Федеральное государственное автономное образовательное учреждение
высшего образования
Национальный исследовательский университет
«Высшая школа экономики»

Факультет социально-экономических и компьютерных наук

Лабораторная работа №2

по дисциплине «Теоретические основы информатики»

Раздел 3 – Основы алгоритмизации

Выполнил: студент Яцишин Л.С., уч. группа ПСАПР-25-2

Пермь, 2025

Содержание

1 Задание 1

Построение из базовых функций

Имеется множество вычислимых функций (базовый набор). Используя операции суперпозиции σ, примитивной рекурсии ρ и минимизации μ, постройте функции и покажите, как выполняются вычисления для заданных значений:

1. Add(x, y) = x + y (покажите порядок рекурсивного вычисления для x = 4, y = 2)
2. Mult(x, y) = x · y (покажите порядок рекурсивного вычисления для x = 4, y = 2)
3. Power(x, y) = x^y (покажите порядок рекурсивного вычисления для x = 4, y = 2)

Покажите порядок вывода формул и примеры вычислений построенных рекурсивных функций для заданных значений. При построении функций можно использовать полученные ранее функции, расширяя ими базовый набор. В примерах вычислений покажите все вызовы функций при прямом ходе рекурсии до функций приведённого выше базового набора.

Базовые функции: $Z(x) = 0$, $\mathrm{Inc}(x) = x + 1$, $I_m^n(x_1, \ldots, x_n) = x_m$.

Сложение.

$$\begin{cases} Add(x, 0) = I_1^1(x), \\ Add(x, y + 1) = \mathrm{Inc}\big(Add(x, y)\big). \end{cases}$$

Умножение.

$$\begin{cases} Mult(x, 0) = Z(x), \\ Mult(x, y + 1) = Add\big(Mult(x, y),\ x\big). \end{cases}$$

Возведение в степень.

$$\begin{cases} Power(x, 0) = 1, \\ Power(x, y + 1) = Mult\big(Power(x, y),\ x\big). \end{cases}$$

Операция суперпозиции σ получает набор из $n + 1$ операндов – функций $f_0, f_1, \ldots, f_n$ и производит результат – функцию

$$f = \sigma(f_0, f_1, \ldots, f_n).$$

Если $f_1, \ldots, f_n$ зависят от одного и того же набора аргументов $x_1, \ldots, x_k$, тогда формула для вычисления значений функции f имеет вид

$$f(x_1, \ldots, x_k) = f_0\big(f_1(x_1, \ldots, x_k), \ldots, f_n(x_1, \ldots, x_k)\big).$$

Операция примитивной рекурсии ρ имеет два операнда:

$$f = \rho(g, h).$$

Если g зависит от n аргументов, а h зависит от $n + 2$ аргументов, тогда функция-результат f определяется уравнениями примитивной рекурсии:

$$\begin{cases} f(x_1, \ldots, x_n, 0) = g(x_1, \ldots, x_n), \\ f(x_1, \ldots, x_n, y + 1) = h\big(x_1, \ldots, x_n, y, f(x_1, \ldots, x_n, y)\big). \end{cases}$$

Сложение: формальное определение.

$$g_{\text{Add}}(x) = I_1^1(x),$$

$$h_{\text{Add}}(x, y, z) = \text{Inc}(z) = \sigma\big(\text{Inc}, I_3^3\big)(x, y, z).$$

$$Add = \rho\big(g_{\text{Add}}, h_{\text{Add}}\big) = \rho\big(I_1^1, \sigma(\text{Inc}, I_3^3)\big).$$

Умножение: формальное определение.

$$g_{\text{Mult}}(x) = Z(x) = \sigma\big(Z, I_1^1\big)(x),$$

$$\begin{aligned} h_{\text{Mult}}(x, y, z) &= Add(z, x) \\ &= Add\big(I_3^3(x, y, z), I_1^3(x, y, z)\big) \\ &= \sigma\big(Add, I_3^3, I_1^3\big)(x, y, z). \end{aligned}$$

$$Mult = \rho\big(g_{\text{Mult}}, h_{\text{Mult}}\big) = \rho\big(\sigma(Z, I_1^1), \sigma(Add, I_3^3, I_1^3)\big).$$

Возведение в степень: формальное определение. Сначала зададим константу 1 как функцию одной переменной:

$$One(x) = \text{Inc}\big(I_1^1(Z(x))\big) = \sigma\big(\text{Inc}, I_1^1, Z\big)(x).$$

Тогда

$$g_{\text{Power}}(x) = One(x) = \sigma\big(\text{Inc}, I_1^1, Z\big)(x),$$

$$\begin{aligned} h_{\text{Power}}(x, y, z) &= Mult(z, x) \\ &= Mult\big(I_3^3(x, y, z), I_1^3(x, y, z)\big) \\ &= \sigma\big(Mult, I_3^3, I_1^3\big)(x, y, z). \end{aligned}$$

$$Power = \rho\big(g_{\text{Power}}, h_{\text{Power}}\big) = \rho\Big(\sigma(\text{Inc}, I_1^1, Z), \sigma(Mult, I_3^3, I_1^3)\Big).$$

Add(4,2).

$$\begin{aligned} Add(4, 2) &= \text{Inc}\big(Add(4, 1)\big) \\ &= \text{Inc}\big(\text{Inc}(Add(4, 0))\big) \\ &= \text{Inc}\big(\text{Inc}(I_1^1(4))\big) \\ &= \text{Inc}\big(\text{Inc}(4)\big) \\ &= \text{Inc}(5) \\ &= 6. \end{aligned}$$

Mult(4,2).

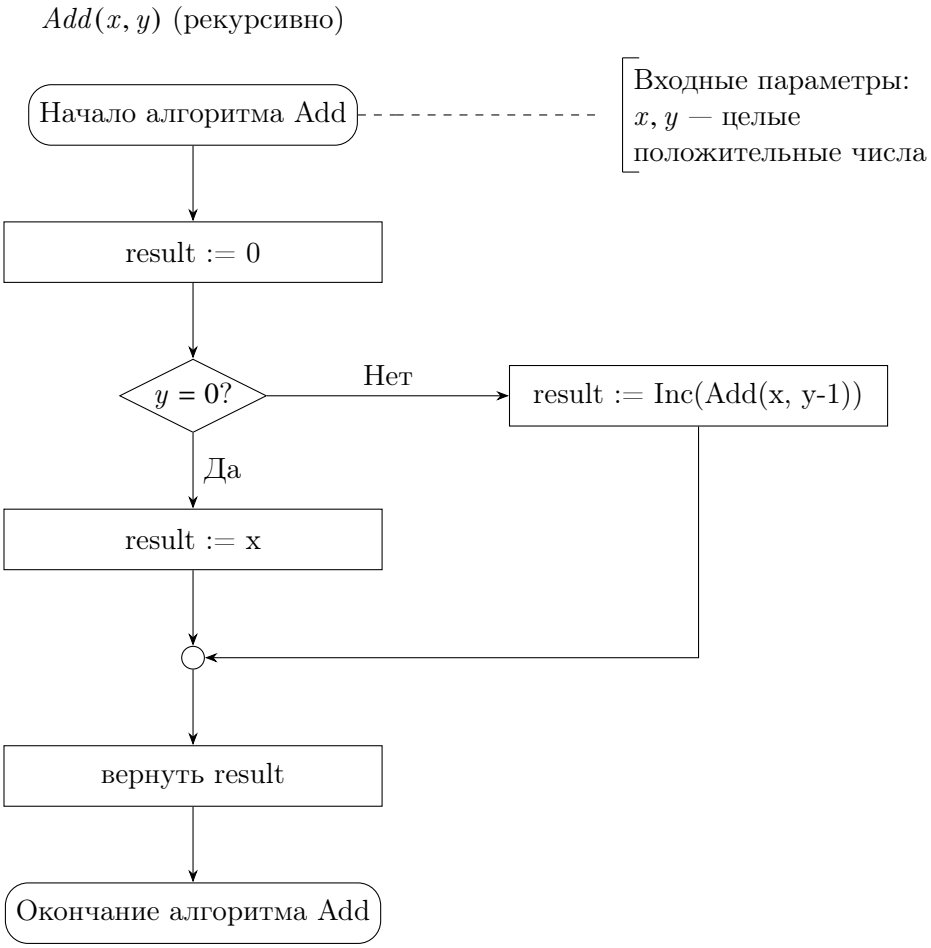
$$\begin{aligned} Mult(4, 2) &= Add(Mult(4, 1), 4) \\ &= Add(Add(Mult(4, 0), 4), 4) \\ &= Add(Add(Z(4), 4), 4) \\ &= Add(Add(0, 4), 4) \\ &= Add\big(\text{Inc}(\text{Inc}(\text{Inc}(\text{Inc}(0)))), 4\big) \\ &= \text{Inc}(\text{Inc}(\text{Inc}(\text{Inc}(I_1^1(\text{Inc}(\text{Inc}(\text{Inc}(\text{Inc}(0))))))))) \\ &= 8. \end{aligned}$$

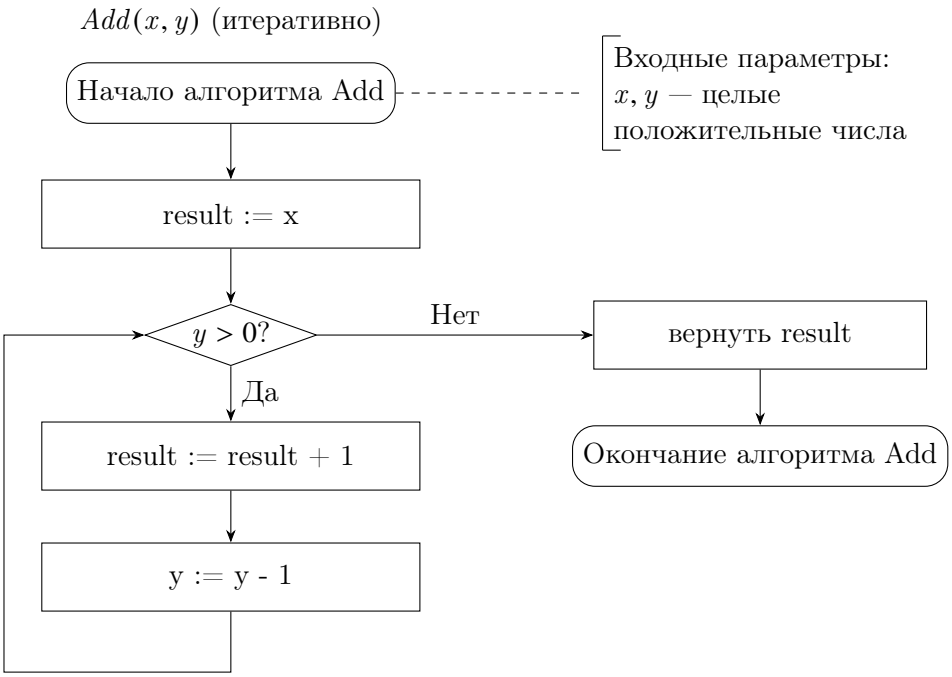
Power(4,2).

$$
\begin{aligned}
Power(4,2) &= Mult(Power(4,1),4)\\
&= Mult(Mult(Power(4,0),4),4)\\
&= Mult(Mult(\text{Inc}(Z(4)),4),4)\\
&= Mult(Mult(\text{Inc}(0),4),4)\\
&= Mult(Mult(1,4),4)\\
&= Mult(4,4)\\
&= Add(Mult(4,3),4)\\
&= Add(Add(Mult(4,2),4),4)\\
&= Add(Add(Add(Mult(4,1),4),4),4)\\
&= Add(Add(Add(Add(Mult(4,0),4),4),4),4)\\
&= Add(Add(Add(Add(Z(4),4),4),4),4)\\
&= Add(Add(Add(Add(0,4),4),4),4)\\
&= \text{Inc(Inc(Inc(Inc(Inc(Inc(Inc(Inc(Inc(Inc(Inc(Inc(Inc(Inc(Inc(0))))))))))))))))}\\
&= 16.
\end{aligned}
$$

Блок-схемы и структураграммы

Опишите рекурсивные и итерационные алгоритмы вычисления функций Add(x, y), Mult(x, y), Power(x, y) в виде блок-схем и структурограмм.





