Министерство образования Тульской области

Государственное профессиональное образовательное учреждение

Тульской области

«Донской политехнический колледж»

РАЗРАБОТКА DESKTOP-ПРИЛОЖЕНИЯ С ИСПОЛЬЗОВАНИЕ ТЕХНОЛОГИИ WPF НА ПЛАТФОРМЕ .NET 7

Курсовая работа ОП.09

«Основы алгоритмизации и программирования»

Студент группы К-21-1 Е.Д. Махортов

Руководитель С.М. Гвоздев

Проверил \_\_\_\_\_\_\_\_\_\_\_\_

Оценка \_\_\_\_\_\_\_\_\_\_\_\_

Донской 2023

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc134463402)

[ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 4](#_Toc134463403)

[1 Высокоуровневый язык С# 4](#_Toc134463404)

[1.1 Система типов C# 4](#_Toc134463405)

[1.2 Система общих типов CTS 5](#_Toc134463406)

[1.3 Неявные типы, анонимные типы и типы, допускающие значение 7](#_Toc134463407)

[1.4 Правила и соглашения об именовании идентификаторов C# 8](#_Toc134463408)

[ПРАКТИЧЕСКАЯ ЧАСТЬ 9](#_Toc134463409)

[2 Индивидуальные задания 9](#_Toc134463410)

[2.1 Выполнение задания № 1 курсовой работы 9](#_Toc134463411)

[ЗАКЛЮЧЕНИЕ 10](#_Toc134463412)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 11](#_Toc134463413)

[ПРИЛОЖЕНИЕ А](#_Toc134463414) [Листинг программы 13](#_Toc134463415)

# **ВВЕДЕНИЕ**

Язык С# как средство обучения программированию обладает рядом несомненных достоинств. Он хорошо организован, строг, большинство его конструкций логичны и удобны. Развитые средства диагностики и редактирования кода делают процесс программирования приятным и эффективным.

Немаловажно, что С# является не учебным, а профессиональным языком, предназначенным для решения широкого спектра задач, и в первую очередь - в быстро развивающейся области создания распределенных приложений. Поэтому базовый курс программирования, построенный на основе языка С#, позволит студентам быстрее стать востребованными специалистами-профессионалами.

Актуальность темы определяется и описывается на основании состояние практики и её потребность в разрешение вопросов, составляющих тему работы, здесь можно отразить информацию о современной платформе .NET 7.

Целью исследования, проводимого в рамках настоящей курсовой работы, является разработка и реализация на языке высокого уровня C# desktop-приложения для решения алгоритмических задач, представленных в задании курсовой работы.

Объектами исследования настоящей курсовой работы являются методы и технологии разработки desktop-приложений.

Предметами исследования настоящей курсовой работы являются методы, алгоритмы и приёмы разработки desktop-приложений обработки двумерных массивов, файлов, строк.

Информационной базой исследования является учебная литература по программированию, техническая документация по языку С# инструментальной среды MS Visual Studio 2022.

# **ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

## **1 Высокоуровневый язык С#**

### **1.1 Система типов C#**

C# является строго типизированным языком. Каждая переменная и константа имеет тип, как и каждое выражение, результатом вычисления которого является значение. Каждое объявление метода задает имя, тип и вид (значение, ссылка или вывод) для каждого входного параметра и для возвращаемого значения. В библиотеке классов .NET определены встроенные числовые типы и комплексные типы, представляющие разнообразные конструкции. К ним относятся файловая система, сетевые подключения, коллекции и массивы объектов, а также даты. Обычная программа на C# использует типы из этой библиотеки классов и пользовательские типы, которые моделируют уникальные концепции конкретной сферы применения.

В типах может храниться следующая информация:

* место, необходимое для хранения переменной этого типа;
* максимальное и минимальное значения, которые могут быть представлены;
* содержащиеся в типе члены (методы, поля, события и т. д.);
* базовый тип, от которого наследует этот тип;
* реализуемые им интерфейсы;
* разрешенные виды операций.

