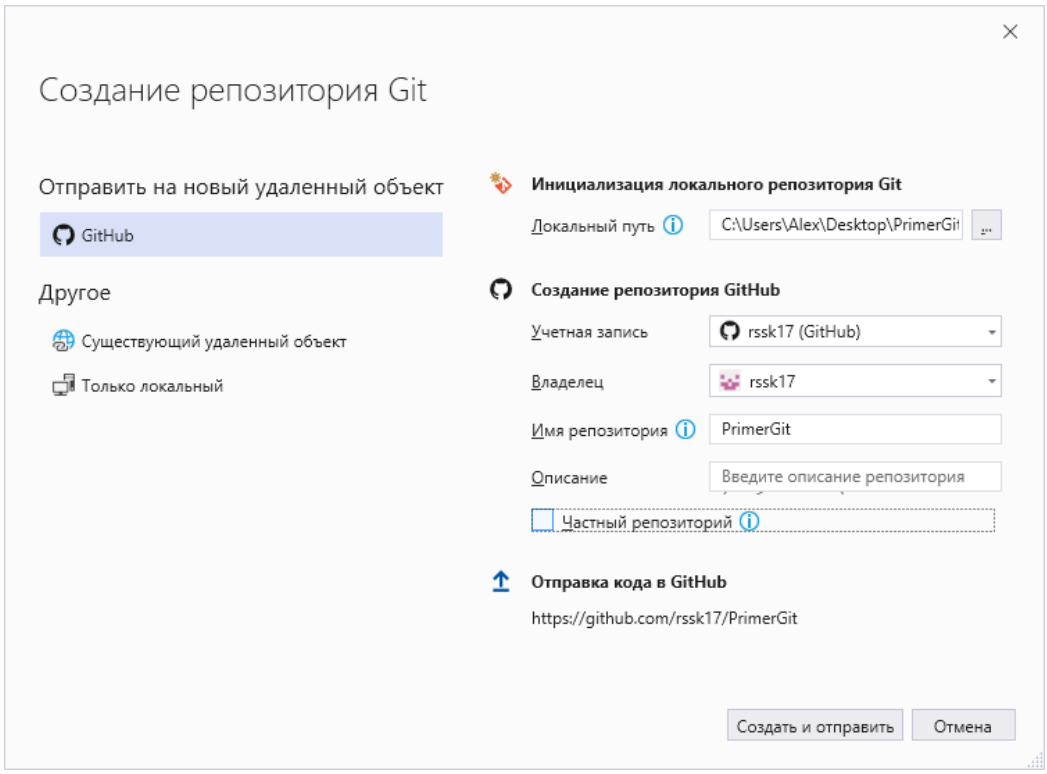
1. Создаем проект приложения, размещаем в нем необходимые визуальные элементы и программный код.

2. Добавляем проект в систему управления версиями, для этого в правом нижнем углу щелкаем надпись «***Добавить в систему управления версиями***» и выбираем тип системы ***Git*** или выбираем команду ***GIT - Создать репозиторий GIT.***

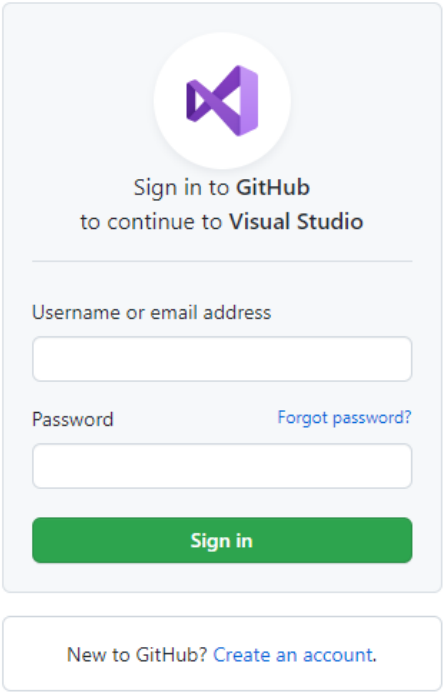


3. Открывается диалоговое окно ***Создание репозитория GIT,*** с помощью которого можно легко создать новый репозиторий в ***GitHub*** или на стороннем сервере ***Git***. Для отправки проекта в репозиторий нужно указать основные параметры:
    - сервер репозитория – ***GitHub***. При отправке на сторонний сервер ***Git*** выбираем - Существующий удаленный объект;
    - учетная запись на GitHub;