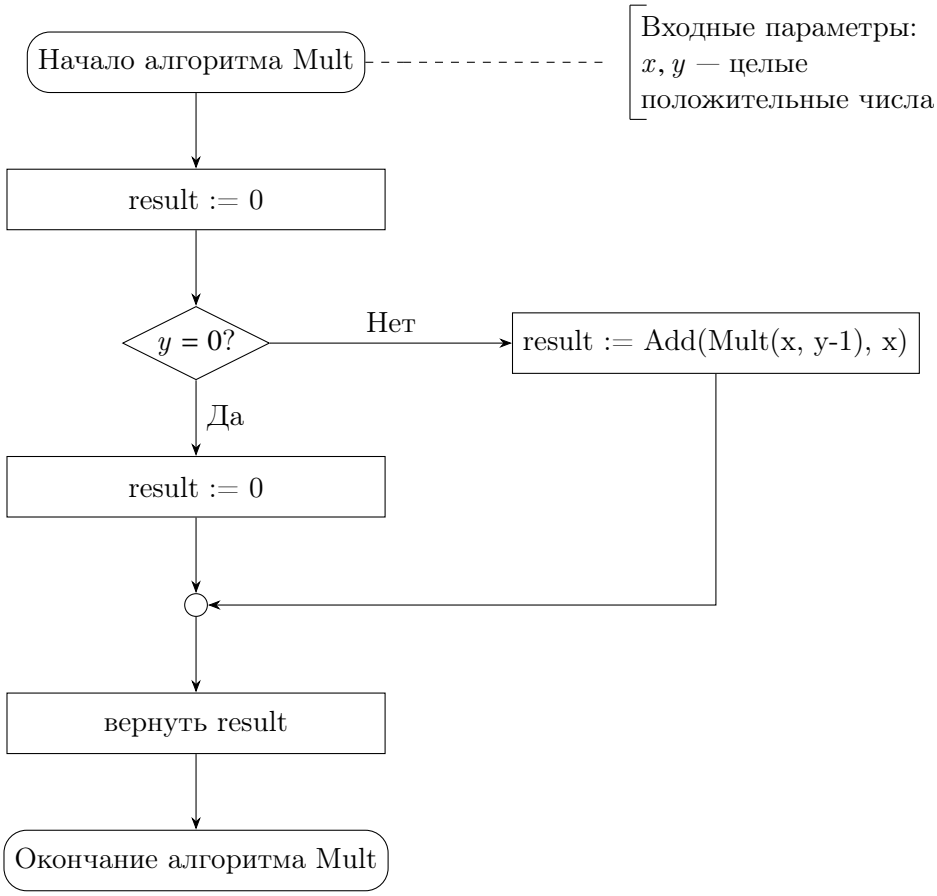
Add(x,y), итеративная реализация

Инициализация
result = x
Выполнить y раз: result := Inc(result)
i = 0; i < y; i = i + 1
result = Inc(result)
Вернуть результат
result

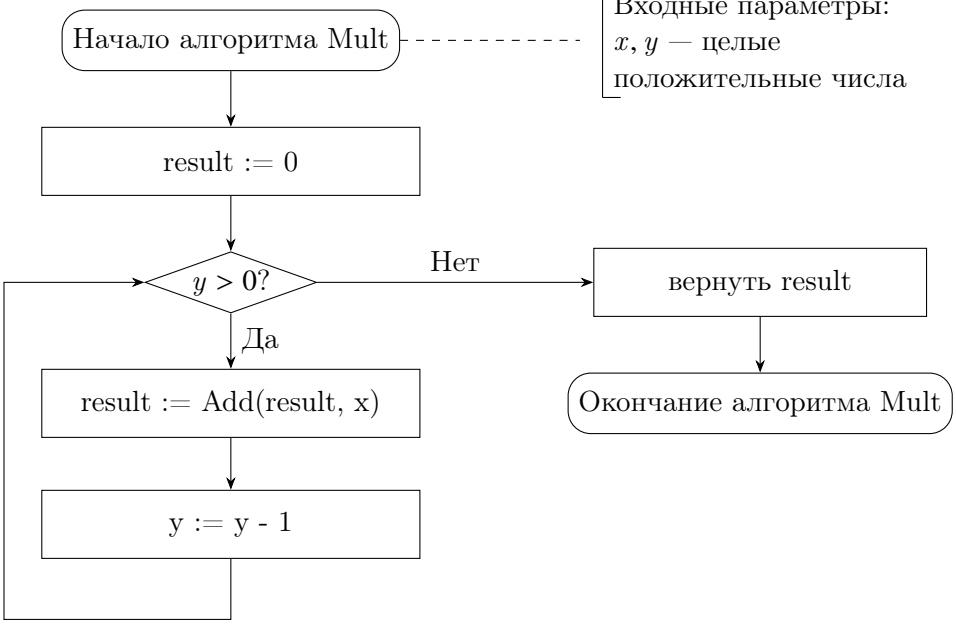
Add(x,y), рекурсивное определение

Базовый случай: y == 0	
y == 0	
true	*false*
I^1_1(x)	*Рекурсивный случай: y > 0*
	Inc(Add(x, y - 1))

Mult(x, y) (рекурсивно)



Mult(x, y) (итеративно)



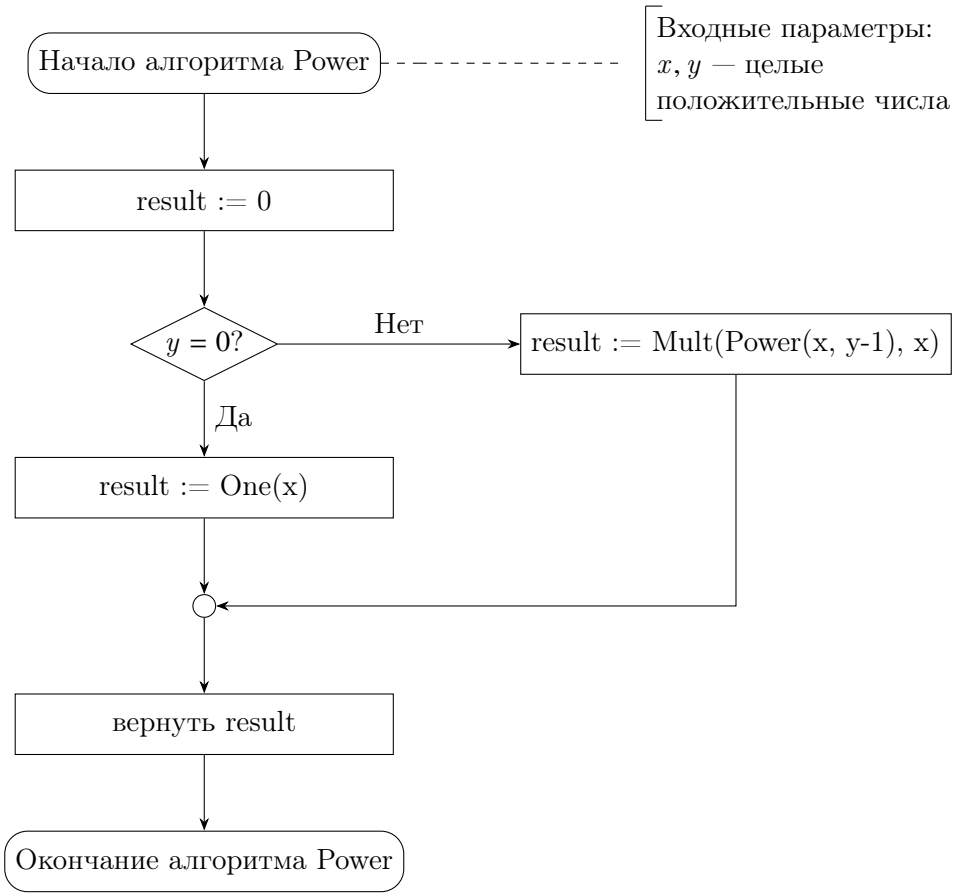
Mult(x,y), итеративная реализация

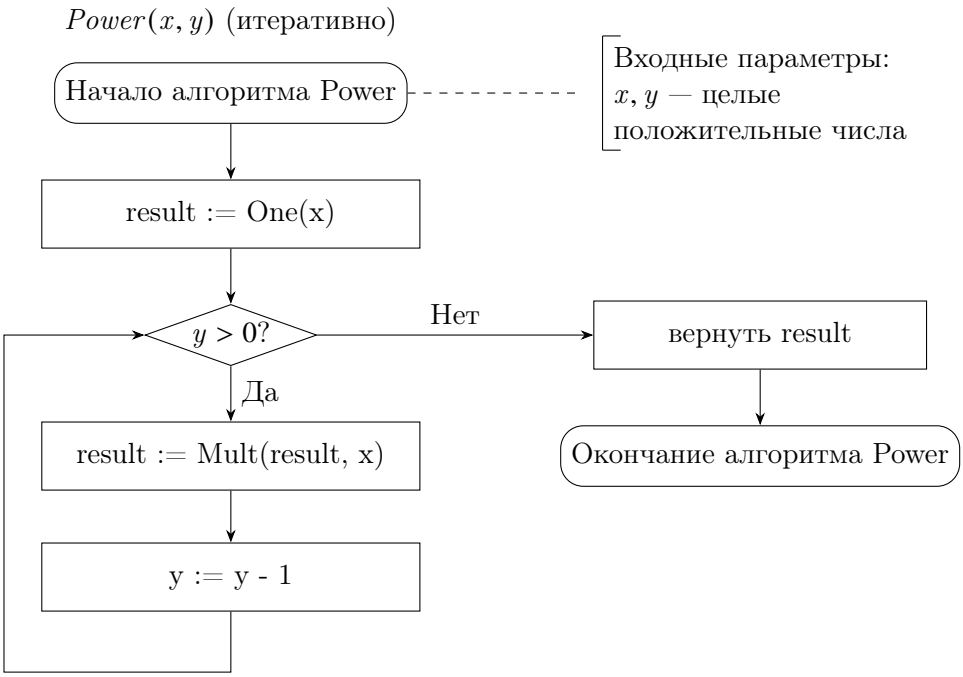
Инициализация (результат = 0)
result = Z(x)
Прибавлять x ровно y раз
i = 0; i < y; i = i + 1
result = Add(result, x)
Вернуть результат
result

