

Министерство образования Республики Беларусь
Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина «Методы трансляции»

ОТЧЕТ
к лабораторной работе № 1
на тему «Определение модели языка. Выбор инструментальной
языковой среды»

Выполнил

Е. В. Савончик

Проверил

Н. Ю. Гриценко

Минск 2024

СОДЕРЖАНИЕ

1 Постановка задачи.....	3
2 Определение подмножества языка программирования C#	4
3 Определение инструментальной языковой среды	10
Выводы	11
Список использованных источников	12
Приложение А (обязательное) Пример реализации программ на языке программирования C#	13

1 ПОСТАНОВКА ЗАДАЧИ

Целью выполнения данной лабораторной работы является определить подмножество выбранного языка программирования, предоставить тексты двух или трех программ, включающих все элементы этого подмножества, а также определить инструментальную языковую среду, которая включает в себя язык программирования, на котором ведется разработка, операционная система, в которой выполняется разработка, и компьютер. ольких программ.

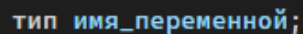
2 ОПРЕДЕЛЕНИЕ ПОДМНОЖЕСТВА ЯЗЫКА ПРОГРАММИРОВАНИЯ C#

Литералы – это неизменяемые значения, которые иногда также называют константами. Они могут быть использованы для передачи значений переменным. В языке C# литералы могут быть логическими (true/false), целочисленными, вещественными, символьными и строковыми. Отдельно стоит упомянуть литерал ключевого слова null, который представляет отсутствие значения. Подмножество числовых и текстовых констант в языке C# определено следующим образом:

- целочисленные константы, представленные различными системами исчисления: 1, -22, 2e10, 00, 071, 0x0, 0x2AF;
- вещественные константы: 3.0, 3., .123, 3.14e-1;
- символьные константы: «a», «T», «\r»;
- текстовые константы: «Это строковая константа»;

Для хранения данных в программе на языке C# применяются переменные. Переменная представляет именованную область памяти, в которой хранится значение определенного типа. Переменная имеет тип, имя и значение. Тип определяет, какого рода информацию может хранить переменная. [1]

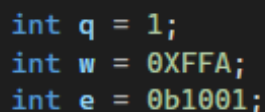
Перед использованием любую переменную надо определить. Пример синтаксиса определения переменной в языке программирования C# показан на рисунке 2.1.



```
тип имя_переменной;
```

Рисунок 2.1 – Инициализация переменной.

В языке программирования C# целочисленные типы представляют целые числа и являются типами значений. Они относятся к простым типам данных и могут быть инициализированы с помощью литералов. Все целочисленные типы поддерживают арифметические операторы, логические операторы, операторы сравнения и операторы равенства. Пример переменной целочисленного типа данных в языке программирования C# показан на рисунке 2.2.



```
int q = 1;  
int w = 0XFFFA;  
int e = 0b1001;
```

Рисунок 2.2 – Целочисленный тип данных.

Для хранения дробных чисел в C# применяются числа с плавающей точкой. Число с плавающей точкой состоит из двух частей: мантиссы и показателя степени. Обе части могут быть как положительными, так и отрицательными. Величина числа представляет собой мантиссу, умноженную на десять в степени экспоненты. В качестве разделителя целой и дробной части используется символ точки.

В языке C# присутствуют два типа данных для представления чисел с плавающей точкой: `float` и `double`.

1 Тип данных `float` представляет вещественное число одинарной точности с плавающей точкой и занимает в памяти 4 байта.

2 Тип данных `double` представляет вещественное число двойной точности с плавающей точкой и занимает в памяти 8 байт.

В языке C# отсутствует тип данных `long double`, который присутствует, например, в других языках программирования, таких как C++. Вместо этого, тип `double` используется для представления чисел с плавающей точкой двойной точности с достаточно высокой точностью.

К символьным типам данных в языке программирования C# относятся `char`. Пример использования переменной символьного типа данных в языке программирования C# показан на рисунке 2.3.

```
char a = 'A';  
char s = '\n';  
char d = '\u0321';
```

Рисунок 2.3 – Символьный тип данных.

Логический тип `bool` может хранить одно из двух значений: `true` либо `false`. Логический тип в основном применяется в условных выражениях. Значением по умолчанию для переменных этого типа является значение `false`.