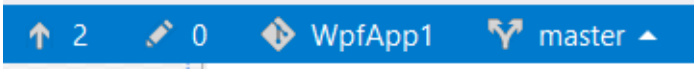
- имя репозитория, которое должно быть уникальным;
- при необходимости описание репозитория;
- тип репозитория – частный или публичный.



4. При первом использовании ***GitHub*** в ***Visual Studio*** или при переходе на другую учетную записать ***GitHub*** необходимо настроить подключение к ***GitHub***. При последующих использованиях ***GitHub*** параметры подключения запоминаются. Для настройки подключения щелкаем на элемент ***Учетная запись***, выбираем ***GitHub*** и входим с помощью браузера. При успешной авторизации в элементе Учетная запись отобразится авторизованная запись, окно браузера покажет ошибку доступа к локальной странице.



5. При успешном создании репозитория в нижнем правом углу отражается статистика фиксации изменений в репозиториях. Цифра 2 означается два неотправленных изменения на удаленный сервер, цифра 0 сколько ожидается изменений для отправки в локальный репозиторий. WpfApp1 – название нашего приложения. Master – название текущей ветки.

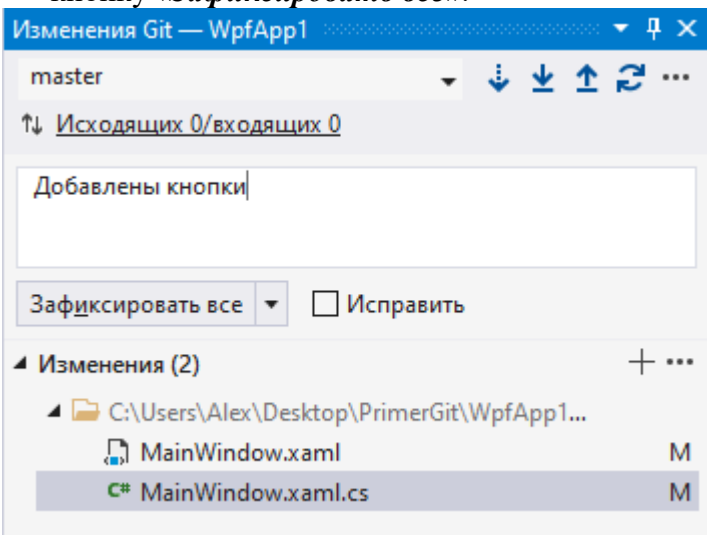


Фиксация изменений внесенных в проект в существующем репозитории

1. Открываем проект приложения, который уже ранее был подключен в систему управления версиями Git и опубликован в репозитории GitHub. Размещаем в проекте необходимые визуальные элементы и создаем далее программный код.

В нижнем правом углу видим, что есть 2 изменения не зафиксированные в локальном репозитории.

2. Щелкаем на цифру 2, открывается окно «***Изменения Git***». Пишем комментарий к произведенным изменениям, например: Добавлены кнопки. Можем просмотреть произведенные изменения, дважды щелкнув на элементы в разделе «***Изменения***». Для фиксации изменений в локальном репозитории щелкните кнопку «***Зафиксировать все***».

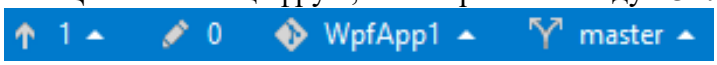


В нижнем правом углу увидим, что есть 1 изменения не зафиксированные в удаленном репозитории.

ЗАМЕЧАНИЕ: При сохранении изменений см. пункт 2. Можно вместо кнопки «***Зафиксировать все***» нажать кнопку «***Зафиксировать все и отправить***», при этом изменения сохраняются как в локальный репозиторий, так и в удаленный репозиторий на GitHub.

