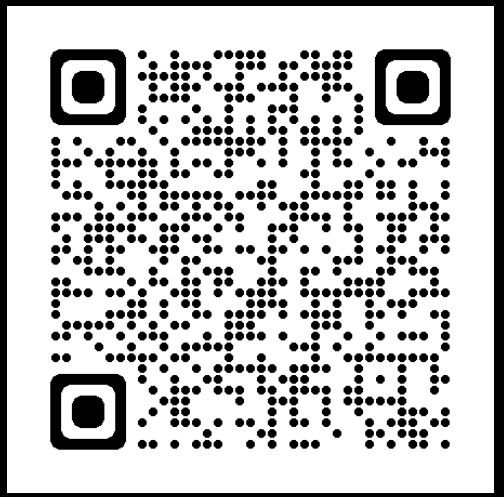
ЛЕКЦИЯ 14

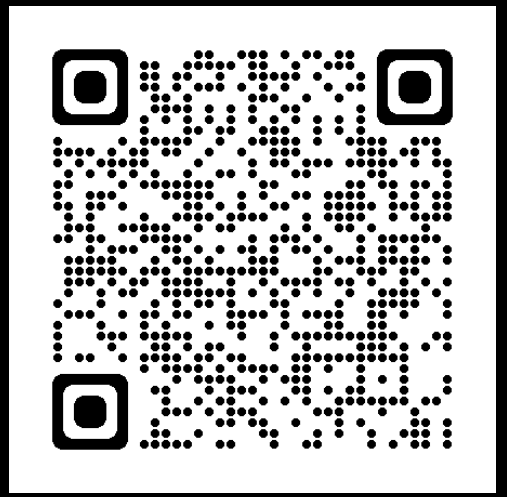
- 18.10.2023
- Рекуррентные последовательности
- Рекурсия

ВНИМАНИЕ ЭКЗАМЕНАЦИОННЫЙ ТЕСТ

- **Когда**: 26 октября 2023 10:00-11:20
- **Где**: ауд. R404, R504, R206, R306
- **Сколько времени**: 1 час, количество вопросов 40
- **Спецификация теста по ссылке:** 7557925981_2_3681600: Дополнительные задачи (hse.ru))
- **Тренировочные задания откроются 18.10.2023 в 17:40 (https://edu.hse.ru/mod/quiz/view.php?id=1008057)**

спецификация





тренировочные задания

ЦЕЛИ ЛЕКЦИИ

- Вспомнить понятие рекуррентных последовательностей
- Разобраться с понятием рекурсии и ее реализацией в C#
- Обсудить понятие стека вызовов и фреймов стека

ПОСЛЕДОВАТЕЛЬНОСТЬ

- Пусть заданы действительные числа $a_1, a_2, \dots, a_n, \dots$ Тогда скажем, что задана **числовая последовательность**

$$\{a_n\}_{n=1}^{\infty}, n \in \mathbb{N}$$

- **Числовую последовательность** также определяют, как функцию, заданную на множестве натуральных чисел

$$f: \mathbb{N} \to X, \text{где } X - \text{некоторое множество}$$

  - a_1 - первый член последовательности, a_2 - второй и т.д.
  - a_n - общий член последовательности

ЗАДАНИЕ ПОСЛЕДОВАТЕЛЬНОСТИ ФОРМУЛОЙ ОБЩЕГО ЧЛЕНА

- $1, 1, 1, \dots, 1, \dots$ → $x_n = 1$
- $1, -1, 1, -1, \dots$ → $x_n = (-1)^{n+1}$
- $0, 1, 0, \frac{1}{2}, 0, \frac{1}{3}, \dots$ → $x_n = \frac{1 + (-1)^n}{n}$

ПРИМЕР 1 (1)

Записать формулу общего члена последовательности:

$$1, \frac{3}{2}, \frac{1}{3}, \frac{3}{4}, \frac{1}{5}, \frac{3}{6}, \dots$$

- Рассуждения:
  - В знаменателе стоят члены натурального ряда, первый член последовательности можно переписать как: $\frac{1}{1}$
  - В числителе происходит чередование единиц и троек
  - Тройки в числителях отвечают чётным членам натурального ряда, а единицы нечётным
  - Перепишем каждый член последовательности, учитывая эти наблюдения

