|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

|  |  |
| --- | --- |
| ФАКУЛЬТЕТ | Инженерный бизнес и менеджмент |
| КАФЕДРА | Промышленная логистика |

**Домашнее задание**

**по дисциплине   
«Парадигмы и конструкции языков программирования»**

***НА ТЕМУ:***

|  |
| --- |
| ***Язык программирования C#*** |
|  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент | ИБМ3-34Б |  |  |  | Булюк Мария |
|  | (группа) |  | (подпись, дата) |  | (И.О. Фамилия) |
| Преподаватель |  |  |  |  | Гапанюк Ю.Е.  Нардид А.Н. |
|  |  |  | (подпись, дата) |  | (И.О. Фамилия) |
|  |  |  |  |  |  |

Москва, 2024 г.

**Оглавление**  
1.Введение…………………………………………………………………..3  
2. Общее описание языка…………………………………………………..4  
3. Сравнение C# с другими языками программирования………………..7  
4. Использование C#...................................................................................11  
5.Заключение……………………………………………………………...13  
6. Список литературы…………………………………………………….14

# **Введение**

# Язык программирования С# был создан в конце 1990-х годов и стал частью общей .NET-стратегии Microsoft. Впервые он увидел свет в качестве б-версии в середине 2000 года. Главным архитектором С# был Андерс Хейлсберг (Anders Hejlsberg) - один из ведущих специалистов в области языков программирования, получивший признание во всем мире. Достаточно сказать, что в 1980-х он был автором весьма успешного продукта Turbo Pascal, изящная реализация которого установила стандарт для всех будущих компиляторов. С# непосредственно связан с С, C++ и Java. И это не случайно. Эти три языка - самые популярные и самые любимые языки программирования в мире. Более того, почти все профессиональные программисты сегодня знают С и C++, и большинство знает Java. Поскольку С# построен на прочном, понятном фундаменте, то переход от этих "фундаментальных" языков к "надстройке" происходит без особых усилий со стороны программистов.

# C# унаследовал синтаксис и многие ключевые слова от своего предка — языка C. Его объектная модель основана на усовершенствованиях, заложенных в C++. Поэтому знание C или C++ значительно упрощает освоение C#. Хотя C# и Java имеют общих предков (C и C++) и схожие черты, такие как переносимость кода, и похожую объектную модель, они развивались независимо, представляя собой скорее «двоюродных братьев», чем прямых потомков друг друга.

# **Общее описание языка**

Хотя C и C++ широко используются в коммерческой разработке, они часто недостаточно продуктивны. Разработка на C++, например, занимает значительно больше времени, чем на таких языках, как Visual Basic. Существующие высокопродуктивные языки, в свою очередь, жертвуют гибкостью и мощью, что неудобно для разработчиков и ограничивает возможности интеграции с современными системами и веб-технологиями. Поэтому возникла потребность в языке, сочетающем простоту Visual Basic с мощью и гибкостью C++.

В ответ на этот запрос Microsoft разработала C#. Этот язык сочетает в себе простоту, объектно-ориентированность, строгую типизацию, автоматическое управление памятью («сборка мусора»), обратную совместимость и другие преимущества, обеспечивающие ускоренную разработку приложений, особенно COM+ и веб-сервисов. C# вобрал в себя лучшие черты таких языков, как C++, C, Java, SmallTalk, Delphi и Visual Basic. Благодаря разработке «с чистого листа», C# смог избежать устаревших ограничений, свойственных многим предшествующим языкам, став современным, удобным и мощным инструментом разработки, превосходящим C++ по продуктивности.

Существует обратная зависимость между безопасностью языка программирования и производительностью создаваемых на нем программ. Assembler, например, позволяет добиться максимальной скорости, но требует значительных усилий для обеспечения корректной работы программы на разных платформах. Java, напротив, обеспечивает переносимость и безопасность, но часто страдает от недостатка скорости, особенно в клиентских приложениях.

C++ представляет собой компромисс между производительностью и безопасностью. Однако, опыт показывает, что незначительная потеря в скорости оправдана преимуществами, такими как автоматическое управление памятью («сборка мусора»). Эта функция не только упрощает разработку, избавляя от ручного управления памятью, но и предотвращает множество потенциальных ошибок. Автоматическое управление памятью и другие механизмы предотвращения ошибок становятся отличительными чертами современных языков.

C#, как современный язык, включает в себя ряд функций для повышения безопасности и надежности. Помимо «сборки мусора», C# автоматически инициализирует переменные и обеспечивает строгую типизацию, предотвращая ошибки, связанные с неинициализированными переменными или некорректными преобразованиями типов. Кроме того, C# поддерживает управление версиями (versioning), что минимизирует риски, связанные с изменением кода в уже существующих приложениях, в отличие от C++ и Java, где изменения в методах могут иметь непредвиденные последствия. Встроенная поддержка интерфейсов и наследование интерфейсов упрощает разработку и развитие сложных программных систем.

C# использует унифицированную систему типов, где каждый тип данных (классы, структуры, массивы, встроенные типы) представляется как объект. Организация кода в пространствах имён (namespaces) заменяет директивы включения заголовочных файлов, используемые в C++. Директива `using` позволяет упростить обращение к классам из определённых пространств имён.