Mult(x,y), рекурсивная реализация

Базовый случай: y == 0	
y == 0	
true	*false*
Z(x)	*Рекурсивный случай: y > 0*
	Add(Mult(x, y - 1), x)

$Power(x, y)$ (рекурсивно)





Power(x,y), итеративная реализация

Инициализация (результат = 1)
result = One(x)
Умножать result на x ровно y раз
i = 0; i < y; i = i + 1
result = Mult(result, x)
Вернуть результат
result

Power(x,y), рекурсивная реализация

Базовый случай: y == 0	
y == 0	
true	*false*
One(x)	*Рекурсивный случай*
	Mult(Power(x, y - 1), x)

Алгоритмы на C++. Трассировка

Разработайте на языке C++ функции, реализующие a) итерационные и b) рекурсивные алгоритмы вычисления приведённых выше функций (Inc(x), Add(x, y), Mult(x, y), Power(x, y)). Выполните вычисления пошагово с помощью средств отладки, проверяя приведённые вычисления, – постройте трассировочные таблицы или деревья вызовов для выполнения каждой функции (используйте скриншоты) и сравните сложность рекурсивного и итерационного алгоритмов.

```
1  int Inc(int x) {
2      return x + 1;
3  }
4
5  int Z(int x) {
6      return 0;
7  }
8
9  int AddRec(int x, int y) {
10     if (y == 0) {
```

```
        return x;
    } else {
        return Inc(AddRec(x, y - 1));
    }
}

int AddIter(int x, int y) {
    int result = x;
    int i = 0;
    while (i < y) {
        result = Inc(result);
        i = i + 1;
    }
    return result;
}

int MultRec(int x, int y) {
    if (y == 0) {
        return Z(x);
    } else {
        int prev = MultRec(x, y - 1);
        return AddRec(prev, x);
    }
}

int MultIter(int x, int y) {
    int result = Z(x);
    int i = 0;
    while (i < y) {
        result = AddIter(result, x);
        i = i + 1;
    }
    return result;
}

int PowerRec(int x, int y) {
    if (y == 0) {
        return 1;
    } else {
        int prev = PowerRec(x, y - 1);
        return MultRec(prev, x);
    }
}

int PowerIter(int x, int y) {
    int result = 1;
    int i = 0;
    while (i < y) {
        result = MultIter(result, x);
        i = i + 1;
    }
    return result;
}

```

Операция	Значения переменных	Комментарий	Номер шага
AddRec(x, y)	$x = 4,\ y = 2$	Вызов функции с аргументами $x = 4,\ y = 2$	0
Проверка $y = 0$	$y = 2$	Условие ложно, выбирается рекурсивная ветка	1
$AddRec(4, 2) = Inc(AddRec(4, 1))$	$x = 4,\ y = 2$	Рекурсивный вызов $AddRec(4, 1)$	2
AddRec(4,1)	$x = 4,\ y = 1$	Вход во внутренний рекурсивный вызов	3
Проверка $y = 0$	$y = 1$	Условие ложно, рекурсия продолжается	4
$AddRec(4, 1) = Inc(AddRec(4, 0))$	$x = 4,\ y = 1$	Рекурсивный вызов $AddRec(4, 0)$	5
AddRec(4,0)	$x = 4,\ y = 0$	Вход в базовый случай рекурсии	6
Проверка $y = 0$	$y = 0$	Условие истинно, возвращаем $x = 4$	7
$Inc(4)$	$result = 5$	Возврат из $AddRec(4, 1)$ со значением 5	8
$Inc(5)$	$result = 6$	Возврат из $AddRec(4, 2)$, итог 6	9

Таблица 1: Трассировка рекурсивной функции AddRec(4,2)

Операция	Значения переменных	Комментарий	Номер шага
AddIter(x, y)	$x = 4,\ y = 2$	Вызов функции с аргументами $x = 4,\ y = 2$	0
$result = x;\ i = 0$	$result = 4,\ i = 0$	Инициализация перед циклом	1
Начало цикла while $(i < y)$	$i = 0,\ y = 2$	Подготовка к выполнению итераций	2
$result = Inc(result);$	до: $result = 4,\ i = 0$	Первая итерация цикла	3
$i = i + 1;$	после: $result = 5,\ i = 1$	Завершение первой итерации	3
Проверка $i < y$	$i = 1,\ y = 2$	Условие истинно, вторая итерация	4
$result = Inc(result);$	до: $result = 5,\ i = 1$	Вторая итерация цикла	5
$i = i + 1;$	после: $result = 6,\ i = 2$	Завершение второй итерации	5
Проверка $i < y$	$i = 2,\ y = 2$	Условие ложно, выход из цикла	6
return result	$result = 6$	Возвращаем сумму 4+2 = 6	7

Таблица 2: Трассировка итерационной функции AddIter(4,2)

Операция	Значения переменных	Комментарий	Номер шага
MultRec(x, y)	$x = 4,\ y = 2$	Вызов функции с аргументами $x = 4,\ y = 2$	0
Проверка $y = 0$	$y = 2$	Условие ложно, выполняется рекурсивная ветка	1
$prev = MultRec(4, 1)$	$x = 4,\ y = 2$	Рекурсивный вызов с $y = 1$	2
MultRec(4,1)	$x = 4,\ y = 1$	Вход во внутренний рекурсивный вызов	3
Проверка $y = 0$	$y = 1$	Условие ложно, рекурсия продолжается	4
$prev = MultRec(4, 0)$	$x = 4,\ y = 1$	Рекурсивный вызов с $y = 0$	5
MultRec(4,0)	$x = 4,\ y = 0$	Вход в базовый случай рекурсии	6
Проверка $y = 0$	$y = 0$	Условие истинно, вычисляем $Z(4) = 0$	7
return 0	$prev = 0$	Возврат из $MultRec(4, 0)$	8
$AddRec(prev, x)$	$prev = 0,\ x = 4$	Вызываем $AddRec(0, 4)$, получаем 4	9
return 4	$result = 4$	Возврат из $MultRec(4, 1)$	10
$AddRec(prev, x)$	$prev = 4,\ x = 4$	Вызываем $AddRec(4, 4)$, получаем 8	11
return 8	$result = 8$	Возврат из $MultRec(4, 2)$, итог 8	12

Таблица 3: Трассировка рекурсивной функции MultRec(4,2)

Операция	Значения переменных	Комментарий	Номер шага
MultIter(x, y)	$x = 4,\ y = 2$	Вызов функции с аргументами $x = 4$, $y = 2$	0
$result = Z(x)$;	$result = 0$	Обнуление результата: $Z(4) = 0$	1
$i = 0$	$i = 0$	Инициализация счётчика цикла	2
Начало цикла while $(i < y)$	$i = 0,\ y = 2$	Подготовка к первой итерации	3
$result = AddIter(result, x)$;	до: $result = 0,\ i = 0$	Первая итерация: вызов $AddIter(0, 4)$	4
$i = i + 1$;	после: $result = 4,\ i = 1$	Завершение первой итерации	4
Проверка $i < y$	$i = 1,\ y = 2$	Условие истинно, вторая итерация	5
$result = AddIter(result, x)$;	до: $result = 4,\ i = 1$	Вторая итерация: вызов $AddIter(4, 4)$	6
$i = i + 1$;	после: $result = 8,\ i = 2$	Завершение второй итерации	6
Проверка $i < y$	$i = 2,\ y = 2$	Условие ложно, выход из цикла	7
return result	$result = 8$	Возвращаем произведение $4 \cdot 2 = 8$	8

Таблица 4: Трассировка итерационной функции MultIter(4,2)

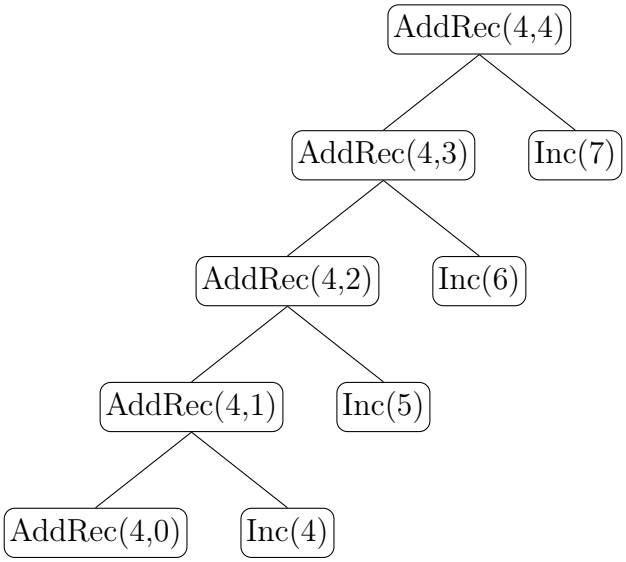
Операция	Значения переменных	Комментарий	Номер шага
PowerRec(x, y)	$x = 4,\ y = 2$	Вызов функции с аргументами $x = 4$, $y = 2$	0
Проверка $y = 0$	$y = 2$	Условие ложно, выбирается рекурсивная ветка	1
$prev = PowerRec(4, 1)$	$x = 4,\ y = 2$	Рекурсивный вызов с $y = 1$	2
PowerRec(4,1)	$x = 4,\ y = 1$	Вход во внутренний рекурсивный вызов	3
Проверка $y = 0$	$y = 1$	Условие ложно, рекурсия продолжается	4
$prev = PowerRec(4, 0)$	$x = 4,\ y = 1$	Рекурсивный вызов с $y = 0$	5
PowerRec(4,0)	$x = 4,\ y = 0$	Базовый случай, степень ноль	6
Проверка $y = 0$	$y = 0$	Условие истинно, возвращаем 1	7
return 1	$prev = 1$	Возврат из $PowerRec(4, 0)$	8
$MultRec(prev, x)$	$prev = 1,\ x = 4$	Вызываем $MultRec(1, 4)$, получаем 4	9
return 4	$prev = 4$	Возврат из $PowerRec(4, 1)$	10
$MultRec(prev, x)$	$prev = 4,\ x = 4$	Вызываем $MultRec(4, 4)$, получаем 16	11
return 16	$result = 16$	Возврат из $PowerRec(4, 2)$, итог 16	12

Таблица 5: Трассировка рекурсивной функции PowerRec(4,2)