ПРИМЕР 1 (2)

$$n = 1: x_1 = \frac{1}{1} = \frac{2-1}{1} = \frac{2+(-1)^1}{1}$$

$$n = 2: x_2 = \frac{3}{2} = \frac{2+1}{2} = \frac{2+(-1)^2}{2}$$

$$n = 3: x_3 = \frac{1}{3} = \frac{2-1}{3} = \frac{2+(-1)^3}{3}$$

$$x_n = \frac{2+(-1)^n}{n}$$

ПРИМЕР 2 (1)

- Проверить принадлежность числа 6 последовательности:

$$x_n = \frac{n^2 + 11}{n + 1}$$

Правило принадлежности числа последовательности

$$\forall \boldsymbol{q} \in \{\boldsymbol{x_n}\}: \left(\exists \boldsymbol{n_0} \in \mathbb{N}: \boldsymbol{x_{n_0}} = \boldsymbol{q}\right)$$

$$x_{n_0} = \frac{n_0^2 + 11}{n_0 + 1} = 6$$

$$n_0^2 - 6n_0 + 5 = 0$$

$$\begin{cases} n_{0_1} = 1 \\ n_{0_2} = 5 \end{cases}$$

ПРИМЕР 2 (2)

- Исходя из формулы общего члена последовательности, проверим получившийся результат:

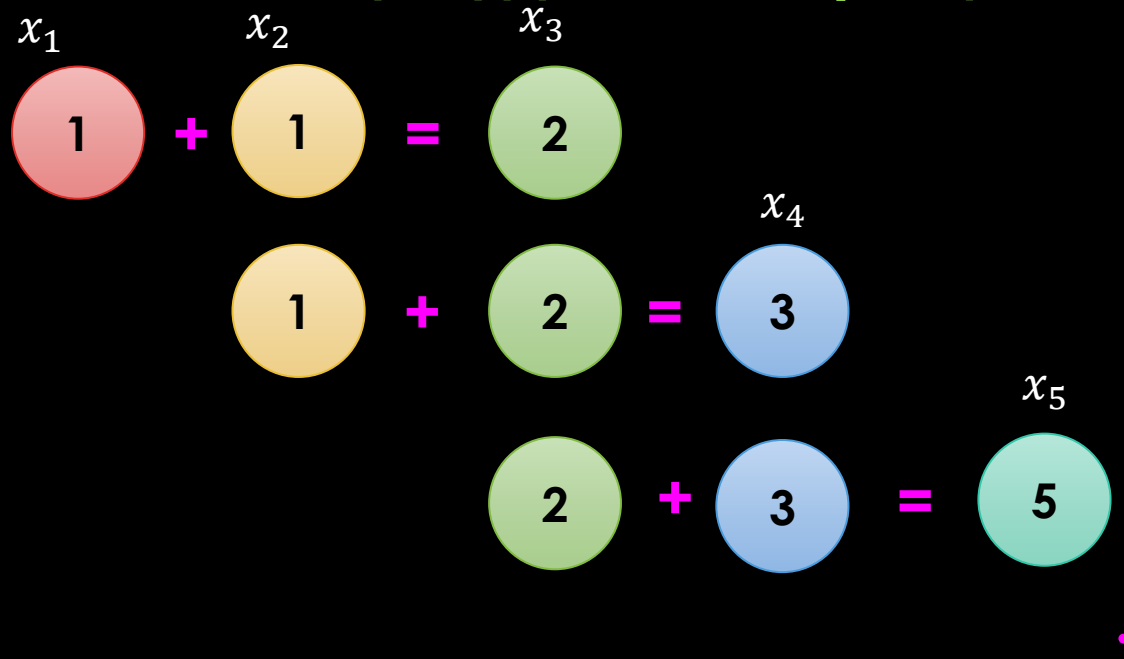
$$x_1 = \frac{1^2 + 11}{1 + 1} = \frac{12}{2} = 6$$

$$x_5 = \frac{5^2 + 11}{5 + 1} = \frac{36}{6} = 6$$

Число 6 является первым и пятым членом последовательности

РЕКУРРЕНТНАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ

Последовательности, в которых каждый член определяется как некоторая функция предыдущих, называются **рекуррентными (возвратными)**



Процесс последовательного определения элементов таких последовательностей называется **рекуррентным процессом**

РЕКУРРЕНТНАЯ ФОРМУЛА

- **Рекуррентная формула** - формула, сводящая вычисление *n*-го члена какой-либо последовательности (чаще всего числовой) к вычислению нескольких предыдущих её членов.