C# значительно проще в использовании, чем C++. В частности, он минимизирует необходимость использования операторов `->` и `::`, практически всегда используя только оператор `.`, что упрощает синтаксис и уменьшает вероятность ошибок.

C# также упрощает работу с типами данных. Типы определены более интуитивно (например, `char` для символов Unicode, `long` для 64-битных целых чисел), исключая необходимость учитывать размер типов данных в зависимости от конкретной реализации, а также упраздняет различие между знаковыми и беззнаковыми символами.

В C#, как и в Visual Basic, в блоках `switch` автоматически добавляется `break` после каждой секции `case`, предотвращая непреднамеренное выполнение последующих секций. Переход к следующей секции `case` можно реализовать, например, с помощью оператора `goto`.

В отличие от C++, в C# отсутствуют указатели. Их функциональность частично заменяется делегатами (`Delegates`), которые обеспечивают типобезопасный и объектно-ориентированный аналог указателей на функции.

Пространства имен (namespaces) играют важную роль в организации кода C#, заменяя директивы включения файлов заголовков. Кроме использования директивы `using`, C# позволяет задавать псевдонимы (aliases) для пространств имен и классов.

C# упрощает отладку с помощью атрибутов, которые позволяют условно включать/выключать фрагменты кода во время разработки, что эффективнее, чем традиционные директивы препроцессора (#ifdef).

C# поддерживает типизированные расширяемые метаданные, позволяющие связать программный код с абстрактными бизнес-процессами. Атрибуты, прикрепленные к классам, интерфейсам и другим элементам, обеспечивают программую проверку соответствия кода бизнес-логике, упрощая, например, автоматизированное тестирование и анализ.

# **Сравнение C# с другими языками программирования**

C#, будучи одним из последних широко используемых языков программирования, призван объединить лучшие практики существующих языков, при этом будучи специально разработанным для работы с платформой .NET. Объектно-ориентированная архитектура .NET определила и его объектно-ориентированную природу, хотя теоретически на .NET возможна разработка компиляторов и для функциональных языков.

Синтаксис C# во многом унаследован от C++ и Java, предоставляя разработчикам с опытом работы на этих языках комфортную среду. Однако, C# также внес инновации: атрибуты, делегаты и события, которые стали неотъемлемой частью .NET-разработки и открыли новые подходы к программированию.

Часто C# сравнивают с Java. Оба языка предназначены для виртуальных сред выполнения, имеют объектно-ориентированную архитектуру и сборку мусора, основанную на ссылках. Однако C# отличается возможностью объявления нескольких классов в одном файле, наличием иерархической системы пространств имен, а также поддерживает перечисления контейнеров. В области объектно-ориентированного программирования сходство проявляется в механизме наследования (одинарное наследование, множественная реализация интерфейсов), но Java не поддерживает свойства, индексаторы, делегаты и события.

Спецификация C# включает в себя не только программистские аспекты. Важной особенностью является возможность использования XML-комментариев для автоматической генерации документации.

В заключение, C# внес собственный вклад в мир программирования с помощью уникальных элементов, таких как события, индексаторы, атрибуты и делегаты, которые обеспечивают расширенные возможности.

**Использование C#**

В этой части будут рассмотрены практические аспекты использования C#. Компиляция, отладка и тому подобные вопросы. Начнем с компиляции:

Переходя к более подробному знакомству с C#, традиционно рассмотрим программу "Hello, world": using System;

class Hello{

static void Main() {

Console.WriteLine("hello, world");

}

}

Поместите эту программу в файл hello.cs и скомпилируйте ее командой

csc hello.cs

Так выглядит вызов компилятора в простейшем случае. Далее мы приведем полный список параметров компилятора. При вызове компилятора убедитесь, что у вас прописан путь csc.exe - C Sharp Compiler

В результате вы получите файл hello.exe, запустив который, вы увидите надпись "hello, world".

Метод Main - это точка входа в программу. С него начинается выполнение (те, кто знаком с C++ или Java) понимают, о чем идет речь. Этот метод обязательно должен быть статическим. Далее в нашем примере используется метод WriteLine из класса System.Console. Этот метод отправляет строку на стандартный вывод.

Вы можете также создавать библиотеки классов - dll-сборки, которые могут использоваться другими приложениями. Для этого нужно указать /target:library в опциях компилятора

Если компилятор обнаружит ошибки в программе, он выдаст соответствующее сообщение и остановит процесс компиляции. Также он может выдать предупреждения - если он обнаружит код, который, строго говоря, правильный, но имеет подозрения на ошибку. К таким предупреждениям стоит прислушиваться.