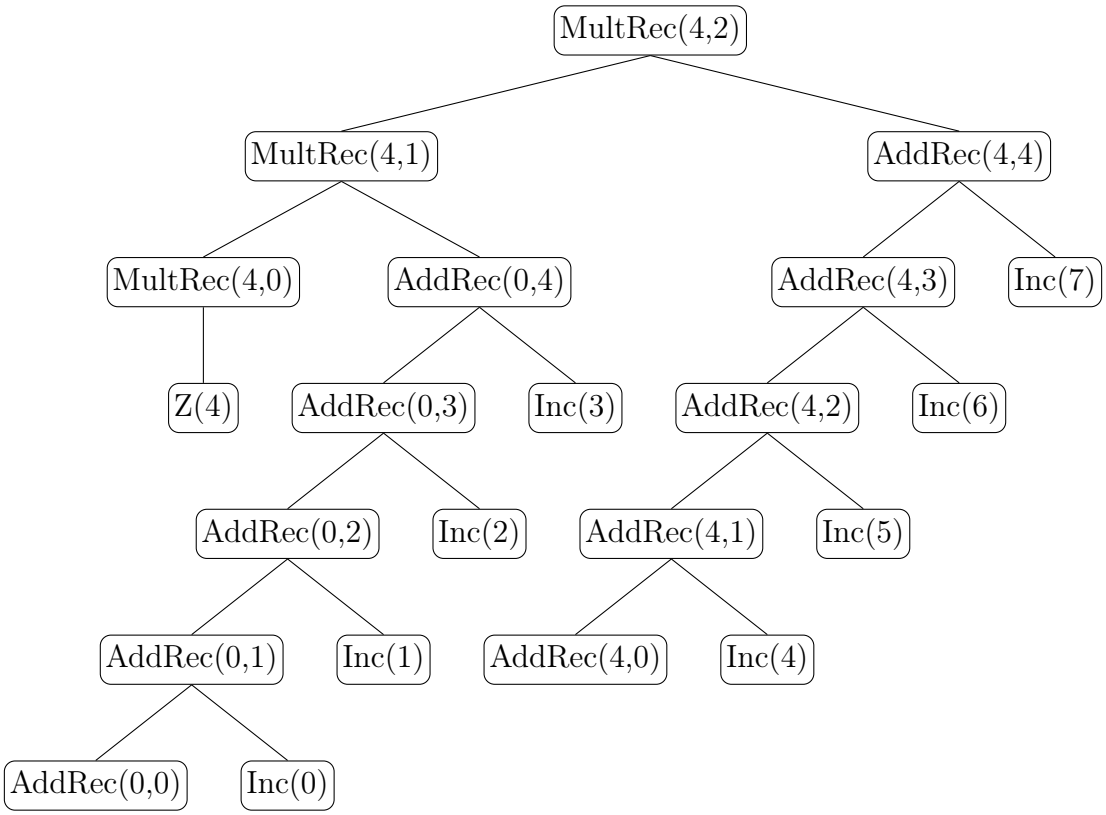
Операция	Значения переменных	Комментарий	Номер шага
PowerIter(x, y)	$x = 4,\ y = 2$	Вызов функции с аргументами $x = 4$, $y = 2$	0
$result = 1$;	$result = 1$	Инициализация результата (единица)	1
$i = 0$	$i = 0$	Инициализация счётчика цикла	2
Начало цикла while $(i < y)$	$i = 0,\ y = 2$	Подготовка к первой итерации	3
$result = MultIter(result, x)$;	до: $result = 1,\ i = 0$	Первая итерация: вызов $MultIter(1, 4)$	4
$i = i + 1$;	после: $result = 4,\ i = 1$	Завершение первой итерации	4
Проверка $i < y$	$i = 1,\ y = 2$	Условие истинно, вторая итерация	5
$result = MultIter(result, x)$;	до: $result = 4,\ i = 1$	Вторая итерация: вызов $MultIter(4, 4)$	6
$i = i + 1$;	после: $result = 16,\ i = 2$	Завершение второй итерации	6
Проверка $i < y$	$i = 2,\ y = 2$	Условие ложно, выход из цикла	7
return result	$result = 16$	Возвращаем $4^2 = 16$	8

Таблица 6: Трассировка итерационной функции PowerIter(4,2)

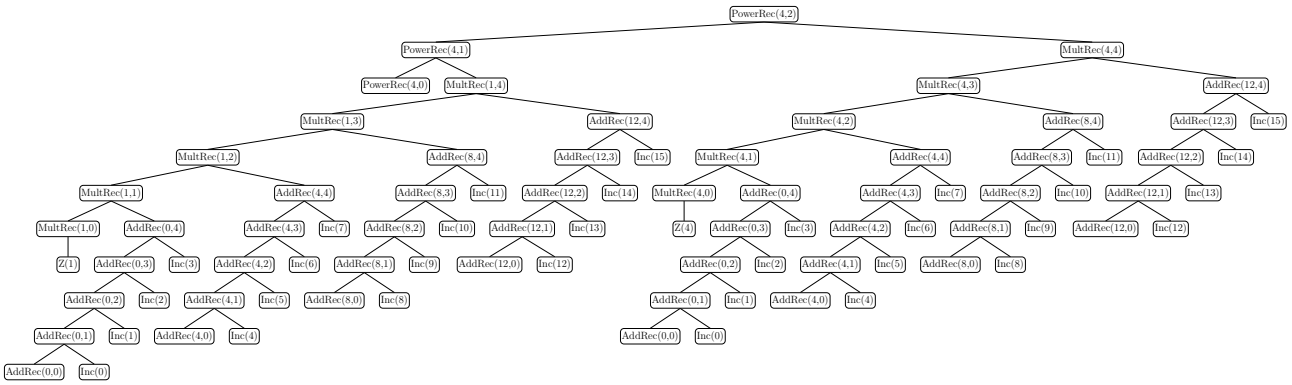
Дерево вызовов AddRec(4,4).



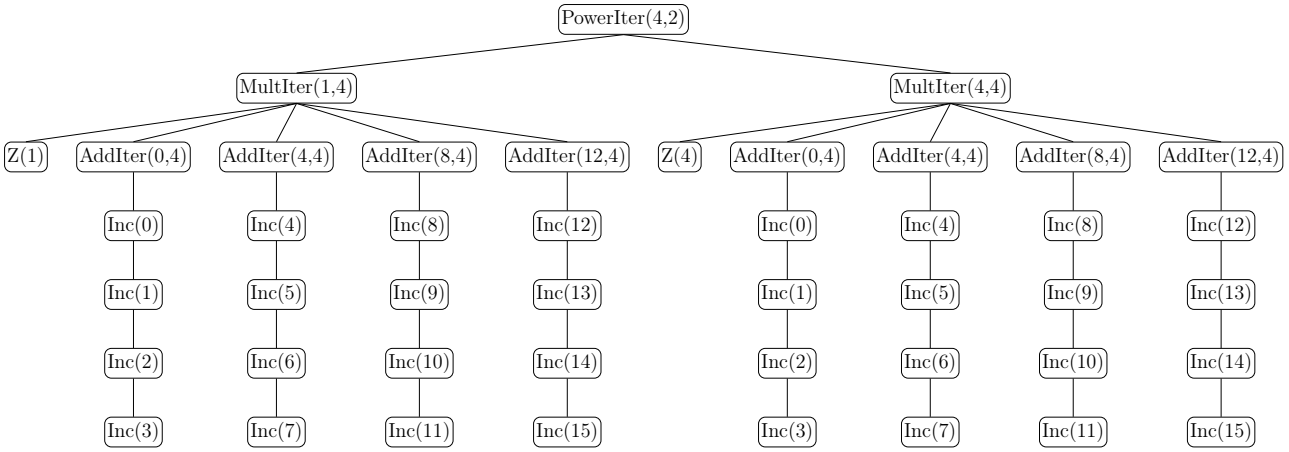
Дерево вызовов MultRec(4,2).



Дерево вызовов PowerRec(4,2).



Дерево вызовов PowerIter(4,2).



Краткая оценка сложности

Рассмотренные рекурсивные и итеративные варианты функций сложения, умножения и возведения в степень основаны на одной и той же идее: итеративный алгоритм по сути просто разворачивает ту же самую рекурсивную схему в цикл. Поэтому при одинаковых входных данных количество шагов в обоих вариантах растёт примерно одинаково, то есть их вычислительная сложность совпадает по порядку роста. Рекурсивная форма немного сложнее из-за вызовов функций.

Если бы итеративный и рекурсивный алгоритмы использовали разные подходы, тогда их сложность могла бы существенно отличаться.

2 Задание 2

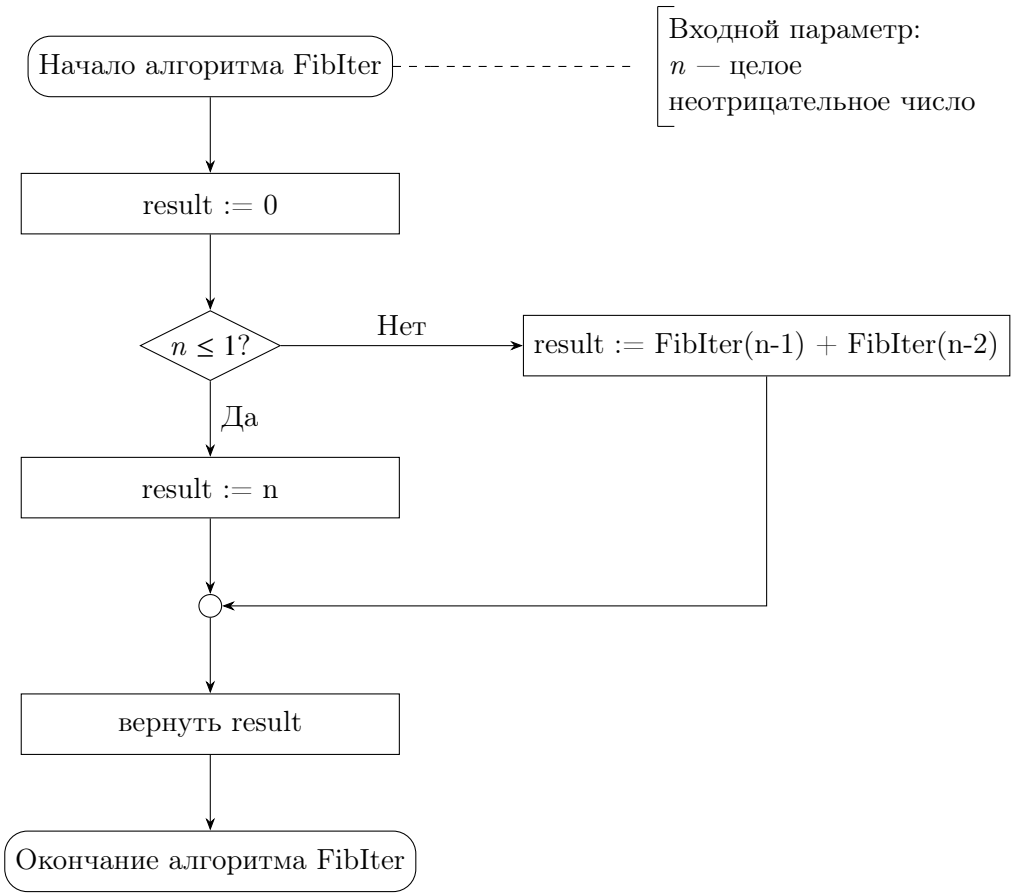
Алгоритм функции Фибоначчи на C++

Напишите а) рекурсивную функцию и б) функцию, реализующую итерационный алгоритм вычисления чисел Фибоначчи по заданному номеру, на языке C++