$$x_1 = 1$$
$$x_2 = 1$$
$$x_3 = x_1 + x_2 = 1 + 1 = 2$$
$$x_4 = x_2 + x_3 = 1 + 2 = 3$$
$$\vdots$$

Члены последовательности Фибоначчи

РЕКУРРЕНТНАЯ ФОРМУЛА К-ГО ПОРЯДКА

Рекуррентная формула k-го порядка, выражающая *n*-й член рекуррентной последовательности через k предыдущих:

$$a_n = f(n, a_{n-1}, a_{n-2}, \dots, a_{n-k})$$

Рекуррентной последовательностью порядка k является последовательность, для задания которой требуется задать первые её *k* членов

$$a_1, a_2, \dots, a_k$$

$$a_n = q_1 \cdot a_{n-1} + q_2 \cdot a_{n-2} + \dots + q_k \cdot a_{n-k}, n > k$$

Коэффициенты рекуррентной формулы: $q_1, q_2, \dots, q_k$

ПРИМЕРЫ (1)

- **Геометрическая прогрессия**
  - Общий член
  - Рекуррентная формула

$$a_n = b \cdot q^{n-1}$$

$$a_1 = b$$
$$a_n = q \cdot a_{n-1}, n > 1$$

- **Пример рекуррентной формулы первого порядка**

$$R_k = R_{k-1} + 3(k-1),$$
$$R_1 = 0$$

Начальные условия

- 0, 3, 9, 18, 30, …
- **Пример рекуррентной формулы второго порядка**

$$S_n = 34 \cdot S_{n-1} - S_{n-2} + 2,$$
$$S_0 = 0, S_1 = 1$$

- 0, 1, 36, 1225, 41616,…

ПРИМЕРЫ (2)

- Последовательность Фибоначчи представляется рекуррентной формулой второго порядка:

$$x_n = x_{n-2} + x_{n-1}$$

$x_1 = 1$
$x_2 = 1$
} начальные условия

ПРИМЕРЫ (3)

- Через **представление в виде функций** последовательность Фибоначчи может быть представлена так:

$$fib(n) = \begin{cases} 0, & n = 0; \\ 1, & n = 1; \\ fib(n-2) + fib(n-1), & n \geq 2 \end{cases}$$

ПРИМЕР. ВЫЧИСЛЕНИЕ ФАКТОРИАЛА НАТУРАЛЬНОГО ЧИСЛА

$$n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot (n-1) \cdot n$$
$$n! = (n-1)! \cdot n, 0! = 1$$

- Это – рекуррентная формула первого порядка
- Начальными являются условия $0! = 1$.

ДЕТАЛИ О ВЫЗОВАХ

Стек вызовов и стековые фреймы

Disclaimer: Все далее следующие слайды содержат информацию серьёзно упрощённую для подачи на первом курсе, многие концепции зависят от архитектур процессоров, соглашений операционных систем и др.!

НЕКОТОРЫЕ ТЕРМИНЫ

Стек вызовов [call stack, execution stack, control stack, run-time stack, machine stack, the stack] – LIFO-структура данных, хранящая в оперативной памяти или специальных регистрах ЦП информацию о всех активных подпрограммах компьютерной программы

Фрейм стека [stack frame] – область памяти, расположенная в стеке вызовов и ассоциированная с каждой функцией для хранения входных параметров, локальных переменных и иногда временных переменных

Соглашение о вызове [calling convention] определяет и формализует в терминах исполнимого кода целевого компьютера использование памяти, стека и регистров процессора, ответственность за выделение и освобождение памяти под аргументы, порядок аргументов в памяти, стеке или регистрах ЦП

ЧТО СОДЕРЖИТ СТЕКОВЫЙ ФРЕЙМ?