Переменные типа `char` хранят числовой код одного символа и занимает один байт. Обычно для символов `char` используется кодировка ASCII. Также в C# можно использовать специальные управляющие последовательности.

Строковый тип данных представлен в C# типом данных `String` которые представляет собой удобный способ работы со строками. `String` автоматически управляет памятью и длиной строки. Также строковые переменные можно организовывать при помощи массива символов. Пример переменной строкового типа данных в языке программирования C# показан на рисунке 2.4.

```
string test1 = "test1";  
string test2 = "\rexample\\n";
```

Рисунок 2.4 – Строковый тип данных.

В случае затруднительного определения типа выражения можно применить спецификатор `var`. При определении переменной со спецификатором `var`, эта переменная должна быть обязательно инициализировано каким-либо значением.

Тип данных `object` может хранить значение любого типа данных и занимает 4 байта на 32-разрядной платформе и 8 байт на 64-разрядной платформе. Представлен системным типом `System.Object`, который является базовым для всех других типов и классов .NET. Пример переменной типа данных `object` в языке программирования C# показан на рисунке 2.5.

```
object o1 = 150;  
object o2 = "string object";  
object o3 = null;
```

Рисунок 2.5 – Тип данных `object`.

В языке C# существует различные структуры данных, подходящие под различные задачи.

`Dictionary` – это структура данных, представляющая собой специальным образом организованный набор элементов хранимых данные. Все данные хранятся в виде пар ключ-значение. Доступ к элементам данных осуществляется по ключу

`Array` – структура данных, хранящая набор значений (элементов массива), идентифицируемых по индексу или набору индексов, принимающих целые (или приводимые к целым) значения из некоторого заданного непрерывного диапазона.

`List` – структура данных, состоящая из элементов, содержащих помимо собственных данных ссылки на следующий и/или предыдущий элемент списка. С помощью списков можно реализовать такие структуры данных как стек и очередь.

`Stack` – структура данных, представляющая из себя упорядоченный набор элементов, в которой добавление новых элементов и удаление существующих производится с одного конца, называемого вершиной стека.

`Queue` – это структура данных, добавление и удаление элементов в которой происходит путём операций и соответственно. При этом первым из очереди удаляется элемент, который был помещен туда первым, то есть в очереди реализуется принцип FIFO.

Цикл состоит из условия и тела цикла. Код, находящийся в теле, выполняется, когда условие равно `true`. Каждое повторение цикла называется итерацией. [2]

Подмножество операторов циклов языка C# определено как:

- операторы while, do while;
- оператор for;
- операторы break, continue.

Цикл for используется для выполнения блока кода заданное количество раз.

Цикл while используется для выполнения блока кода до тех пор, пока условие истинно, условие проверяется до выполнения каждой итерации.

Цикл do-while похож на цикл while, но условие проверяется после каждой итерации, так что тело цикла выполняется хотя бы один раз.

Функция – это самостоятельная единица программы, которая спроектирована для реализации конкретной подзадачи. Функция является подпрограммой, которая может содержаться в основной программе, а может быть создана отдельно (в библиотеке). Каждая функция выполняет в программе определенные действия.

Сигнатура функции определяет правила использования функции. Обычно сигнатура представляет собой описание функции, включающее имя функции, перечень формальных параметров с их типами и тип возвращаемого значения.

Семантика функции определяет способ реализации функции. Обычно представляет собой тело функции.

Локальные функции представляют функции, определенные внутри других методов. Локальная функция, как правило, содержит действия, которые применяются только в рамках ее метода. [3]

Пример использования функции в языке программирования C# показан на рисунке 2.6.

```
public string Encrypt(string source)
{
    StringBuilder result = new StringBuilder();

    foreach (char letter in source)
    {
        if(char.IsLetter(letter) && char.IsUpper(letter))
        {
            result.Append((char)((letter + Step - 'A') % ALPHABET_LENGTH + 'A'));
        }
        else if(char.IsLetter(letter))
        {
            result.Append((char)((letter + Step - 'a') % ALPHABET_LENGTH + 'a'));
        }
        else
        {
            result.Append(letter);
        }
    }

    return result.ToString();
}
```

Рисунок 2.6 – Пример функции.

