ЛАБОРАТОРНАЯ РАБОТА № 25
РЕКУРСИВНЫЕ АЛГОРИТМЫ

Цель работы – приобретение навыков работы с рекурсивными методами.

КРАТКАЯ ТЕОРИЯ

Понятие рекурсии и простейшие примеры ее использования

В математике под рекурсией понимается способ организации вычислений, при котором функция вызывает сама себя с другим аргументом. Большинство современных языков высокого уровня поддерживают механизм рекурсивного вызова. Ввиду отсутствия в языке Java понятия функции рассматриваются рекурсивно вызываемые методы.

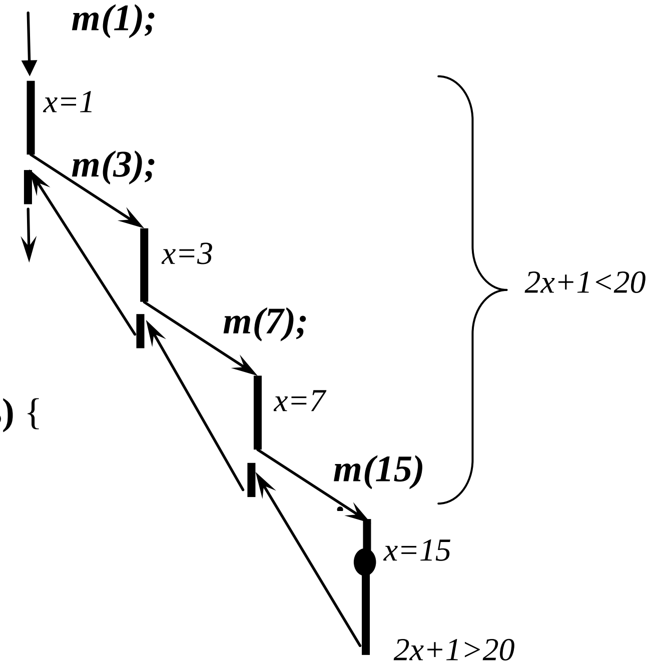
Таким образом, в языке **Java рекурсия** – это вызов метода из самого себя непосредственно (**простая рекурсия**) или через другие методы (**сложная**, или **косвенная рекурсия**). Пример сложной рекурсии: метод $m1$ вызывает метод $m2$, а метод $m2$ – метод $m1$.

Рассмотрим несколько примеров простой рекурсии.

Пример 2.1. Для заданного параметра ***x*** вывести последовательность значений элементов числового ряда в соответствии со следующими требованиями:

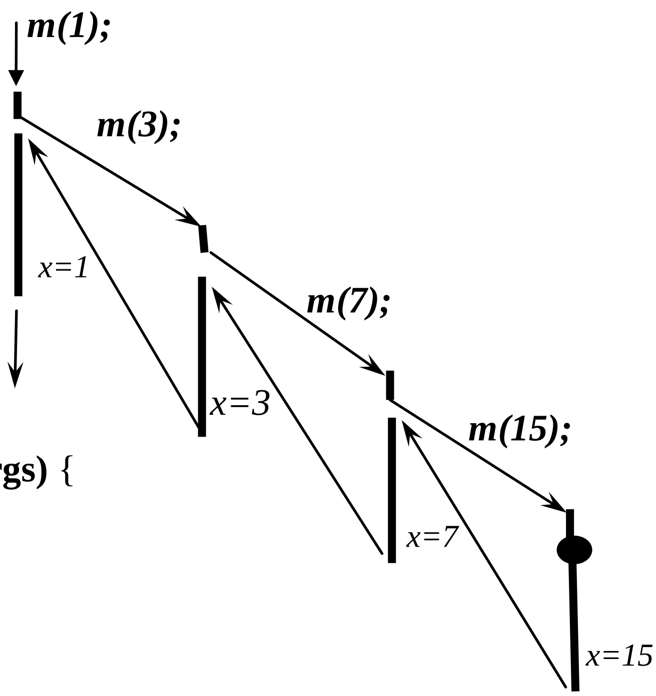
– очередной элемент $x = 2*x+1$ (новое значение вычисляется с использованием старого);
– $0 \leq x < 20$.

```
public class Rec1 {
   public static void m(int x) {
      System.out.println("x="+x);
      if ( (2*x+1) <20) {
         m(2*x+1);
      }
   }
public static void main(String[] args) {
      m(1);
   }
}
```