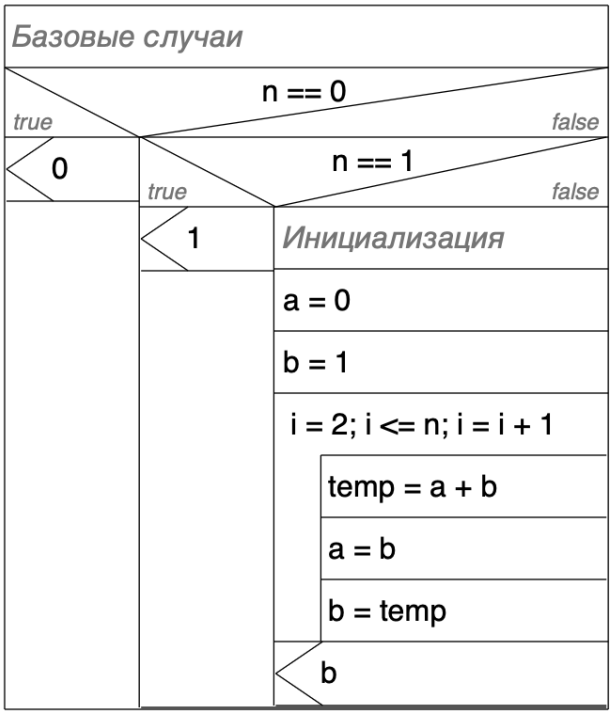
```cpp
int fibRec(int n) {
    if (n <= 1) {
        return n;
    } else {
        return fibRec(n - 1) + fibRec(n - 2);
```

```
    }
}

int fibIter(int n) {
    if (n <= 1) {
        return n;
    }
    int a = 0;
    int b = 1;
    int i = 2;
    while (i <= n) {
        int c = a + b;
        a = b;
        b = c;
        i = i + 1;
    }
    return b;
}
```

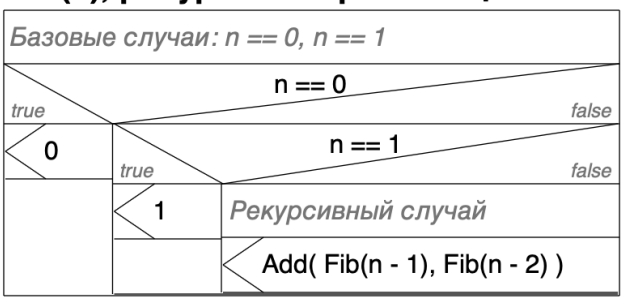
FibIter(n) (рекурсивно)



Fib(n), итеративная реализация



Fib(n), рекурсивная реализация



Трассировка

Выполните трассировку выполнения функций для рекурсивного алгоритма со значением n = 6. Проверьте полученный результат, выполнив программу в пошаговом режиме, используя средства отладки.

Шаг	Операция	Стек вызовов	Комментарий
1	Вызов fibRec(6)	[6]	Вход в рекурсивный вызов: нужно вычислить fibRec(5) + fibRec(4).
2	Вызов fibRec(5)	[6, 5]	Вход в рекурсивный вызов: нужно вычислить fibRec(4) + fibRec(3).
3	Вызов fibRec(4)	[6, 5, 4]	Вход в рекурсивный вызов: нужно вычислить fibRec(3) + fibRec(2).
4	Вызов fibRec(3)	[6, 5, 4, 3]	Вход в рекурсивный вызов: нужно вычислить fibRec(2) + fibRec(1).
5	Вызов fibRec(2)	[6, 5, 4, 3, 2]	Вход в рекурсивный вызов: нужно вычислить fibRec(1) + fibRec(0).
6	Вызов fibRec(1)	[6, 5, 4, 3, 2, 1]	Вход в базовый случай: $n = 1$, сразу вернём 1.

Шаг	Операция	Стек вызовов	Комментарий
7	Возврат из fibRec(1)	[6, 5, 4, 3, 2, 1]	Базовый случай: fibRec(1) = 1.
8	Вызов fibRec(0)	[6, 5, 4, 3, 2, 0]	Вход в базовый случай: $n = 0$, сразу вернём 0.
9	Возврат из fibRec(0)	[6, 5, 4, 3, 2, 0]	Базовый случай: fibRec(0) = 0.
10	Возврат из fibRec(2)	[6, 5, 4, 3, 2]	Завершён fibRec(2): получено значение 1 из fibRec(1) и fibRec(0).
11	Вызов fibRec(1)	[6, 5, 4, 3, 1]	Вход в базовый случай: $n = 1$, сразу вернём 1.
12	Возврат из fibRec(1)	[6, 5, 4, 3, 1]	Базовый случай: fibRec(1) = 1.
13	Возврат из fibRec(3)	[6, 5, 4, 3]	Завершён fibRec(3): получено значение 2 из fibRec(2) и fibRec(1).
14	Вызов fibRec(2)	[6, 5, 4, 2]	Вход в рекурсивный вызов: нужно вычислить fibRec(1) + fibRec(0).
15	Вызов fibRec(1)	[6, 5, 4, 2, 1]	Вход в базовый случай: $n = 1$, сразу вернём 1.
16	Возврат из fibRec(1)	[6, 5, 4, 2, 1]	Базовый случай: fibRec(1) = 1.
17	Вызов fibRec(0)	[6, 5, 4, 2, 0]	Вход в базовый случай: $n = 0$, сразу вернём 0.
18	Возврат из fibRec(0)	[6, 5, 4, 2, 0]	Базовый случай: fibRec(0) = 0.
19	Возврат из fibRec(2)	[6, 5, 4, 2]	Завершён fibRec(2): получено значение 1 из fibRec(1) и fibRec(0).
20	Возврат из fibRec(4)	[6, 5, 4]	Завершён fibRec(4): получено значение 3 из fibRec(3) и fibRec(2).
21	Вызов fibRec(3)	[6, 5, 3]	Вход в рекурсивный вызов: нужно вычислить fibRec(2) + fibRec(1).
22	Вызов fibRec(2)	[6, 5, 3, 2]	Вход в рекурсивный вызов: нужно вычислить fibRec(1) + fibRec(0).
23	Вызов fibRec(1)	[6, 5, 3, 2, 1]	Вход в базовый случай: $n = 1$, сразу вернём 1.
24	Возврат из fibRec(1)	[6, 5, 3, 2, 1]	Базовый случай: fibRec(1) = 1.
25	Вызов fibRec(0)	[6, 5, 3, 2, 0]	Вход в базовый случай: $n = 0$, сразу вернём 0.
26	Возврат из fibRec(0)	[6, 5, 3, 2, 0]	Базовый случай: fibRec(0) = 0.
27	Возврат из fibRec(2)	[6, 5, 3, 2]	Завершён fibRec(2): получено значение 1 из fibRec(1) и fibRec(0).

Шаг	Операция	Стек вызовов	Комментарий
28	Вызов fibRec(1)	[6, 5, 3, 1]	Вход в базовый случай: $n = 1$, сразу вернём 1.
29	Возврат из fibRec(1)	[6, 5, 3, 1]	Базовый случай: fibRec(1) = 1.
30	Возврат из fibRec(3)	[6, 5, 3]	Завершён fibRec(3): получено значение 2 из fibRec(2) и fibRec(1).
31	Возврат из fibRec(5)	[6, 5]	Завершён fibRec(5): получено значение 5 из fibRec(4) и fibRec(3).
32	Вызов fibRec(4)	[6, 4]	Вход в рекурсивный вызов: нужно вычислить fibRec(3) + fibRec(2).
33	Вызов fibRec(3)	[6, 4, 3]	Вход в рекурсивный вызов: нужно вычислить fibRec(2) + fibRec(1).
34	Вызов fibRec(2)	[6, 4, 3, 2]	Вход в рекурсивный вызов: нужно вычислить fibRec(1) + fibRec(0).
35	Вызов fibRec(1)	[6, 4, 3, 2, 1]	Вход в базовый случай: $n = 1$, сразу вернём 1.
36	Возврат из fibRec(1)	[6, 4, 3, 2, 1]	Базовый случай: fibRec(1) = 1.
37	Вызов fibRec(0)	[6, 4, 3, 2, 0]	Вход в базовый случай: $n = 0$, сразу вернём 0.
38	Возврат из fibRec(0)	[6, 4, 3, 2, 0]	Базовый случай: fibRec(0) = 0.
39	Возврат из fibRec(2)	[6, 4, 3, 2]	Завершён fibRec(2): получено значение 1 из fibRec(1) и fibRec(0).
40	Вызов fibRec(1)	[6, 4, 3, 1]	Вход в базовый случай: $n = 1$, сразу вернём 1.
41	Возврат из fibRec(1)	[6, 4, 3, 1]	Базовый случай: fibRec(1) = 1.
42	Возврат из fibRec(3)	[6, 4, 3]	Завершён fibRec(3): получено значение 2 из fibRec(2) и fibRec(1).
43	Вызов fibRec(2)	[6, 4, 2]	Вход в рекурсивный вызов: нужно вычислить fibRec(1) + fibRec(0).
44	Вызов fibRec(1)	[6, 4, 2, 1]	Вход в базовый случай: $n = 1$, сразу вернём 1.
45	Возврат из fibRec(1)	[6, 4, 2, 1]	Базовый случай: fibRec(1) = 1.
46	Вызов fibRec(0)	[6, 4, 2, 0]	Вход в базовый случай: $n = 0$, сразу вернём 0.
47	Возврат из fibRec(0)	[6, 4, 2, 0]	Базовый случай: fibRec(0) = 0.

Шаг	Операция	Стек вызовов	Комментарий
48	Возврат из fibRec(2)	[6, 4, 2]	Завершён fibRec(2): получено значение 1 из fibRec(1) и fibRec(0).
49	Возврат из fibRec(4)	[6, 4]	Завершён fibRec(4): получено значение 3 из fibRec(3) и fibRec(2).
50	Возврат из fibRec(6)	[6]	Завершён fibRec(6): получено значение 8 из fibRec(5) и fibRec(4).
51	Завершение работы fibRec(6)	[ ]	Все рекурсивные вызовы завершены, стек пуст.

Сравнение рекурсивный и итерационных реализаций

Таким образом, при вычислении fibRec(6) по классическому рекурсивному определению строится дерево вызовов, где каждый небазовый вызов fibRec(n) (при $n \geq 2$) порождает два подвызыва: fibRec(n-1) и fibRec(n-2), причём обход идёт в глубину слева направо. В сумме получается 25 вызовов функции (включая базовые случаи fibRec(0) и fibRec(1)) и, соответственно, 25 возвратов из этих вызовов.
Каждый лист дерева соответствует базовому случаю с известным значением (fibRec(0) = 0, fibRec(1) = 1), а при подъёме вверх по дереву и по стеку вызовов последовательно вычисляются значения fibRec(2) = 1, fibRec(3) = 2, fibRec(4) = 3, fibRec(5) = 5. В итоге корневой вызов fibRec(6) завершается с результатом 8, после чего стек вызовов полностью опустошается.