Компилятор использует сведения о типах, чтобы проверить, все ли операции, выполняемые в коде, являются тип безопасными. Например, при объявлении переменной типа int компилятор позволяет в дополнение использовать переменную и операции вычитания. При попытке выполнить эти же операции для переменной типа bool компилятор выдаст ошибку, как показано в следующем примере:

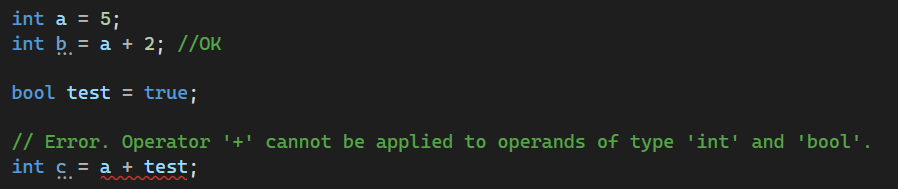


Рисунок 1.1 – Ошибка компиляции

Компилятор внедряет сведения о типе в исполняемый файл в виде метаданных. Среда CLR использует эти метаданные во время выполнения для дальнейшего обеспечения безопасности типа при выделении и освобождении памяти.

### **1.2 Система общих типов CTS**

Важно понимать две основные вещи, касающиеся системы типов, используемой в .NET:

Она поддерживает принцип наследования. Типы могут быть производными от других типов, которые называются базовыми типами. Производный тип наследует все (с некоторыми ограничениями) методы, свойства и другие члены базового типа. Базовый тип, в свою очередь, может быть производным от какого-то другого типа, при этом производный тип наследует члены обоих базовых типов в иерархии наследования. Все типы, включая встроенные числовые типы, например System.Int32 (ключевое слово C#: int), в конечном счете являются производными от одного базового типа System.Object (ключевое слово C#: object). Эта унифицированная иерархия типов называется Системой общих типов CTS. Дополнительные сведения о наследовании в C# см. в статье Inheritance (Наследование).

Каждый тип в CTS определяется как тип значения либо ссылочный тип. Это справедливо и для всех пользовательских типов, в том числе включенных в библиотеку классов .NET или определенных вами. Если в определении типа используется ключевое слово struct, он является типом значения. Например, все встроенные числовые типы определены как structs. Если в определении типа используется ключевое слово class или record, он является ссылочным типом. Для ссылочных типов и типов значений используются разные правила компиляции, и они демонстрируют разное поведение во время выполнения.

Ниже показаны взаимоотношения между типами значения и ссылочными типами в CTS.

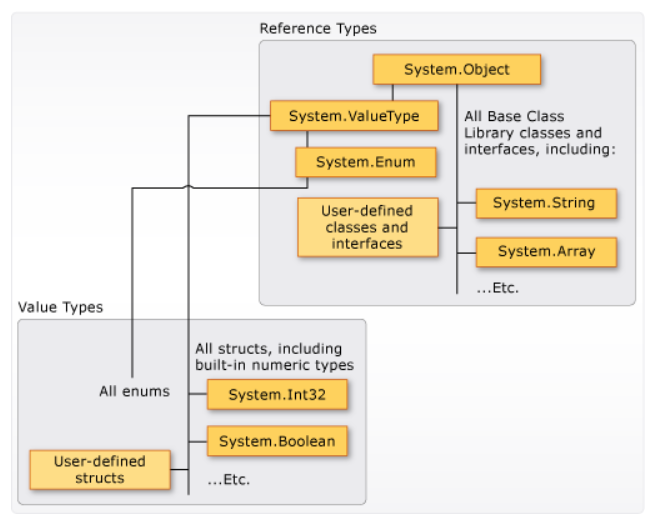


Рисунок 1.2 – Взаимоотношения между типами значения и ссылочными типами в CTS

Классы и структуры являются двумя основными конструкциями системы общих типов CTS, используемой на платформе .NET. В C# 9 добавлены записи, которые представляют собой тип класса. Оба они являются структурами данных, которые инкапсулируют набор данных и поведений в одной логической сущности. Данные и поведение являются элементами класса, структуры или записи. К ним относятся методы, свойства, события и другие элементы.