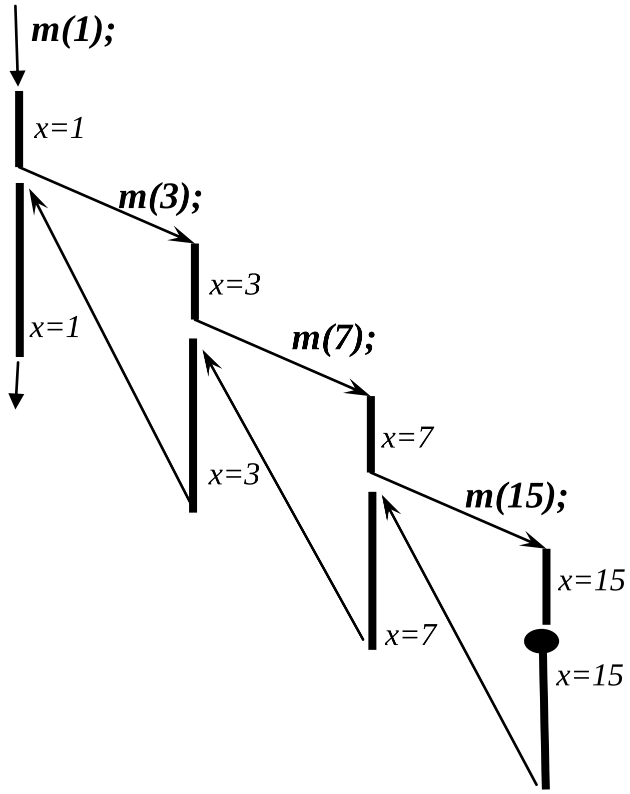
Пример 2.2. Вывести последовательность, представленную в предыдущем примере, в обратном порядке.

```
public class Rec2 {
   public static void m(int x) {
      if (  (2*x+1) <20) {
         m(2*x+1);
      }
      System.out.println("x="+x);
     }
   public static void main(String[] args) {
      m(1);
   }
}
```



Пример 2.3. Для вышеописанного задания сделать вывод параметра перед вхождением в рекурсивный вызов и после него.

```
public class Rec3 {
   private static int step=0;
   public static void m(int x) {
       space();
       System.out.println(""+x+"-> ");
       step++;
      if ((2*x+1) <20) {
        m(2*x+1);
      }
       step - -;
       space();
      System.out.println(""+x+ " <-");
   }
   public static void space() {
      for (int i = 0; i < step; i++) {
           System.out.print("  ");
      }
   }
   public static void main(String[] args) {
     m(1);
   }
}
```



Количество вложенных вызовов методов называется **глубиной рекурсии**.

Реализация рекурсивных вызовов опирается на механизм стека вызовов. Адрес возврата и локальные переменные метода записываются в стек, благодаря чему каждый следующий рекурсивный вызов этого метода пользуется своим набором локальных переменных и за счёт этого работает корректно.

На каждый рекурсивный вызов требуется некоторое количество оперативной памяти компьютера, и при чрезмерно большой глубине рекурсии может наступить переполнение стека. Будет сгенерирована исключительная ситуация **StackOverflowError** (переполнение стека).

Вследствие этого рекомендуется избегать рекурсивных программ, которые приводят к слишком большой глубине рекурсии.

Также следует отметить, что рекурсию можно заменить циклом.

Далее приведем наиболее часто используемые в учебных материалах примеры демонстрации работы рекурсии – вычисление факториала и чисел Фибоначчи.