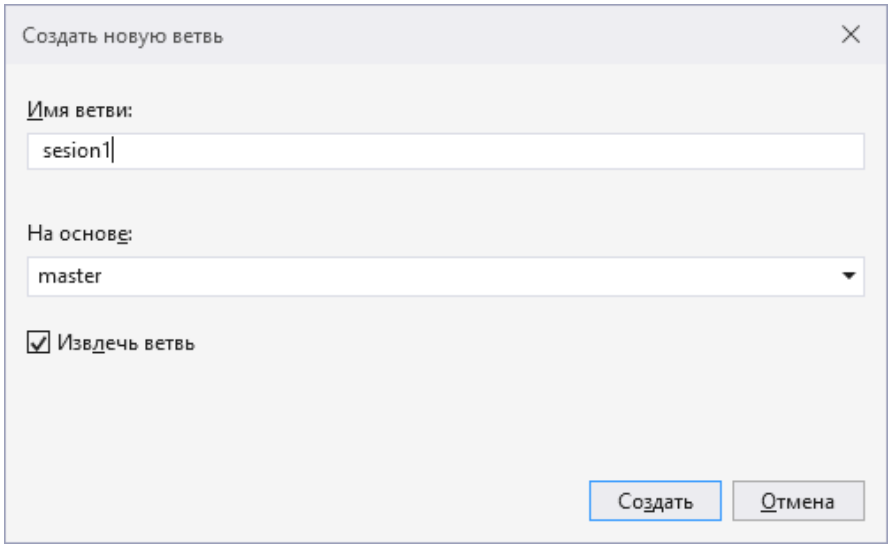
Отправка изменений GitHub

1. Если при фиксации изменений мы не отправляли их на ***GitHub***, в строке состояния в соответствующем поле видим 1 неотправленную фиксацию. Щелкните на цифру 1, и выберите команду «***Отправить***».



Создание ветви проекта

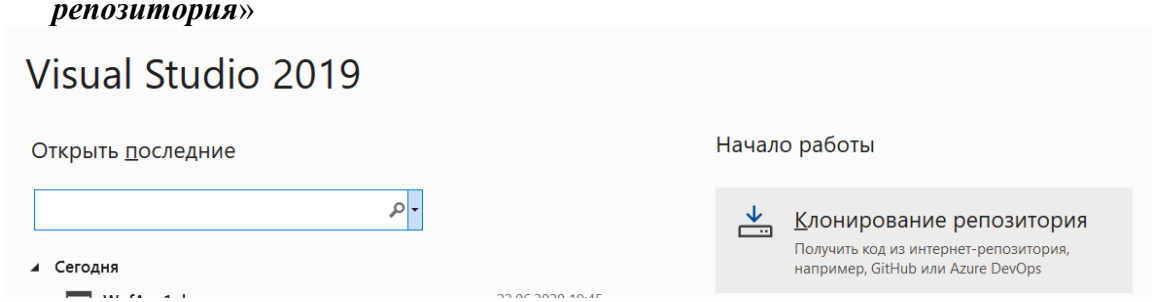
Для создания новой ветви проекта выберите команду «***Git – создать ветвь***». Введите имя новой ветви и укажите существующую ветвь, на основе которой будет создана данная. Также можно создать ветвь щелкнув на название текущей ветви в строке статуса.



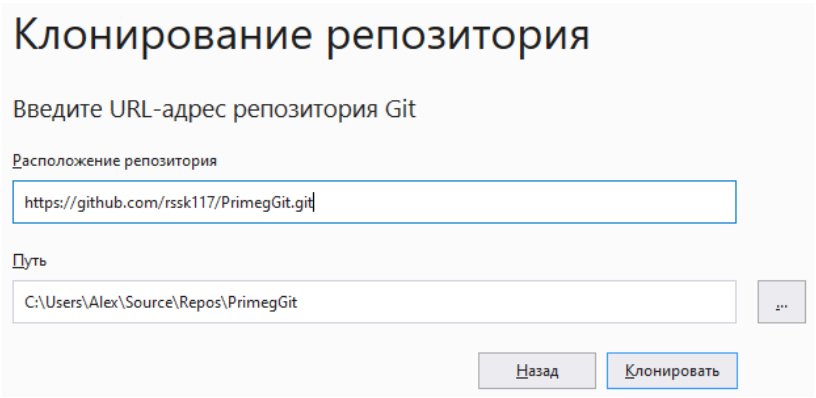
Загрузка (клонирование) проекта с GitHub