Объявление класса, структуры или записи представляет собой своего рода чертеж, на основе которого создаются экземпляры или объекты во время выполнения. Если определить класс, структуру или запись с именем Person, то Person здесь обозначает имя типа. Если объявить и инициализировать переменную p типа Person, принято говорить, что p является объектом (или экземпляром) Person. Можно создать несколько экземпляров одного типа Person, и каждый экземпляр будет иметь разные значения свойств и полей.

Класс является ссылочным типом. Когда создается объект типа и назначается его переменной, эта переменная содержит только ссылку на память объекта. Если ссылка на объект сохраняется в новую переменную, эта переменная также ссылается на исходный объект. Изменения, внесенные через одну переменную, отражаются и в другой переменной, поскольку обе они ссылаются на одни и те же данные.

Структура (struct) является типом значения. При создании структуры переменная, которой присвоена структура, содержит фактические данные этой структуры. Если структура присваивается новой переменной, все данные копируются. Таким образом, новая переменная и исходная переменная содержат две отдельные копии одинаковых данных. Изменения, внесенные в одну копию, не влияют на другую.

Типы записей могут быть либо ссылочными типами (record class), либо типами значений (record struct).

Как правило, классы используются для моделирования более сложного поведения. Классы обычно хранят данные, которые должны быть изменены после создания объекта класса. Структуры лучше всего подходят для небольших структур данных. Структуры обычно хранят данные, которые не должны изменяться после создания структуры. Типы записей — это структуры данных с дополнительными элементами, синтезированными компилятором. Записи обычно хранят данные, которые не должны изменяться после создания объекта.

### **1.3 Неявные типы, анонимные типы и типы, допускающие значение NULL**

Можно неявно типизировать локальную переменную (но не элементы класса) с помощью ключевого слова var. Такая переменная получает конкретный тип во время компиляции, но этот тип предоставляется компилятором.

Иногда нет смысла создавать именованный тип для простых наборов связанных значений, которые не будут сохраняться или передаваться за пределами метода. В таких случаях можно применить анонимные типы.

Обычные типы значений не могут иметь значение null. Но можно создать специальные типы, допускающие значения NULL, добавив символ ? после имени типа. Например, тип int? является типом int, который может иметь значение null. Типы, допускающие значение NULL, представляют собой экземпляры универсального типа структуры System.Nullable<T>. Типы, допускающие значение NULL, особенно полезны при передаче данных в базы данных, где могут использоваться числовые значения null, и из таких баз данных.

### **1.4 Правила и соглашения об именовании идентификаторов C#**

Идентификатор — это имя, присваиваемое типу (классу, интерфейсу, структуре, записи, делегату или перечислению), элементу, переменной или пространству имен. Идентификаторы должны начинаться с буквы или знака подчеркивания (\_). Идентификаторы могут содержать буквенные символы Юникода, десятичные числа, символы соединения Юникода, несамостоятельные знаки Юникода или символы форматирования Юникода. Дополнительные сведения о категориях Юникода см. в разделе База данных категорий Юникода. Вы можете объявить идентификаторы, соответствующие ключевым словам C#, с помощью префикса идентификатора @. @ не является частью имени идентификатора. Например, @if объявляет идентификатор с именем if. Эти буквальные идентификаторы предназначены главным образом для взаимодействия с идентификаторами, объявленными в других языках.

Помимо правил, существует множество соглашений об именовании идентификаторов, используемых во всех API .NET. По соглашению программы C# используют PascalCase для имен типов, пространства имен и всех открытых элементов.

# 

# **ПРАКТИЧЕСКАЯ ЧАСТЬ**

## **2 Индивидуальные задания**

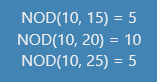
### **2.1 Выполнение задания № 1 курсовой работы**

Для нахождения наибольшего общего делителя двух натуральных чисел A и B, нужно использовать алгоритм Евклида с помощью функции NOD.

Эта функция принимает два целых аргумента a и b возвращает их наибольший общий делитель. Функция работает, рекурсивно вызывая себя с аргументами b % a до тех пор, пока a не станет равной нулю. В этот момент функция возвращает b, который является наибольшим общим делителем.