Пример 2.4. Вычислить факториал числа *n* с использованием рекурсии.

Факториал числа *n* (обозначается *n!*) – произведение всех натуральных чисел от ***1*** до ***n*** включительно: *n!=1*2*…*n*. Пример 5!=1*2*3*4*5 =4!*5. Можно записать *n!=(n-1)!*n*.

```
public static  int fact(int n){
    int result;
    if (n==1)
       return 1;
    else{
       result=fact(n-1)*n;
       return result;
    }
}
```

Пример 2.5. Вывести число Фибоначчи, заданное его номером в последовательности.

Последовательность Фибоначчи формируется так: нулевой член последовательности равен нулю, первый – единице, а каждый следующий – сумме двух предыдущих.

№ числа	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*…*	*20*
число	*0*	*1*	*1*	*2*	*3*	*5*	*8*	*13*	*21*	*…*	*6765*

Графическое представление порождаемой данным алгоритмом цепочки рекурсивных вызовов называется деревом рекурсивных вызовов. Для рассматриваемого алгоритма оно показано на рис. 2.1.

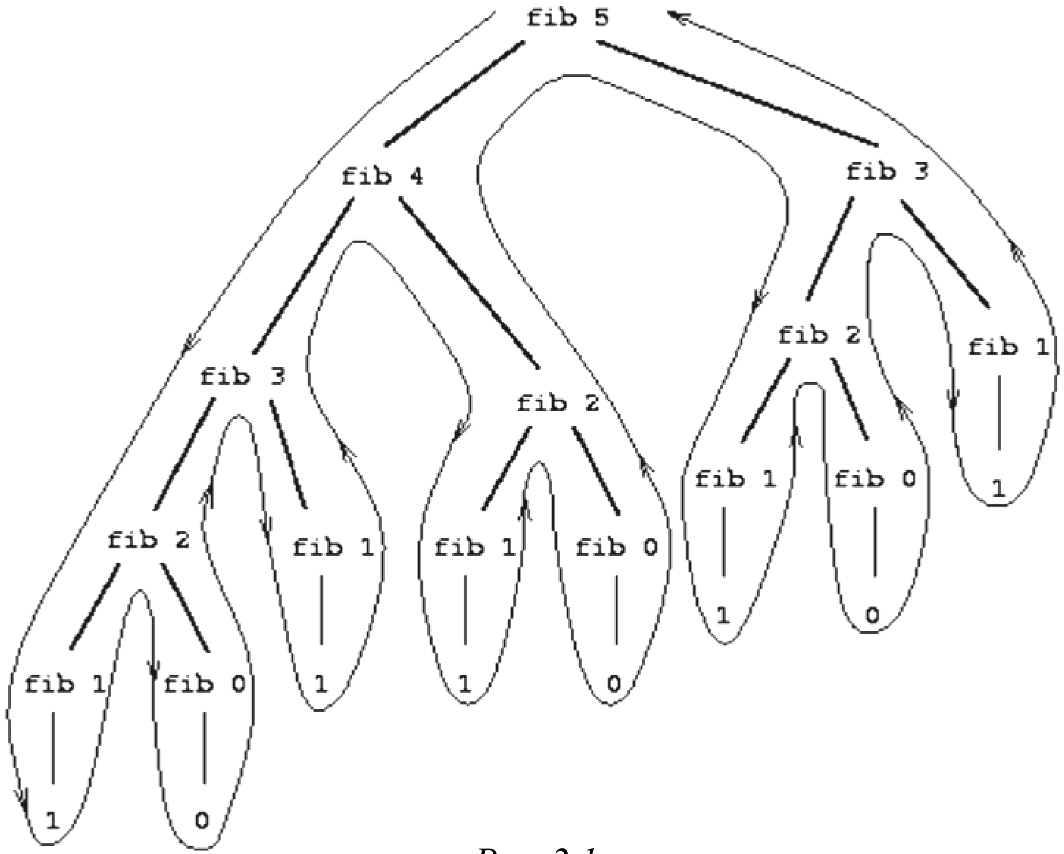


Рис. 2.1

```
public static int  f(int n){
   if (n==0){
       return 0;
   }else
     if (n==1){
         return 1;
     } else {
         return f(n-2)+f(n-1);
   }
}
```

Трудоемкость рекурсивных реализаций алгоритмов зависит как от количества операций, выполняемых при одном вызове функции, так и от количества таких вызовов.