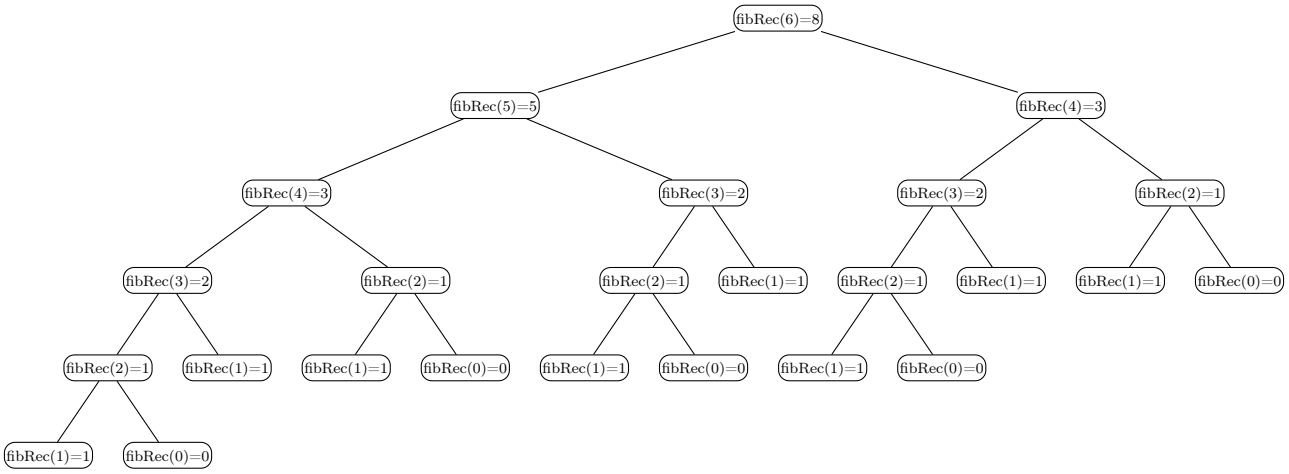


Рис. 1: Дерево вызовов для fibRec(6)

3 Задание 3

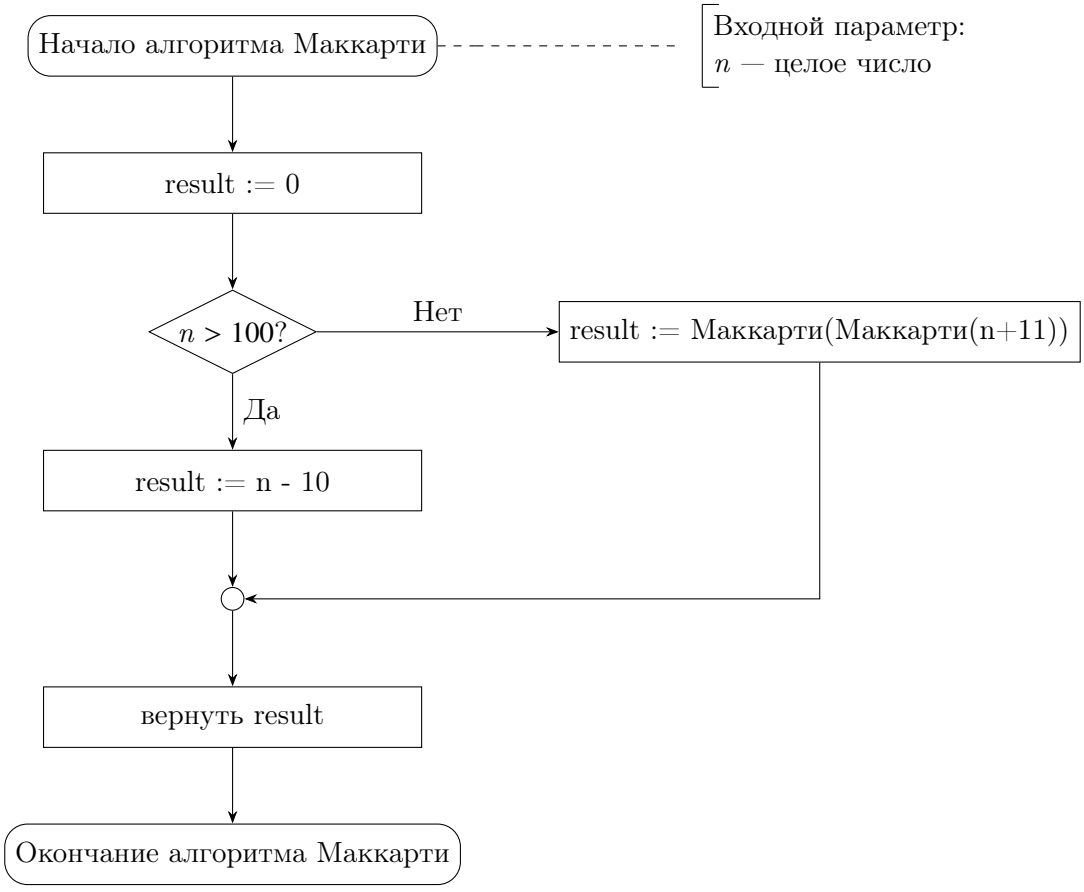
Функция Маккарти. Порядок вычисления

Покажите порядок вычисления функции Маккарти при вызове функции со значением параметра n = 98

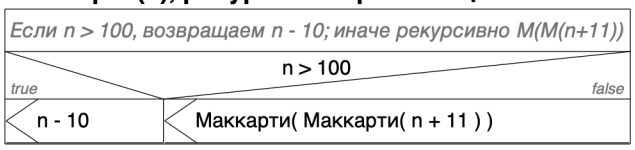
Определение.

$$M(n) = \begin{cases} n - 10, & n > 100, \\ M(M(n + 11)), & n \le 100 \end{cases}$$

Маккарти(n) (рекурсивно)



Маккарти(n), рекурсивная реализация



Порядок вычисления M(98).

$$
\begin{aligned}
M(98) &= M\big(M(98+11)\big) && (98 \le 100)\\
&= M\big(M(109)\big)\\
&= M(109-10) && (109 > 100)\\
&= M(99)\\
&= M\big(M(99+11)\big) && (99 \le 100)\\
&= M\big(M(110)\big)\\
&= M(110-10) && (110 > 100)\\
&= M(100)\\
&= M\big(M(100+11)\big) && (100 \le 100)\\
&= M\big(M(111)\big)\\
&= M(111-10) && (111 > 100)\\
&= M(101)\\
&= 101-10 && (101 > 100)\\
&= 91
\end{aligned}
$$

Функция Маккарти. C++

Напишите программу на языке C++, которая позволила бы ввести данные (число n) с клавиатуры и вычислить и вывести на экран значение функции Маккарти.

```cpp
#include <iostream>
#include <cstdio>

int McCarthy(int n) {
    if (n > 100) {
        return n - 10;
    } else {
        return McCarthy(McCarthy(n + 11));
    }
}

int main() {
    int n;

    printf("Введите n: ");
    std::cin >> n;

    int result = McCarthy(n);

    printf("M(n) = %d\n", result);

    return 0;
}
```

Порядок вычисления $M(94)$.

$$\begin{aligned}
M(94) &= M\big(M(94+11)\big) && (94 \le 100)\\
&= M\big(M(105)\big)\\
&= M(105-10) && (105 > 100)\\
&= M(95)\\
&= M\big(M(95+11)\big) && (95 \le 100)\\
&= M\big(M(106)\big)\\
&= M(106-10) && (106 > 100)\\
&= M(96)\\
&= M\big(M(96+11)\big) && (96 \le 100)\\
&= M\big(M(107)\big)\\
&= M(107-10) && (107 > 100)\\
&= M(97)\\
&= M\big(M(97+11)\big) && (97 \le 100)\\
&= M\big(M(108)\big)\\
&= M(108-10) && (108 > 100)\\
&= M(98)\\
&= M\big(M(109)\big) = M(99) && (109 > 100)\\
&= M\big(M(110)\big) = M(100) && (110 > 100)\\
&= M\big(M(111)\big) = M(101) && (111 > 100)\\
&= 101 - 10 = 91 && (101 > 100)
\end{aligned}$$

$$\Rightarrow\ M(94) = 91.$$

Порядок вычисления $M(102)$.

$$\begin{aligned}
M(102) &= 102 - 10 && (102 > 100)\\
&= 92.
\end{aligned}$$

Порядок вычисления $M(113)$.

$$\begin{aligned}
M(113) &= 113 - 10 && (113 > 100)\\
&= 103.
\end{aligned}$$

Итог.

$$M(94) = 91, \quad M(102) = 92, \quad M(113) = 103.$$

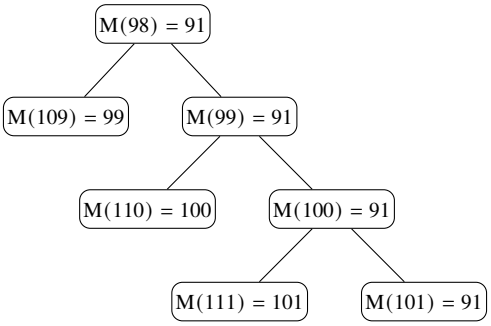


Рис. 2: Дерево вызовов для M(98)

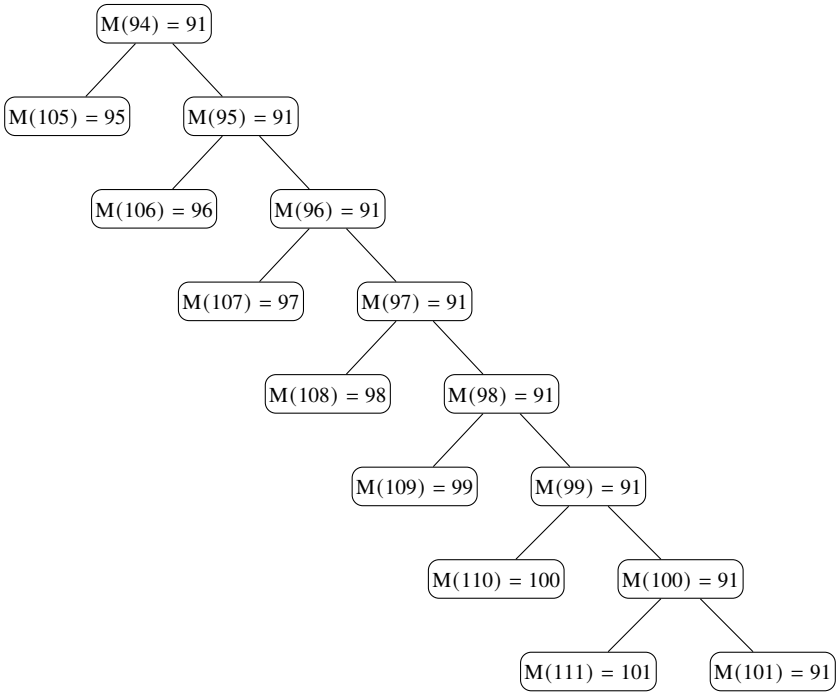


Рис. 3: Дерево вызовов для M(94)

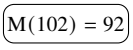


Рис. 4: Дерево вызовов для M(102)