Класс представляет собой шаблон, по которому определяется форма объекта. В нем указываются данные и код, который будет оперировать этими данными. В С# используется спецификация класса для построения объектов, которые являются экземплярами класса. Следовательно, класс, по существу, представляет собой ряд схематических описаний способа построения объекта. При этом очень важно подчеркнуть, что класс является логической абстракцией. Физическое представление класса появится в оперативной памяти лишь после того, как будет создан объект этого класса. [2]

Классы и структуры в С# представляют собой шаблоны, по которым можно создавать объекты. Каждый объект содержит данные и методы, которые манипулируют этими данными.

При определении класса указываются данные, которые он содержит, а также код, который оперирует этими данными. Хотя простейшие классы могут содержать либо только код, либо только данные, большинство реальных классов содержат и то, и другое.

Данные хранятся в членах данных, которые определяются классом, а код находится в методах-членах. В языке С# существует несколько разновидностей членов данных и методов-членов.

Операторы в языке С# представлены арифметическими, логическими, сравнительными операциями.

Арифметические операторы:

- сложения (+);
- вычитания (-);
- деления (/);
- умножения (*);
- деления с остатком (%);
- инкремент (++);
- декремент (--).

Логические операторы:

- логическое И (&&);
- логическое ИЛИ (||);
- логическое НЕ (!).

Операторы сравнения:

- оператор равенства (==);
- оператор неравенства (!=);
- оператор больше (>);
- оператор меньше (<);
- оператор больше или равно (>=);
- оператор меньше или равно (<=).

Также в С# представлен тернарный оператор предоставляет, который предоставляет удобный способ выбора одного

из двух возможных вариантов действий на основе значения логического выражения.

Операции присваивания:

- базовая операция присваивания (=);
- присваивание после сложения (+=);
- присваивание после вычитания (-=);
- присваивание после умножения (*=);
- присваивание после деления (/=);
- присваивание после получения остатка от деления (%=);
- присваивание после поразрядной конъюнкции (&=);
- присваивание после поразрядной дизъюнкции (|=);
- присваивание после операции исключающего ИЛИ (^=).

Операторы доступа к элементам

- операторы квадратных скобок ([]);
- оператор точки (.);
- оператор двойной точки (::).

Условный оператор в С# предназначен для выбора к исполнению одного из возможных действий в зависимости от некоторого условия. Если условие после слова `if` верно, то выполняется оператор 1, в другом случае оператор 2 после команды `else`.

Подмножество условных операторов языка С# включает в себя операторы `if`, `else if`, `else` и операторы `switch case`.

В языке программирования С# для интеграции библиотек используется ключевое слово `using`. Это ключевое слово предназначено для включения содержимого сторонних библиотек. После включения библиотеки заголовочные файлы становятся доступными для использования в программе, что позволяет обращаться к их функциям, классам и другим компонентам.

3 ОПРЕДЕЛЕНИЕ ИНСТРУМЕНТАЛЬНОЙ ЯЗЫКОВОЙ СРЕДЫ

Для разработки транслятора был выбран язык программирования Python 3.12.

Главное достоинство Python – простота синтаксиса и команд, а также большое количество библиотек, которые содержат уже написанный программный код для решения широкого спектра задач. Python даже применяют в своих исследованиях и разработках специалисты, чьи профессии напрямую не связаны с программированием. Один из самых частых примеров – применение Python для анализа большого количества данных и нахождения корреляции между ними. [4]

Python обладает важным преимуществом в автоматическом управлении памятью, благодаря механизмам подсчета ссылок и циклического сборщика мусора. Это означает, что при разработке программ на Python нет нужды явно заботиться о выделении и освобождении памяти, что делает процесс программирования более простым и менее подверженным ошибкам.

В дополнение, Python является кроссплатформенным языком программирования, что позволяет запускать программы, написанные на нем, на различных операционных системах без необходимости изменения исходного кода.

В качестве интегрированной среды разработки был выбран Visual Studio Code от Microsoft.

Операционная система инструментальной языковой среды является Windows.

Компьютер, на котором ведётся разработка является персональным компьютером.

ВЫВОДЫ

В ходе выполнения данной лабораторной было определено подмножество языка программирования (типы констант, переменных, операторов и функций), определены язык программирования и операционная система для разработки, а также реализованы тексты нескольких программ.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Переменные [Электронный ресурс]. – Режим доступа: <https://metanit.com/sharp/tutorial/2.25.php/> – Дата доступа: 04.02.24
- [2] Циклы [Электронный ресурс]. – Режим доступа: <https://tproger.ru/explain/typy-ciklov-v-jazykah-programmirovanija-for-foreach-while-i-do-while>. – Дата доступа: 04.02.24
- [3] Функции [Электронный ресурс]. – Режим доступа: <https://metanit.com/sharp/tutorial/2.20.php>. – Дата доступа: 04.02.24
- [4] Основы Python [Электронный ресурс]. – Режим доступа: <https://education.yandex.ru/handbook/python/article/intro>. – Дата доступа: 05.02.2024.