1. На GitHub находим нужный проект и копируем ссылку на него, например https://github.com/rssk17/Project1.git.

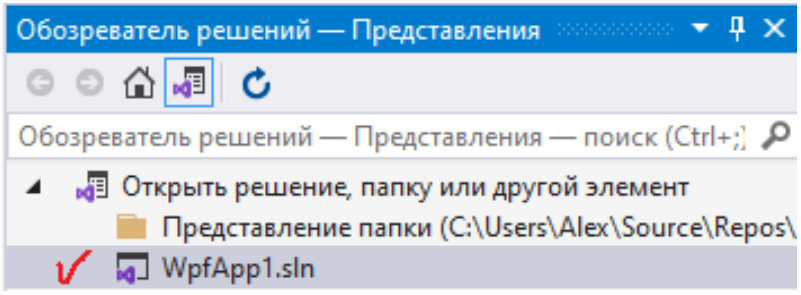
2. Запускаем Visual Studio и в стартовом окне выбираем «***Клонирование репозитория***»



3. Указываем расположение репозитория на GitHub, например https://github.com/rssk17/Project1.git Указываем локальный путь размещение проекта, нажимаем кнопку «***Клонировать***».



4. В обозревателе решений двойным щелчком на файле проекта, открываем дерево проекта в привычном виде.



Практическая работа №1

Разработка и оформление алгоритма циклической структуры.

Самостоятельная работа в домашних условиях

1. Изучить формулировку задания.
2. Разработать спецификации модулей.
3. Составить паспорта модулей (функций) производящих вычисления по заданию.
4. Написать алгоритмы вычислительных модулей (функций) программы.

Практическое занятие в учебном классе

1. Разработать библиотеку, содержащую вычислительные модули (функции) программы для решения задачи по варианту задания. Название библиотеки Lib_x, где x – номер варианта задания. Оформить модули комментариями.
2. Разработать программу.
3. Использовать для отображения информации элемент **TextBox**.
4. Массивы и таблицы не использовать.
5. Использовать меню **Menu**.
6. Добавить иконку в заголовок программы и исполняемый файл. Заполнить заголовок программы.
7. Предусмотреть в программе две кнопки «**Выход**» и «**О программе**», где вывести ФИО разработчика, номер работы и формулировку задания.
8. Оформить программу комментариями.

Варианты заданий (Базовый уровень)

1. Вычислить сумму целых случайных чисел, распределенных в диапазоне от 2 до 10, пока эта сумма не превышает некоторого числа K. Вывести на экран сгенерированные числа, значение суммы, и количество сгенерированных чисел.
2. Вычислить разницу целых случайных чисел, распределенных в диапазоне от 2 до 10, пока эта разница не станет меньше некоторого числа K (K<0). Вывести на экран сгенерированные числа, значение суммы, и количество сгенерированных чисел.
3. Генерировать случайные числа X, распределенные в диапазоне от -4 до 7 и вычислять для чисел > 0 $\sqrt{X}$, а для чисел < 0 функцию x^2. Вычисления прекратить, когда подряд появится два одинаковых случайных числа. На экран необходимо выводить сгенерированное число и результат расчета функции на разных строках.
4. Найти минимум из *n* целых случайных чисел X, распределенных в диапазоне от 10 до 40. Вывести на экран на одной строке сгенерированные числа, на другой строке результат.
5. Вычислить сумму целых случайных чисел, распределенных в диапазоне от - 7 до 3, пока эта сумма не превышает некоторого числа K. Вывести на экран сгенерированные числа, значение суммы, и количество сгенерированных чисел.
6. Генерировать случайные числа X, распределенные в диапазоне от -5 до 4 и вычислять для чисел > 0 $\sqrt{X}$, а для чисел < 0 функцию x^2. Вычисления прекратить, когда подряд появится два одинаковых случайных числа. На экран необходимо выводить сгенерированное число и результат расчета функции на разных строках.