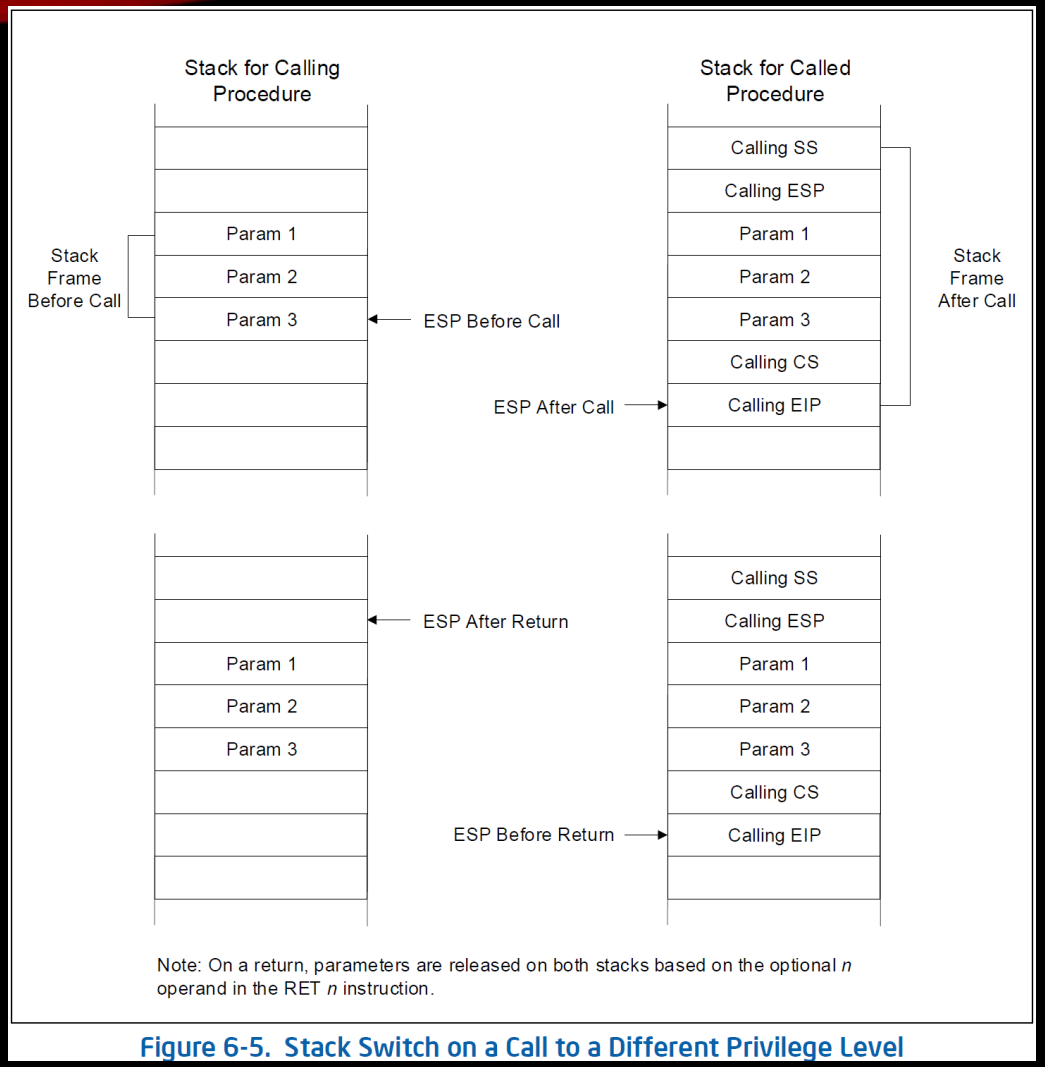


Figure 6-5. Stack Switch on a Call to a Different Privilege Level

- Локальные переменные и аргументы, передаваемые при вызове
- Адрес стекового фрейма вызывающей функции
- Адрес, куда передать управление после завершения работы

Служебные данные

ПРИМЕР

```
public class StackFramesDemo
{
    public static void MethodA(int par1, int par2)
    {
        Console.WriteLine($"Entered MethodA: {par1}, {par2}.");
        MethodB(11, 18); // Вызов MethodB.
        Console.WriteLine("Left MethodA.");
    }
    static void MethodB(int par1, int par2)
    {
        Console.WriteLine($"Entered MethodB: {par1}, {par2}.");
        Console.WriteLine("Left MethodB.");
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Entered Main.");
        StackFramesDemo.MethodA(15, 30);    // Вызов MethodA.
        Console.WriteLine("Left Main.");
    }
}
```

РАБОТА СО СТЕКОМ В C#

- StackFrame Class (https://learn.microsoft.com/en-us/dotnet/api/system.diagnostics.stacktrace)
- StackTrase.GetFrames Method StackTrace.GetFrames Method (System.Diagnostics) | Microsoft Learn