Вообще, компилятор C# имеет множество возможностей, узнать о которых можно из слеующей таблицы. В ней описаны все опции командной строки компилятора CSC:

Выходные файлы/out:<file> Имя выходного файла (если не указано - производное от имени первого исходного файла)

/target:exe Скомпилировать консольный запускаемый файл (по умолчанию) (Краткая форма: /t:exe)

/target:winexe Скомпилировать запускаемый файл Windows (Краткая форма: /t:winexe)

/target:library Скомпилировать в библиотеку DLL (Краткая форма: /t:library)

/target:module Скомпилировать модуль, который может быть добавлен в другую конструкцию (Краткая форма: /t:module)

\* Примечание: Модуль и библиотека имеют одинаковые расширения (DLL), но несмотря на это, они представляют из себя разные вещи. Библиотека - это классическая библиотека DLL. Модуль - это файл откомпилированный без создания конструкции (assembly). В дальнейшем вы можете добавить модуль в любую другую конструкцию при компиляции (с помощью опции /addmodule). /nooutput[+|-] Только проверять ошибки в коде; не создавать запускаемый файл

/define:<список символов> Определяет директивы препроцессора (Краткая форма: /d)

Написание csc /define:mark1 MyApp.cs равносильно наличию в файле строки

#define mark1

/doc:<file> Создать XML файл документации

Входные файлы/recurse:<маска> Включить все файлы, удовлетворяющие маске.

/main:<тип> Обозначить класс, содержащий точку входа в программу (все остальные будут игнорироваться) (Краткая форма: /m)

/reference:<список файлов> Включить метаданные из указанных файлов конструкций (Краткая форма: /r)

\* Примечание: все данные, помеченные как public, из указаных файлов конструкций будут включены в данную конструкцию. Если файл не содержит объявления конструкции, то метаданные из него могут быть включены с помощью /addmodule./addmodule:<список файлов> Включить указанные модули в сборку

Ресурсы/win32res:<file> Добавляет файл ресурсов Win32(.res) в выходной файл

/win32icon:<file> Добавляет иконку в выходной файл

/resource:<resinfo> Включает .NET ресурс (Краткая форма: /res)

\* Примечание: resinfo являет собой следующую последовательность: filename[,identifier[,mimetype]], где filename - имя файла ресурса, identifier - логическое имя ресурса (используется для его загрузки), mimetype - медиа тип ресурса (обычно отсутствует). /linkresource:<resinfo> Создает ссылку на ресурсный файл в данной сборке (не включая его в выходной файл)(Краткая форма: /linkres)

Создание кода/debug[+|-] Создавать информацию отладки.

/debug:{full|pdbonly} Указание типа отладки ('full' - по умолчанию, позволяет прикреплять отладчик к запускаемой программе)

/optimize[+|-] Запускать оптимизацию (Краткая форма: /o)

/incremental[+|-] Запускать компиляцию только измененных частей кода(Краткая форма: /incr)

Ошибки и предупреждения /warnaserror[+|-] Относиться к предупреждениям, как к ошибкам

/warn:<n> Установить уровень предупреждений (0-4) (Краткая форма: /w)

/nowarn:<список предупреждений> Исключить указанные предупреждения

\* Примечание: список предупреждений - последовательность их номеров num1[,num2[...]]

Язык/checked[+|-] Выполнять проверки на переполнения и опустошения

/unsafe[+|-] Допускать unsafe код (unsafe код - небезопасный код, он будет обсужден в дополнительных главах)

Разное@<file> Прочтение команд и опций компилятора из файла

/help Показывает информацию о использовании (Краткая форма: /?)

/nologo Не показывать копирайт при компиляции

Дополнительно/baseaddress:<address> Базовый адрес для создаваемой библиотеки

/bugreport:<file> Создавать файл отчета об ошибках

/codepage:<n> Указать таблицу символов, для использования во время открытия исходного текста

/fullpaths Компилятор будет указывать полные имена файлов

/nostdlib[+|-] Не допускать включения стандартной библиотеки (mscorlib.dll). В этой библиотеке хранятся все основные классы .NET Framework.

# **Заключение**

Благодаря удобному объектно-ориентированному дизайну, C# идеально подходит для быстрой разработки разнообразных компонентов — от высокоуровневой бизнес-логики до низкоуровневых системных приложений.

C# также ориентирован на веб-разработку: его встроенные средства позволяют легко превращать компоненты в веб-сервисы, доступные через Интернет с любых платформ и на любых языках программирования. Использование передовых веб-технологий, таких как XML и SOAP, расширяет возможности C# и обеспечивает преимущества перед другими языками.

Среда разработки веб-сервисов в C# позволяет рассматривать существующие веб-приложения как объекты C#, что упрощает работу разработчиков, знакомых с принципами объектно-ориентированного программирования, и обеспечивает эффективное взаимодействие с существующими веб-сервисами.

# **Список использованных источников**

# 1. Лахатин, А.С. Языки программирования. Учеб. пособие / А.С. Лахатин, Л.Ю. Искакова. - Екатеринбург, 1998. - 548с.: ил.

# 2. Богатырев, А. Язык программирования С [Электронный ресурс] / А. Богатырев.- электр. дан. - Режим доступа: http://www.refby.com. - Загл. с экрана.

# 3. Уэйт, М. Язык С. Руководство для начинающих. / М. Уэйт, С. Прага, Д. Мартин. - М.: Мир, 1995. - 521с.: ил.

# 4. Фридман, А.Л. Язык программирования С++ / А.Л.Фридман. - М.: Бином, 2006. - 523с.: ил.