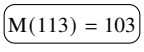
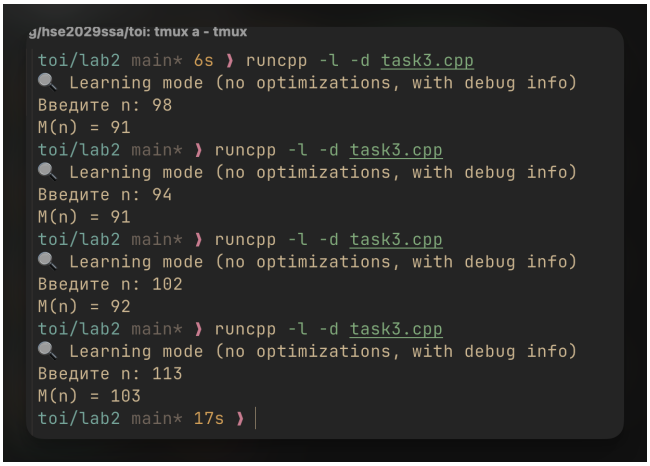


Рис. 5: Дерево вызовов для M(113)



4 Задание 4

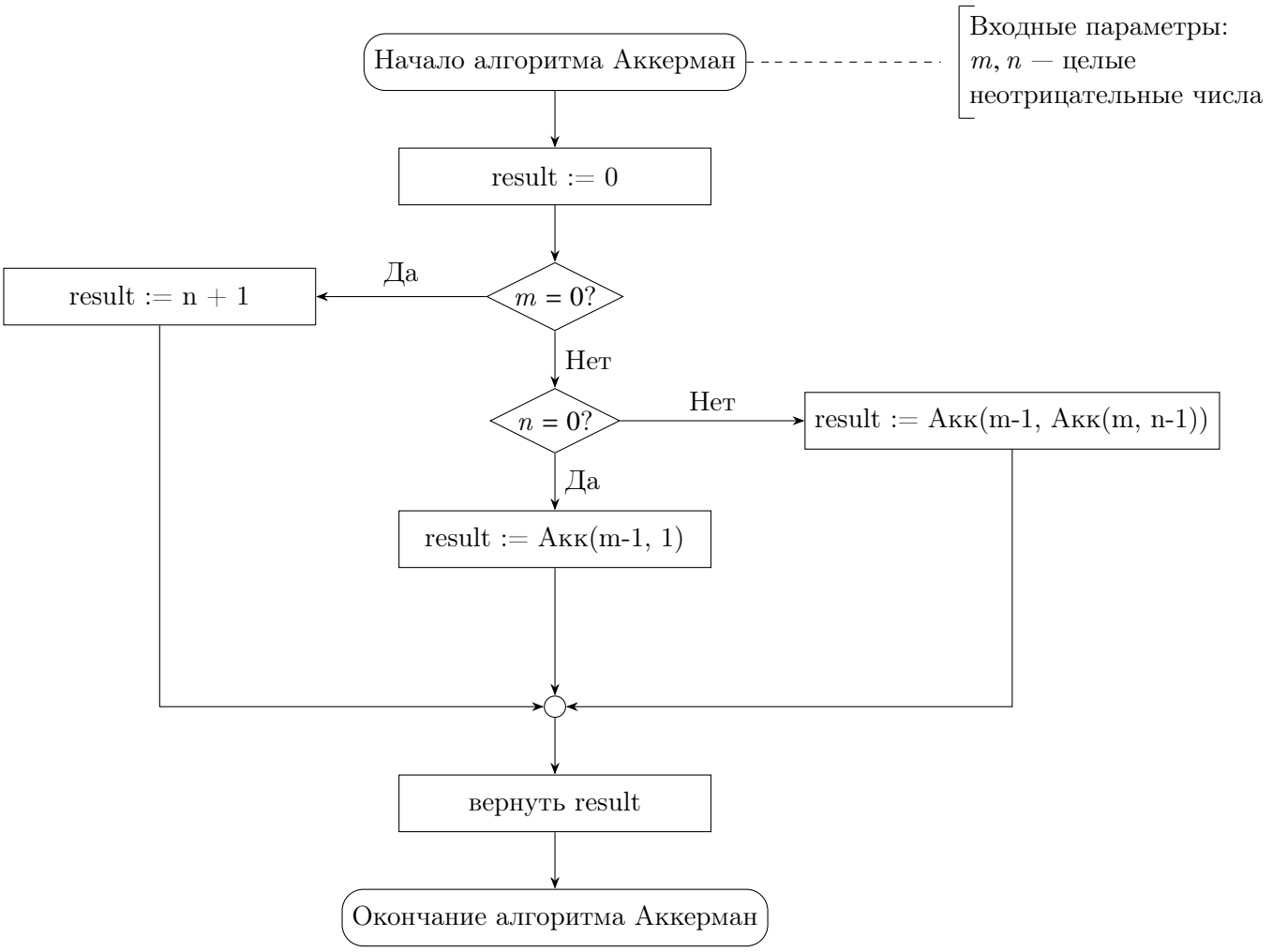
Функция Аккермана. Порядок вычисления

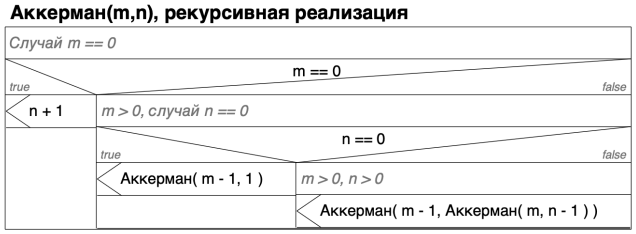
Покажите порядок вычисления функции Аккермана при вызове функции со следующими значениями параметров: n = 2, m = 1

Определение.

$$A(m,n) = \begin{cases} n+1, & m = 0, \\ A(m-1,1), & m > 0 \ \wedge \ n = 0, \\ A\big(m-1,\ A(m,n-1)\big), & m > 0 \ \wedge \ n > 0. \end{cases}$$

(m, n) (рекурсивно)





Порядок вычисления $A(1,2)$.

$$
\begin{aligned}
A(1,2) &= A\big(0,\ A(1,1)\big) && (m>0, n>0)\\
&= A\Big(0,\ A\big(0,\ A(1,0)\big)\Big)\\
&= A\Big(0,\ A\big(0,\ A(0,1)\big)\Big) && (m>0, n=0)\\
&= A\Big(0,\ A\big(0,\ 2\big)\Big) && (A(0,1)=2)\\
&= A\Big(0,\ A(0,2)\Big)\\
&= A\Big(0,\ 3\Big) && (A(0,2)=3)\\
&= 4 && (A(0,3)=4).
\end{aligned}
$$

Функция Аккермана. C++

```cpp
#include <iostream>
#include <cstdio>

int Ackerman(int m, int n) {
    if (m == 0) {
        return n + 1;
    } else if (n == 0) {
        return Ackerman(m - 1, 1);
    } else {
        return Ackerman(m - 1, Ackerman(m, n - 1));
    }
}

int main() {
    int m;
    int n;

    printf("Введите m и n: ");
    std::cin >> m >> n;

    int result = Ackerman(m, n);

    printf("A(%d, %d) = %d\n", m, n, result);

    return 0;
}
```

Порядок вычисления $A(0,4)$.

$$\begin{aligned} A(0,4) &= 4+1 \\ &= 5. \end{aligned}$$

Порядок вычисления $A(1,0)$.

$$\begin{aligned} A(1,0) &= A(0,1) \qquad (m>0, n=0) \\ &= 2. \end{aligned}$$

Порядок вычисления $A(2,2)$.

$$
\begin{aligned}
A(2,2) &= A\big(1,\ A(2,1)\big) && (m>0, n>0)\\
&= A\Big(1,\ A\big(1,\ A(2,0)\big)\Big)\\
&= A\Big(1,\ A\big(1,\ A(1,1)\big)\Big) && (m>0, n=0)\\
&= A\Big(1,\ A\big(1,\ A(0,\ A(1,0))\big)\Big)\\
&= A\Big(1,\ A\big(1,\ A(0,\ A(0,1))\big)\Big)\\
&= A\Big(1,\ A\big(1,\ A(0,\ 2)\big)\Big)\\
&= A\Big(1,\ A\big(1,\ 3\big)\Big) && (A(0,2)=3)\\
&= A\Big(1,\ A\big(0,\ A(1,2)\big)\Big)\\
&= A\Big(1,\ A\big(0,\ 4\big)\Big) && (\text{см. выше: } A(1,2)=4)\\
&= A\big(1,\ 5\big) && (A(0,4)=5)\\
&= A\big(0,\ A(1,4)\big)\\
&= A\Big(0,\ A\big(0,\ A(1,3)\big)\Big)\\
&= A\Big(0,\ A\big(0,\ A\big(0,\ A(1,2)\big)\big)\Big)\\
&= A\Big(0,\ A\big(0,\ A\big(0,\ 4\big)\big)\Big)\\
&= A\Big(0,\ A\big(0,\ 5\big)\Big)\\
&= A\big(0,\ 6\big)\\
&= 7.
\end{aligned}
$$

Итог.