7. Найти сумму *n* целых, случайных чисел (ДСЧ), распределенных в диапазоне от 0 до *n* и меньше *n/2*. Вывести на экран на одной строке сгенерированные числа, на другой строке результат.
8. Найти максимум из *n* целых случайных чисел X, распределенных в диапазоне от 0 до *n*. Вывести на экран на одной строке сгенерированные числа, на другой строке результат.
9. Сформировать n целых чисел в диапазоне 2-14. Найти сумму чисел < 8. Результат вывести на экран.
10. Вычислить сумму целых случайных чисел, распределенных в диапазоне от 5 до 10, пока эта сумма не превышает некоторого числа K. Вывести на экран сгенерированные числа, значение суммы, и количество сгенерированных чисел.
11. Найти произведение *n* целых случайных чисел X., распределенных в диапазоне от 0 до *n*. Вывести на экран на одной строке сгенерированные числа, на другой строке результат.
12. Найти сумму *n* целых, случайных чисел, распределенных в диапазоне от 0 до *n*. Вывести на экран на одной строке сгенерированные числа, на другой строке результат
13. Генерировать случайные числа X, распределенные в диапазоне от -5 до 5 и вычислять для чисел > 0 $\sqrt{X}$, а для чисел < 0 функцию x^2. Вычисления прекратить, когда подряд появится два одинаковых случайных числа. На экран необходимо выводить сгенерированное число и результат расчета функции на разных строках.
14. Вычислить сумму целых случайных чисел, распределенных в диапазоне от 55 до 70, пока эта сумма не превышает некоторого числа K. Вывести на экран сгенерированные числа, значение суммы, и количество сгенерированных чисел.
15. Найти сумму *n* целых, четных, случайных чисел, распределенных в диапазоне от 0 до *n*. Вывести на экран на одной строке сгенерированные числа, на другой строке результат.
16. Генерировать случайные числа X, распределенные в диапазоне от -1 до 6 и вычислять для чисел > 0 $\sqrt{X}$, а для чисел < 0 функцию x^2. Вычисления прекратить, когда подряд появится два одинаковых случайных числа. На экран необходимо выводить сгенерированное число и результат расчета функции на разных строках.
17. Найти сумму *n* целых, нечетных, случайных чисел, распределенных в диапазоне от 0 до *n*. Вывести на экран на одной строке сгенерированные числа, на другой строке результат.

Практическая работа №2

Разработка и оформление алгоритма работы с одномерным массивом

Самостоятельная работа в домашних условиях

1. Изучить формулировку задания.
2. Разработать спецификации модулей.
3. Составить паспорта модулей (функций) производящих вычисления по заданию.
4. Написать алгоритмы вычислительных модулей (функций) программы.

Практическое занятие в учебном классе

1. Разработать библиотеку, содержащую базовые модули (функции) программы для работы с любым массивом: сохранить, открыть, заполнить, очистить и т.д. Название библиотеки LibMas. Оформить модули комментариями.
2. Разработать библиотеку, содержащую вычислительные модули (функции) программы для решения задачи по варианту задания. Название библиотеки Lib_x, где x – номер варианта задания. Оформить модули комментариями.
3. Разработать программу.
4. Использовать меню **Menu**.
5. Добавить иконку в заголовок программы и исполняемый файл. Заполнить заголовок программы.
6. Предусмотреть в программе две кнопки «**Выход**» и «**О программе**», где вывести ФИО разработчика, номер работы и формулировку задания.
7. Оформить программу комментариями.