Пространство имен `System.Diagnostics`

РЕКУРСИЯ

Рекурсивные функции и вызовы

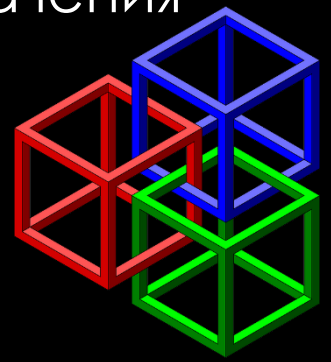
Стек при рекурсии

РЕКУРСИВНЫЕ ФУНКЦИИ

Рекурсивные функции [от *recursio*] – это такие функции, значения которых для данного аргумента вычисляются с помощью значений для предшествующих аргументов

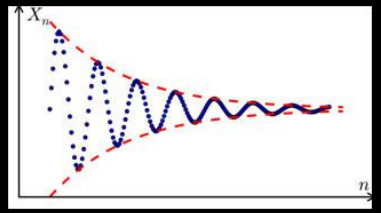
«**Рекурсия**» подразумевает способ задания функции, при котором значения определяемой функции для любых значений аргумента выражаются по известным правилам через значения этой функции для меньших значений аргументов

Функция называется **рекурсивной**, если существует эффективная процедура для ее вычисления, то есть существует определенный конечный набор известных математических операций, при которых мы получаем требуемые значения функции

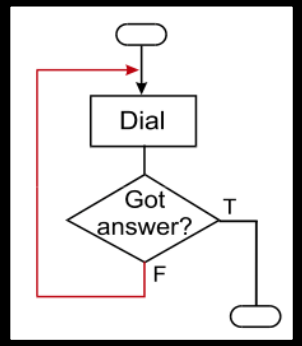


ТРАКТОВКА ТЕРМИНОВ

Рекуррентное соотношение задает некоторую функцию целочисленного аргумента в рамках понимания термина «функция» в классическом ***математическом анализе***



В контексте «алгоритм в виде процедурно реализованной рекурсивной функции *f(n)*» термин «функция» используется в рамках его понимания в ***программировании***



ПРЯМАЯ РЕКУРСИЯ

- F(n)
  - If (n=n0)
    - Then
      - F <- C
    - Else
      - F <- …F(m)…
  - Return (F)
- End.

1. Рекурсивная функция

2. Проверка останова рекурсии

3. Прямое вычисление

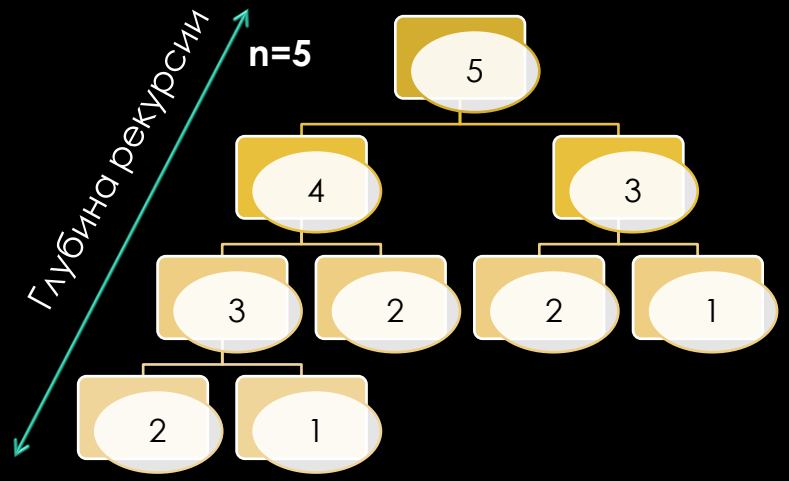
4. Рекурсивный вызов

Схема прямой рекурсии.

```
F(n)                                  (рекурсивная функция)
    If (n=n0)                         (проверка останова рекурсии)
        then
            F ← C                     (прямое вычисление F при n=n0)
        else
            F ← ...F(m)...            (рекурсивный вызов, m<n)
    Return (F)
End.
```

ДЕРЕВО РЕКУРСИИ

Дерево рекурсии – граф, имеющий древовидную структуру, вершины которого отражают рекурсивную функцию с данными аргументами, а дуги непосредственно рекурсивные вызовы




```
public static int Fib(int n)    {
   if (n == 1 || n == 2)
      return 1;
   else
     return Fib(n - 1) + Fib(n - 2);
}
```