$$A(0,4)=5, \quad A(1,0)=2, \quad A(2,2)=7.$$

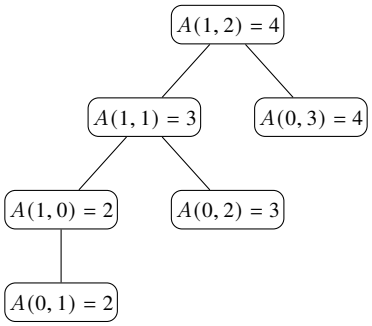


Рис. 6: Дерево вызовов для $A(1,2)$

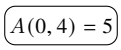


Рис. 7: Дерево вызовов для $A(0,4)$

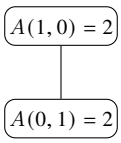


Рис. 8: Дерево вызовов для $A(1,0)$

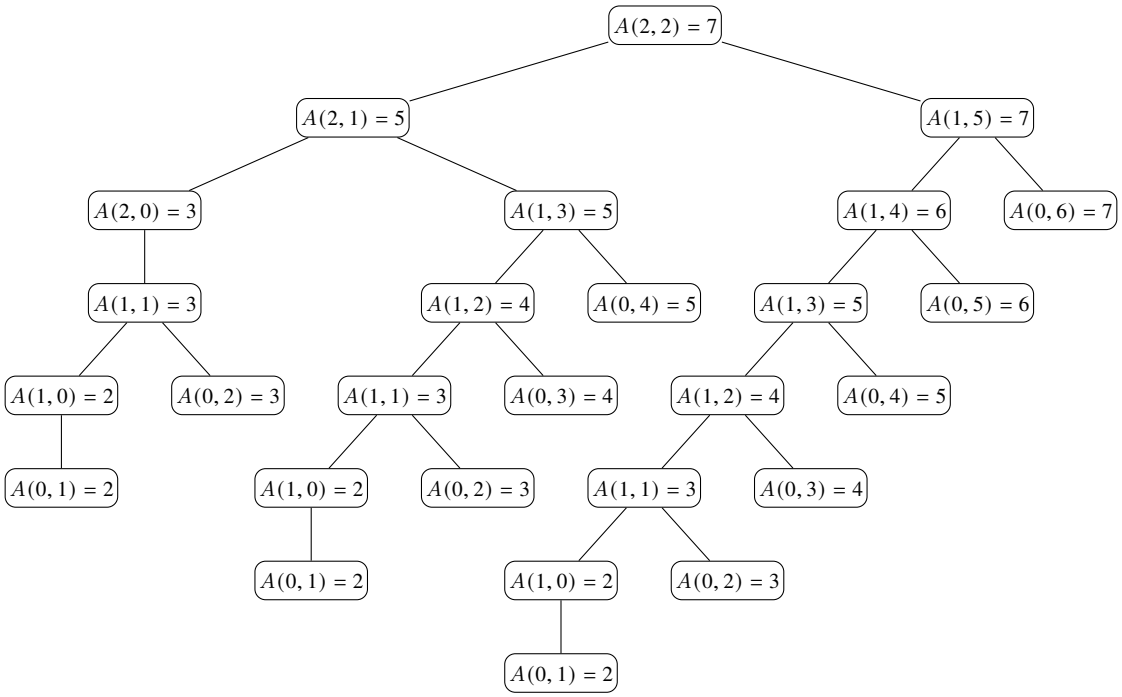


Рис. 9: Дерево вызовов для $A(2,2)$

```
toi/lab2 main* 17s ❱ runcpp -l -d task4.cpp
🔍 Learning mode (no optimizations, with debug info)
Введите m и n: 0
4
A(0, 4) = 5
toi/lab2 main* 16s ❱ runcpp -l -d task4.cpp
🔍 Learning mode (no optimizations, with debug info)
Введите m и n: 1
0
A(1, 0) = 2
toi/lab2 main* ❱ runcpp -l -d task4.cpp
🔍 Learning mode (no optimizations, with debug info)
Введите m и n: 2
2
A(2, 2) = 7
toi/lab2 main* ❱ runcpp -l -d task4.cpp
🔍 Learning mode (no optimizations, with debug info)
Введите m и n: 1
2
A(1, 2) = 4
```

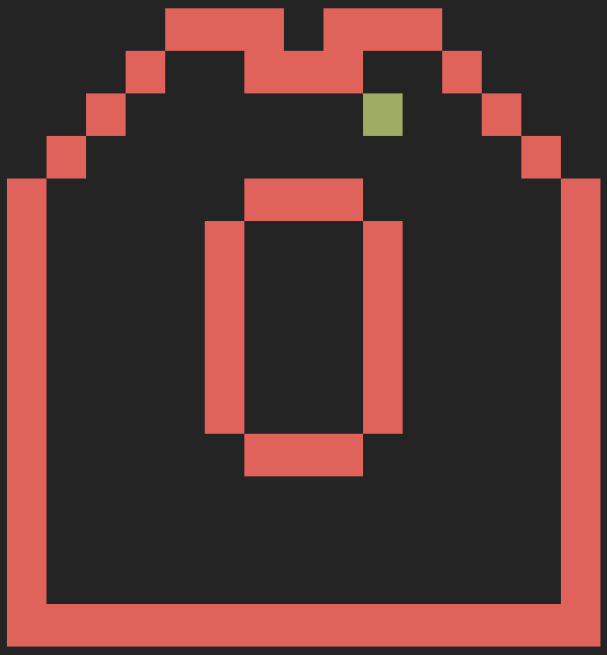
5 Задание 5

Напишите рекурсивную процедуру закраски фигуры. Покажите порядок закраски по шагам

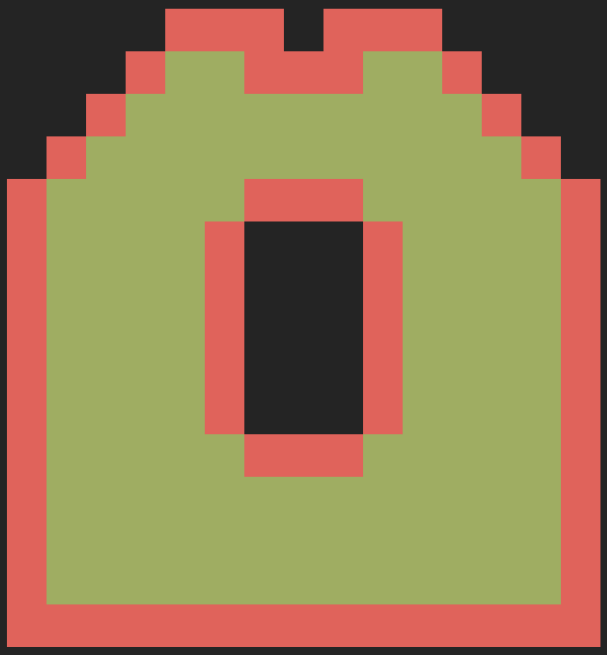
```cpp

#include <cstdio>

const int H = 17;
const int W = 17;

// '.' - пусто, '#' - граница, '+' - цвет заливки
char picture[H][W + 1] = {
    ".................",
    ".....###.###.....",
    "....#..###..#....",
    "...#.....+..#...",
    "..#.........#..",
    ".#.....###.....#.",
    ".#....#...#....#.",
    ".#....#...#....#.",
    ".#....#...#....#.",
    ".#....#...#....#.",
    ".#....#...#....#.",
    ".#.....###.....#.",
    ".#.............#.",
    ".#.............#.",
    ".#.............#.",
    ".################.",
    "................."
};

void print_picture() {
    for (int y = 0; y < H; y++) {
        for (int x = 0; x < W; x++) {
            char c = picture[y][x];

            if (c == '#') {
                // граница - красный фон
                printf("\x1b[41m  \x1b[0m");
            } else if (c == '+') {
                // заливка - зелёный фон
                printf("\x1b[42m  \x1b[0m");
            } else {
                // фон - просто пробелы
                printf("  ");
            }
        }
        printf("\n");
    }
}

void fill(int x, int y, char border_color, char the_color) {
    if (x < 0 || x >= W || y < 0 || y >= H) {
        return;
    }

    char c = picture[y][x];

```

```
55      if (c != border_color && c != the_color) {
56          picture[y][x] = the_color;
57
58          fill(x - 1, y, border_color, the_color);
59          fill(x + 1, y, border_color, the_color);
60          fill(x, y - 1, border_color, the_color);
61          fill(x, y + 1, border_color, the_color);
62      }
63  }
64
65  int main() {
66      int start_x = -1;
67      int start_y = -1;
68
69      printf("Before fill:\n");
70      print_picture();
71
72      // ищем единственный первый плюсик
73      for (int y = 0; y < H; y++) {
74          for (int x = 0; x < W; x++) {
75              if (picture[y][x] == '+') {
76                  start_x = x;
77                  start_y = y;
78              }
79          }
80      }
81
82      if (start_x != -1) {
83          picture[start_y][start_x] = '.';
84          fill(start_x, start_y, '#', '+');
85      }
86
87      printf("\nAfter fill:\n");
88      print_picture();
89
90      return 0;
91  }
```

```
toi/lab2 main* ❱ runcpp -l -d task5.cpp
🔍 Learning mode (no optimizations, with debug info)
Before fill:
```



```
After fill:
```



```
toi/lab2 main* ❱
```

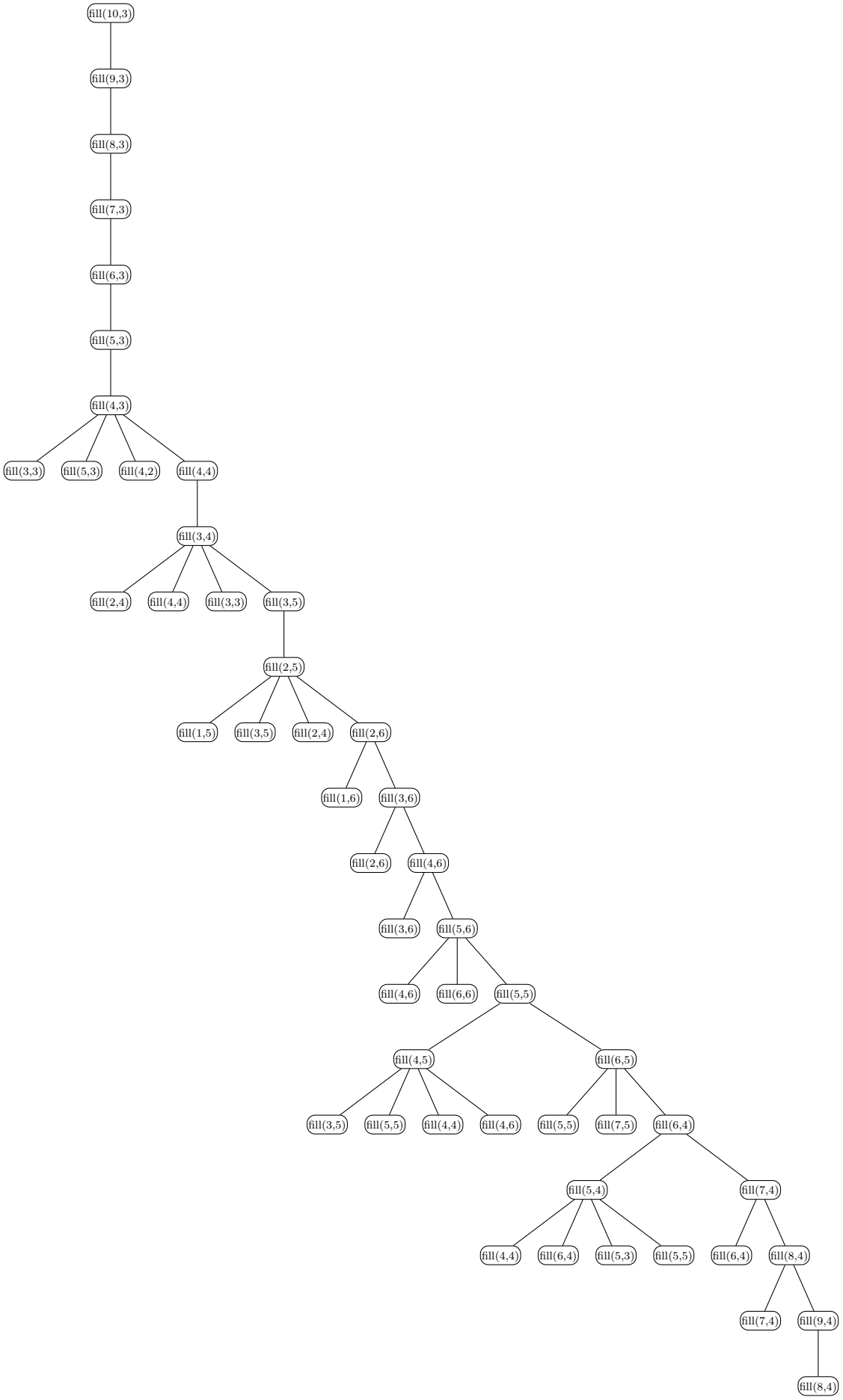


Рис. 10: Фрагмент дерева первых вызовов