Более детальное рассмотрение рекурсий при расчете их трудоемкости приводит к необходимости учета затрат как на организацию вызова функции и передачи параметров, так и на возврат вычисленных значений и передачу управления в точку вызова.

Можно заметить, что некоторая ветвь дерева рекурсивных вызовов обрывается при достижении такого значения передаваемого параметра, при котором функция может быть вычислена непосредственно. Таким образом, рекурсия эквивалентна конструкции цикла, в котором каждый проход есть выполнение рекурсивной функции с заданным параметром.

ЗАДАНИЯ

Задание 2.1. Создать приложения для демонстрации примеров $2.1 \div 2.5$. Для примера 2.5 дополнительно вывести последовательность обхода дерева рекурсивных вызовов. Отработать код с помощью отладчика.

Задание 2.2. Создать приложение с использованием рекурсии для перевода целого числа, введенного с клавиатуры, в двоичную систему счисления.

Задание 2.3. Создать приложение, позволяющее ввести и вывести одномерный массив целых чисел. Для ввода и вывода массива разработать рекурсивные методы вместо циклов for.

Задание 2.4. Ознакомиться с теорией и исследовать работу программы для нахождения корней нелинейного уравнения методом Ньютона и методом деления отрезка пополам, описанную в приложении 1 лабораторного практикума. Выполнить собственную реализацию методов с помощью рекурсии и проверить их на уравнениях, приведенных в табл. 2.1 согласно заданному варианту.

Таблица 2.1

Вариант 1 $x^3 – 4x^2 – 7x + 10 = 0$, корни: 1; -2; 5
Вариант 2 $x^3 + 3x^2 – 6x – 8 = 0$, корни: -4; -1; 2
Вариант 3 $x^3 – 5x^2 + 2x + 8 = 0$, корни: -1; 2; 4
Вариант 4 $x^3 – 8x^2 + 11x + 20 = 0$, корни: -1; 4; 5
Вариант 5 $x^3 + 2x^2 – 5x – 6 = 0$, корни: -3; -1; 2
Вариант 6 $x^3 – 4{,}5x^2 + 6{,}5x – 3 = 0$, корни: 1; 1,5; 2
Вариант 7 $x^3 – 4x^2 + x + 6 = 0$, корни: -1; 2; 3
Вариант 8 $x^3 – 1{,}5x^2 – x +1{,}5 = 0$, корни: -1; 1; 1,5
Вариант 9 $x^3 – 1{,}5x^2 – 2{,}5x + 3 = 0$, корни: -1,5; 1; 2
Вариант 10 $x^3 – 3{,}5x^2 + 0{,}5x + 5 = 0$, корни: -1; 2; 2,5
Вариант 11 $x^3 + 2{,}5x^2 – x – 2{,}5 = 0$, корни: -2,5; -1; 1
Вариант 12 $x^3 – 4x^2 – 20x + 48 = 0$, корни: -4; 2; 6

СОДЕРЖАНИЕ ОТЧЕТА

Готовится один отчет в печатном виде для заданий, указанных в лабораторной работе. Он должен содержать следующие разделы:

– титульный лист;

– задание согласно варианту;

– словесное описание алгоритма;

– текст программы **и скриншот результата**

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое рекурсия?
2. Что означает простая и косвенная рекурсия?
3. Как осуществить вывод информации в прямом и обратном порядке при вызове рекурсии?
4. Что такое глубина рекурсии?
5. Что такое дерево рекурсивных вызовов?
6. Каковы достоинства и недостатки использования рекурсии?