РЕКУРСИЯ

```
class MyMath
{
    static long Factorial(long value)
    {
        if (value <= 1) return 1;
        else
            return value * Factorial(value - 1);
    }
}
```

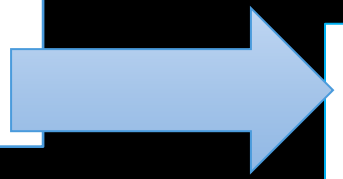
$$n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot (n-1) \cdot n$$
$$n! = (n-1)! \cdot n, 0! = 1$$

Проверка останова рекурсии

Рекурсивный вызов

ПРИМЕР РЕКУРСИИ В ЛОКАЛЬНОЙ ФУНКЦИИ

```
void Count(int inVal)
{
    Console.WriteLine($"Вход: {inVal}");
    if (inVal == 0) return; // Условие останова.
    Count(inVal - 1);   // Рекурсивный вызов.
    Console.WriteLine($"Выход: {inVal}");
}
Count(3); // Вызов функции.
```



```
Вход: 3
Вход: 2
Вход: 1
Вход: 0
Выход: 1
Выход: 2
Выход: 3
```

РЕКУРСИЯ И СТЕК

- При рекурсивных вызовах в C# стек заполняется аналогичным образом, как и при всех остальных
- Глубокая рекурсия может приводить к `StackOverflowException`

ХВОСТОВАЯ РЕКУРСИЯ

- **Хвостовая рекурсия** [tail recursion] - это специальный случай рекурсии, при котором рекурсивный вызов является последней операцией перед возвратом из функции и вызывающий код не оперирует результатом рекурсивного вызова

Хвостовая рекурсия может быть заменена на итерацию путём формальной и гарантированно корректной перестройки кода функции

Оптимизация хвостовой рекурсии путём преобразования её в плоскую итерацию реализована во многих оптимизирующих компиляторах

ТРАДИЦИОННАЯ И ХВОСТОВАЯ РЕКУРСИЯ

```
using System.Numerics;
static ulong Fact(uint k) { // Традиционная.
    if (k < 2) return 1;
    return k * Fact(k - 1);
}

static BigInteger FactTail(uint k, BigInteger product) { // Хвостовая.
    if (k < 2) return product;
    return FactTail(k - 1, k * product);
}
static BigInteger FactTailWrapper(uint k) {
    return FactTail(k, 1);
}
```

ТРАДИЦИОННАЯ И ХВОСТОВАЯ РЕКУРСИЯ

```
static void Main()
{
    Console.WriteLine(Fact(10));
    Console.WriteLine(Fact(40));
    Console.WriteLine(FactTailWrapper(10));
    Console.WriteLine(FactTailWrapper(40));
}
```

Результат выполнения:
3628800
18376134811363311616
3628800
18376134811363311616

Обратите внимание: хвостовая рекурсия поддерживается для IL, но не компилятором C#.
Подробнее про хвостовую рекурсию в Intermediate language и использование «трамплинов»:
https://thomaslevesque.com/2011/09/02/tail-recursion-in-c/

ИСПОЛЬЗОВАННАЯ ЛИТЕРАТУРА

- Материалы лекций по C# Подбельского В.В., Дударева В.А
- Richter, J. CLR via C#. Fourth edition
- Баррон д. Рекурсивные методы в программировании. – М.: «МИР», 1974
- Бердж В. Методы рекурсивного программирования. – М.: Машиностроение, 1983
- Головешкин В.А., Ульянов М.В. Теория рекурсии для программистов. – М.:Физматлит, 2006
- Организация памяти (https://habr.com/ru/companies/smart_soft/articles/185226/)
- Путешествуем по стеку. Часть 1. (https://habr.com/ru/companies/smart_soft/articles/234239/)
- P. Krzyzanowski Stack Frames. A really quick explanation of stack frames and frame pointers (https://people.cs.rutgers.edu/~pxk/419/notes/frames.html)
- Что происходит за кулисами C#: основы работы со стеком / Хабр (habr.com))
- Volgarev, P. Tail Recursion And Trampolining In C# (https://volgarev.me/2013/09/27/tail-recursion-and-trampolining-in-csharp.html)