В результате получаем:



**2.2 Выполнение задания № 2 курсовой работы**

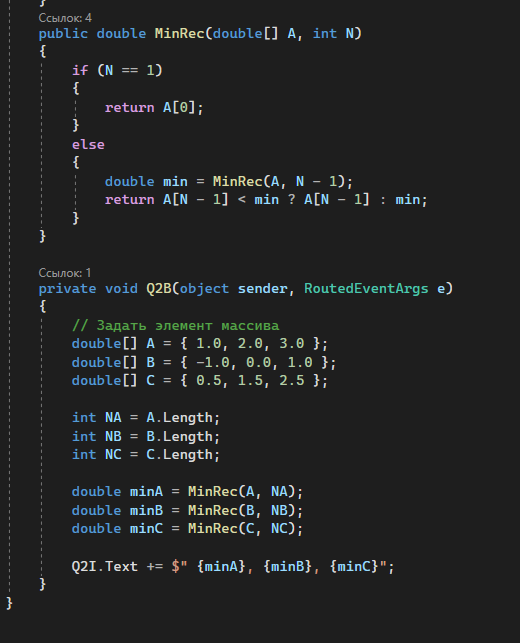
Для выполнения этого задания нам необходимо:

1. Определить базовый случай для рекурсии. Если N равно 1, вернуть первый элемент массива A.

2. В противном случае вызываем MinRec с A и N-1 в качестве аргументов и сохраняем результат в переменной min.

3. Сравниваем min с последним элементом A. Если min меньше последнего элемента A, возвращаем min. В противном случае надо вернуть последний элемент A.

Чтобы найти минимальные элементы массивов A, B и C размеров NA, NB и NC соответственно, мы можем вызвать функцию MinRec для каждого массива.

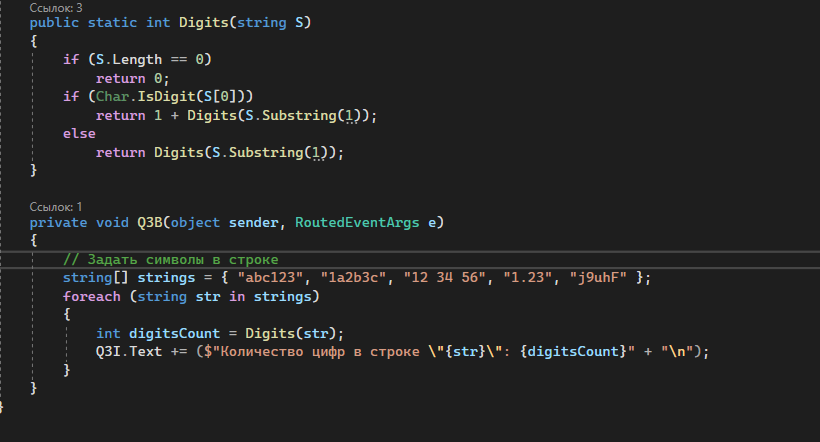


В результате получим:

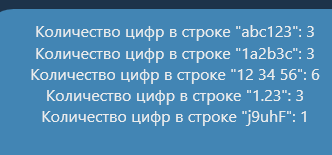


**2.3 Выполнение задания № 3 курсовой работы**

Для решения данной задачи необходимо: 1. Определить рекурсивную функцию Digits(S), которая принимает строку S в качестве входных данных и возвращает количество цифр в строке. 2. Базовый случай: если длина строки равна 0, вернуть 0. 3. Рекурсивный случай: если первый символ строки является цифрой, необходимо добавить 1 к результату и выполнить рекурсивный вызов с остальной частью строки в качестве входных данных. Если первый символ не является цифрой, выполняем рекурсивный вызов с остальной частью строки в качестве входных данных. 4. Вызываем функцию Digits(S) для каждой из 5 заданных строк.



**В результате получаем:**



**2.4 Выполнение задания № 4 курсовой работы**

Для описания рекурсивной функции Simm(S) логического типа необходимо:

1. Определяем рекурсивную функцию с именем Simm(S), которая принимает строку S в качестве аргумента и возвращает логическое значение.

