**BigSharp**

Для разработчиков:

Путь к исполняемому файлу: [DownloadFolder]\BigSharpCompiler\bin\Debug\net6.0\BigSharpCompiler.exe

Описание:

BigSharp – преемник C#. Он имеет похожий синтаксис, то же ООП, но немного изменен. В нем Вы можете выбирать, какую типизацию хотите использовать. После 2\_147\_483\_647 у Вас будет не -2\_147\_483\_648, а 2\_147\_483\_648. В приемнике C# отсутствуют неточности в вычислениях (сведены к минимуму). Повышена читабельность кода. Отлично подходит для написания простеньких консольных программ или для их прототипирования.

Отличия BigSharp от C#:

Плюсы:

1. 100 знаков после запятой
2. 5/3 = 1,666666..., а не 1
3. локальные константы (в разработке)
4. сделать переменную константой не сразу (в разработке)
5. числа бесконечного размера
6. степени через “^”
7. нет goto, теперь лишь “break POINT” и “continue POINT”
8. никаких неточностей в не целых числах
9. задержки на бесконечное время
10. задержки в секундах
11. динамические переменные (var. См. объяснение №1)
12. строго типизированные переменные (freeze. См. объяснение №2)
13. короткий код с использованием функции при объявлении значения

(3.WriteLine(); 3.Also(SomeMethod))

1. оптимизация методов с использованием атрибута “[Optimized]”

(при условии, что метод возвращает значения. См. объяснение №3)

1. простое объявление листов и словарей

(var a = list {3, “Hello”, 3.44}; var b = dict {3, “Hello”, 3.44})

1. нет точки с запятой
2. elif вместо else if
3. строки с переносами
4. “f” строки, вместо “$”
5. FastString, вместо string
6. Строки можно изменять по символам
7. Строки можно умножать и складывать
8. Массивы, списки и прочее начинаются с единицы
9. Проверить, есть ли значение в списке, можно с помощью оператора “in”

((3, 2) in {(3, 2), (4, 6)}. См. объяснение №4)

1. Можно использовать перечисление дней недели (Console.WriteLine(Friday))
2. Оптимизируются все методы

(при компиляции создаются атрибуты Inlining и Optimization)

Минусы:

1. Низкая скорость

(в худшем случае, из-за безграничных чисел и динамической типизации, скорость снижается в 10 раз, по сравнению с C#)

Объяснения:

1. Как работают динамические переменные?

Все просто, на этапе компиляции, “var” заменяется на “dynamic”, что разрешает переменной менять свой тип.

1. Как работают строго типизированные переменные?

Все просто, на этапе компиляции, “freeze” заменяется на “var”, что запрещает переменной менять свой тип.

1. Как работает оптимизация при атрибуте “[Optimized]”?

Создается словарь, который содержит в себе все аргументы, приведенные к string. В начале метода пишется условие: если словарь содержит ключ (аргументы приведенные к строке), то возвращается значение, сохраненное в словаре.

Перед return производится добавление в словарь. Добавляемый ключ в словарь является аргументами, приведенными к строке, а добавляемое значение копируется из return

Поэтому **не используйте** атрибут “[Optimized]”, если метод для вычисления значения использует глобальные переменные, привязан ко времени или имеет значимую задержку.

1. Как работает оператор “in”?

На этапе компиляции, искомое значение и список заворачиваются в “Enumerable.Contains”. Первый аргумент до оборачивания – значение, для проверки на вхождение, Второй аргумент до оборачивания – массив/лист в котором производится проверка.

Если объект, в котором производится проверка является словарем, то используется метод ContainsKey