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг кода

Листинг 1 – Файл example1.cs

```
class Program
{
    static void Main()
    {
        // Объявление переменных
        int a = 5;
        int b = 3;
        string input;

        // Вывод текста на консоль
        Console.WriteLine("Пример на C#.");

        // Чтение строки с консоли
        Console.Write("Введите что-нибудь: ");
        input = Console.ReadLine();

        // Операции с переменными
        int sum = a + b;
        int operations = (a * b) / a + a % b - b;

        Console.WriteLine($"Ваш ввод, {input}!");
        Console.WriteLine($"Сумма чисел {a} и {b} равна {sum}.");
        Console.WriteLine(operations);

        // Пример использования цикла for
        Console.WriteLine("Вывод чисел от 1 до 10 (цикл for):");
        for (int i = 1; i <= 10; i++)
        {
            Console.WriteLine(i);
        }

        // Пример использования цикла while
        int counter = 1;
        Console.WriteLine("Вывод чисел от 1 до 10 (цикл while):");
        while (counter <= 10)
        {
            Console.WriteLine(counter);
            counter++;
        }

        // Пример использования цикла do-while
        counter = 1;
        Console.WriteLine("Вывод чисел от 1 до 10 (цикл do-while):");
        do
        {
            Console.WriteLine(counter);
            counter++;
        }
        while (counter <= 10);

        // Пример использования цикла foreach
        string[] arr = { "первый", "второй", "третий" };
        Console.WriteLine("Вывод элементов массива (цикл foreach):");
        foreach (string elem in arr)
```

```

        {
            Console.WriteLine(elem);
        }
    }
}

```

Листинг 2 – Файл example2.cs

```

class TestClass
{
    private int val;

    // Функция для вычисления суммы двух чисел
    public int Sum(int a, int b)
    {
        return a + b;
    }

    // Функция для умножения двух чисел
    public int Multiply(int a, int b)
    {
        return a * b;
    }

    public int GetVal()
    {
        return val;
    }
}

class Program
{
    static void Main()
    {
        // Создание объекта класса TestClass
        TestClass TestClass = new TestClass();

        // Вызов функций объекта TestClass
        int sumResult = TestClass.Sum(5, 3);
        int productResult = TestClass.Multiply(5, 3);
        int val = TestClass.GetVal();

        // Вывод результатов на консоль
        Console.WriteLine($"Сумма чисел: {sumResult}");
        Console.WriteLine($"Произведение чисел: {productResult}");
        Console.WriteLine(val);
    }
}

```

Листинг 3 – Файл examle3.cs

```

class QuickSort
{
    static void Swap(ref int a, ref int b)
    {
        int temp = a;
        a = b;
        b = temp;
    }
}

```

```

static int Partition(int[] array, int low, int high)
{
    int pivot = array[high];
    int i = low - 1;

    for (int j = low; j < high; j++)
    {
        if (array[j] <= pivot)
        {
            i++;
            Swap(ref array[i], ref array[j]);
        }
    }

    Swap(ref array[i + 1], ref array[high]);

    return i + 1;
}

static void Sort(int[] array, int low, int high)
{
    if (low < high)
    {
        int pi = Partition(array, low, high);

        Sort(array, low, pi - 1);
        Sort(array, pi + 1, high);
    }
}

public static void QuickSortFunc(int[] array)
{
    Sort(array, 0, array.Length - 1);
}

}

class Program
{
    static void Main()
    {
        int[] array = { 12, 7, 9, 4, 3, 2, 8, 6, 5, 10, 1, 11 };

        Console.WriteLine("Исходный массив:");
        PrintArray(array);

        QuickSort.QuickSortFunc(array);

        Console.WriteLine("\nОтсортированный массив:");
        PrintArray(array);

        Console.ReadLine();
    }

    static void PrintArray(int[] array)
    {
        foreach (int item in array)
        {
            Console.Write(item + " ");
        }
        Console.WriteLine();
    }
}

```