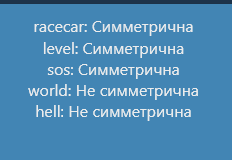
2. Базовый случай: если длина строки S равна 0 или 1, вернуть true.

3. Рекурсивный случай: если первый и последний символы строки S совпадают, удаляем их и вызываем функцию Simm с оставшейся подстрокой в ​​качестве аргумента. Если они не совпадают, вернуть false.

4. В основной программе вызываем функцию Simm с каждой из пяти заданных строк в качестве аргумента и выводим результат.



**Результат:**



**2.5 Выполнение задания № 5 курсовой работы**

Чтобы отсортировать массив в неубывающем порядке с помощью алгоритма сортировки слиянием, выполняем следующие действия:

1. Реализуем функцию merge() для объединения двух подмассивов данного массива. Эта функция должна принимать следующие параметры:

* Массив для сортировки
* Левый индекс подмассива
* Средний индекс подмассива
* Правый индекс подмассива

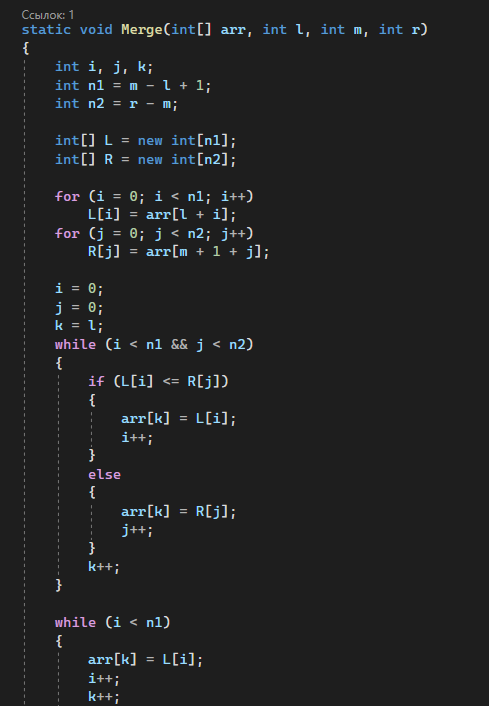
Функция должна сначала создать два временных массива для хранения левого и правого подмассивов, а затем скопировать соответствующие элементы из исходного массива в эти временные массивы. Затем функция должна сравнить элементы из левого и правого подмассивов и скопировать меньший элемент в исходный массив. Наконец, функция должна скопировать оставшиеся элементы из левого и правого подмассивов в исходный массив.

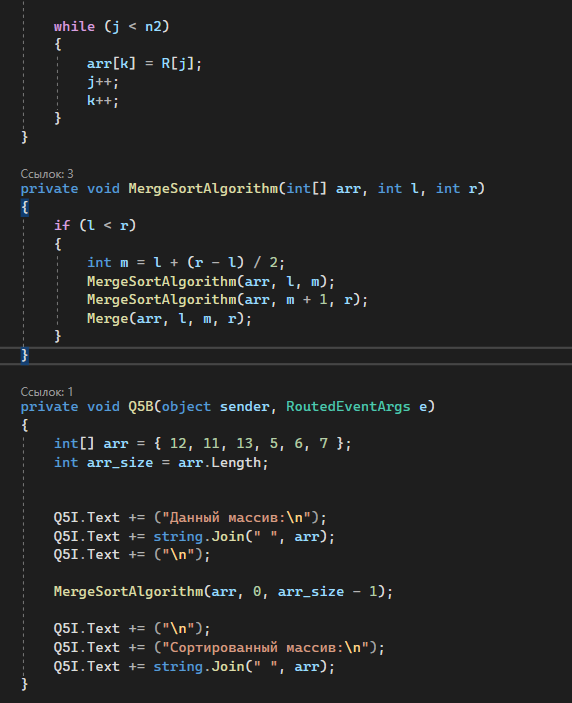
2. Реализуем функцию mergeSort() для рекурсивной сортировки заданного массива. Эта функция должна принимать следующие параметры:

* Массив для сортировки
* Левый индекс подмассива
* Правый индекс подмассива

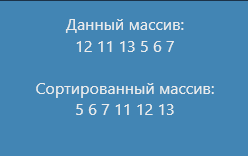
Функция должна сначала проверить, меньше ли левый индекс правого индекса. Если нет, он должен вернуться. В противном случае он должен вычислить средний индекс и рекурсивно вызвать функцию mergeSort() для левого и правого подмассивов. Наконец, он должен вызвать функцию merge() для слияния двух отсортированных подмассивов.