Варианты заданий (Базовый уровень)

1. Ввести *n* целых чисел. Найти сумму чисел > 5. Результат вывести на экран.
2. Ввести *n* целых чисел(>0 или <0). Найти произведение чисел. Результат вывести на экран.
3. Ввести *n* целых чисел. Найти разницу чисел. Результат вывести на экран.
4. Ввести *n* целых чисел. Вычислить для чисел > 0 функцию $\sqrt{x}$. Результат обработки каждого числа вывести на экран.
5. Ввести *n* целых чисел. Найти произведение чисел < 3. Результат вывести на экран.
6. Ввести *n* целых чисел. Найти сумму чисел < 15. Результат вывести на экран.
7. Ввести *n* целых чисел. Вычислить для чисел < 0 функцию x^2. Результат обработки каждого числа вывести на экран.
8. Ввести *n* целых чисел. Вычислить косинус (cos) суммы чисел < 3. Результат вывести на экран.
9. Ввести *n* целых чисел(>0 или <0). Найти произведение чисел. Результат вывести на экран.
10. Ввести *n* целых чисел. Вычислить для чисел > 0 функцию $\sqrt{x}$. Результат обработки каждого числа вывести на экран.
11. Ввести *n* целых чисел(>0 или <0). Найти разницу чисел. Результат вывести на экран.
12. Ввести *n* целых чисел. Найти сумму чисел >15. Результат вывести на экран.
13. Ввести *n* целых чисел. Найти произведение чисел > 2. Результат вывести на экран.
14. Ввести *n* целых чисел. Найти сумму чисел < 8. Результат вывести на экран.
15. Ввести *n* целых чисел. Вычислить sin суммы чисел > 3. Результат вывести на экран.

16. Найти сумму *n* целых, четных, случайных чисел, распределенных в диапазоне от 0 до *n*.
17. Ввести *n* целых чисел. Найти разницу четных из них чисел. Результат вывести на экран.
18. Ввести *n* целых чисел. Найти сумму чисел кратных 5. Результат вывести на экран.

Практическая работа №3

Разработка и оформление алгоритма работы с двухмерным массивом

Самостоятельная работа в домашних условиях

1. Изучить формулировку задания.
2. Разработать спецификации модулей.
3. Составить паспорта модулей (функций) производящих вычисления по заданию.
4. Написать алгоритмы вычислительных модулей (функций) программы.

Практическое занятие в учебном классе

5. При необходимости доработать библиотеку LibMas разработанную в практической работе №2, содержащую базовые модули (функции) программы для работы с любым массивом: сохранить, открыть, заполнить, очистить и т.д.
6. Разработать библиотеку, содержащую вычислительные модули (функции) программы для решения задачи по варианту задания. Название библиотеки Lib_x, где x – номер варианта задания.
7. Разработать программу.
8. Использовать меню **Menu**.
9. Добавить иконку в заголовок программы и исполняемый файл. Заполнить заголовок программы.
10. Предусмотреть в программе две кнопки «**Выход**» и «**О программе**», где вывести ФИО разработчика, номер работы и формулировку задания.
11. Оформить программу комментариями.

Варианты заданий (Базовый уровень)

I подгруппа

1. Дана матрица размера M × N и целое число K ($1 \le K \le M$). Найти сумму и произведение элементов K-й строки данной матрицы.
2. Дана матрица размера M × N и целое число K ($1 \le K \le N$). Найти сумму и произведение элементов K-го столбца данной матрицы.
3. Дана матрица размера M × N. Для каждой строки матрицы найти сумму ее элементов.
4. Дана матрица размера M × N. Для каждого столбца матрицы найти произведение его элементов.
5. Дана матрица размера M × N. Для каждой строки матрицы с нечетным номером (1, 3, …) найти среднее арифметическое ее элементов. Условный оператор не использовать.
6. Дана матрица размера M × N. Для каждого столбца матрицы с четным номером (2, 4, …) найти сумму его элементов. Условный оператор не использовать.
7. Дана матрица размера M × N. В каждой строке матрицы найти минимальный элемент.
8. Дана матрица размера M × N. В каждом столбце матрицы найти максимальный элемент.
9. Дана матрица размера M × N. Найти номер ее строки с наибольшей суммой элементов и вывести данный номер, а также значение наибольшей суммы.