3. Вызываем функцию mergeSort() с заданным массивом, левым индексом (0) и правым индексом (длина массива - 1).





Результат:



# 

# **ЗАКЛЮЧЕНИЕ**

При выполнении курсовой работы были освоены основные принципы разработки алгоритмов и desktop-приложений, методы отладки и решения задач на ЭВМ в различных режимах. Так же были разобраны основы программирования и основы объектно-ориентированного подхода к программированию.

# **СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

1. **Абрамян, Михаил** Visual C# на примерах / Михаил Абрамян. - М.: БХВ-Петербург, 2017. – 496 c.
2. **Агуров, Павел** C#. Сборник рецептов / Павел Агуров. - М.: БХВ-Петербург, 2018. – 432 c.
3. **Бишоп, Дж**. С# в кратком изложении / Дж. Бишоп, Н. Хорспул. - М.: Бином. Лаборатория знаний, 2019. – 472 c.
4. **Васильев, Алексей** C#. Объектно-ориентированное программирование / Алексей Васильев. - М.: Питер, 2018. – 320 c.
5. **Зиборов, В. В**. Visual C# 2012 на примерах / В.В. Зиборов. - М.: БХВ-Петербург, 2019. – 480 c.
6. **Зиборов, Виктор** Visual C# 2010 на примерах / Виктор Зиборов. - М.: "БХВ-Петербург", 2019. – 432 c.
7. **Касаткин, А. И**. Программирование на языке си. Управление ресурсами / А.И. Касаткин. - М.: Высшая школа, 2016. – 432 c.
8. **Культин, Н.** Microsoft Visual C# в задачах и примерах / Н. Культин. - М.: БХВ-Петербург, 2017. – 314 c.
9. **Подбельский, В. В.** Язык С#. Базовый курс / В.В. Подбельский. - М.: Финансы и статистика, 2017. – 408 c.
10. **Рендольф, Ник** Visual Studio 2010 для профессионалов / Ник Рендольф и др. - М.: Диалектика, 2017. – 584 c.
11. **Скит, Джон** C# для профессионалов. Тонкости программирования / Джон Скит. - М.: Вильямс, 2017. – 608 c.
12. **Троелсен, Эндрю** Язык программирования C# 5.0 и платформа .NET 4.5 / Эндрю Троелсен. - М.: Вильямс, 2017. – 633 c.
13. **Фленов, Михаил** Библия C# / Михаил Фленов. - М.: БХВ-Петербург, 2017. – 560 c.
14. **Фримен, Адам** LINQ. Язык интегрированных запросов в C# 2020 для профессионалов / Адам Фримен, Джозеф Раттц-мл. - М.: Вильямс, 2020. – 656 c
15. **Вагнер, Б.** Эффективное программирование на C#. 50 способов улучшения кода. / Б. Вагнер. - М.: Вильямс И.Д., 2017. – 224 c.
16. **Васильев, А.** C#. Объектно-ориентированное программирование: Учебный курс / А. Васильев. - СПб.: Питер, 2017. – 320 c.
17. **Васильев, А.** C#. Объектно-ориентированное программирование. Учебный курс / А. Васильев. - СПб.: Питер, 2017. – 320 c.
18. **Васильев, А.Н.** Объектно-ориентированное программирование на C# / А.Н. Васильев. - СПб.: Наука и техника, 2018. – 544 c.
19. **Васильев, А.Н.** Программирование на C# для начинающих. Основные сведения / А.Н. Васильев. - М.: ЭКСМО, 2017. – 384 c.
20. **Васильев, А.Н.** Программирование на C# для начинающих. Особенности языка / А.Н. Васильев. - М.: ЭКСМО, 2017. – 128 c.

# **ПРИЛОЖЕНИЕ А**

# **Листинг программы**