10. Дана матрица размера M × N. Найти номер ее столбца с наименьшим произведением элементов и вывести данный номер, а также значение наименьшего произведения.
11. Дана матрица размера M × N. Найти максимальный среди минимальных элементов ее строк.
12. Дана матрица размера M × N. Найти минимальный среди максимальных элементов ее столбцов.
13. Дана матрица размера M × N. В каждой ее строке найти количество элементов, меньших среднего арифметического всех элементов этой строки.
14. Дана матрица размера M × N. В каждом ее столбце найти количество элементов, больших среднего арифметического всех элементов этого столбца. 21
15. Дана матрица размера M × N. Найти номера строки и столбца для элемента матрицы, наиболее близкого к среднему значению всех ее элементов.

II подгруппа

1. Дана целочисленная матрица размера M × N. Найти номер первой из ее строк, содержащих равное количество положительных и отрицательных элементов (нулевые элементы матрицы не учитываются). Если таких строк нет, то вывести 0.
2. Дана целочисленная матрица размера M × N. Найти номер последнего из ее столбцов, содержащих равное количество положительных и отрицательных элементов (нулевые элементы матрицы не учитываются). Если таких столбцов нет, то вывести 0.
3. Дана целочисленная матрица размера M × N. Найти номер последней из ее строк, содержащих только четные числа. Если таких строк нет, то вывести 0.
4. Дана целочисленная матрица размера M × N. Найти номер первого из ее столбцов, содержащих только нечетные числа. Если таких столбцов нет, то вывести 0.
5. Дана целочисленная матрица размера M × N, элементы которой могут принимать значения от 0 до 100. Различные строки матрицы назовем похожими, если совпадают множества чисел, встречающихся в этих строках. Найти количество строк, похожих на первую строку данной матрицы.
6. Дана целочисленная матрица размера M × N, элементы которой могут принимать значения от 0 до 100. Различные столбцы матрицы назовем похожими, если совпадают множества чисел, встречающихся в этих столбцах. Найти количество столбцов, похожих на последний столбец данной матрицы.
7. Дана целочисленная матрица размера M × N. Найти количество ее строк, все элементы которых различны.
8. Дана целочисленная матрица размера M × N. Найти количество ее столбцов, все элементы которых различны.
9. Дана целочисленная матрица размера M × N. Найти номер последней из ее строк, содержащих максимальное количество одинаковых элементов.
10. Дана целочисленная матрица размера M × N. Найти номер первого из ее столбцов, содержащих максимальное количество одинаковых элементов.
11. Дана матрица размера M × N. Найти количество ее строк, элементы которых упорядочены по возрастанию.
12. Дана матрица размера M × N. Найти количество ее столбцов, элементы которых упорядочены по убыванию. 22
13. Дана матрица размера M × N. Найти минимальный среди элементов тех строк, которые упорядочены либо по возрастанию, либо по убыванию. Если упорядоченные строки в матрице отсутствуют, то вывести 0.

14. Дана матрица размера M × N. Найти максимальный среди элементов тех столбцов, которые упорядочены либо по возрастанию, либо по убыванию. Если упорядоченные столбцы в матрице отсутствуют, то вывести 0.
15. Дана целочисленная матрица размера M × N. Найти элемент, являющийся максимальным в своей строке и минимальным в своем столбце. Если такой элемент отсутствует, то вывести 0.

Практическая работа №4

Изучение и настройка системы контроля версий

Самостоятельная работа в домашних условиях

1. Изучить основные принципы работы с системой контроля версий на базе GitHub.

Практическое занятие в учебном классе

1. Зарегистрироваться в системе контроля версий GitHub.
2. Выгрузить в репозиторий GitHub практические работы №1, №2 и №3.
3. Внести любое изменение в практическую работу №1 и выгрузить изменения в локальный и удаленный репозиторий.
4. Загрузить из удаленного репозитория GitHub работы на локальный компьютер.
5. Отчитаться о проделанной работе.