# Лабораторная работа №8

Тема: Целочисленная арифметика многократной точности

Выполнила: Исламова Сания Маратовна

Группа: НПИмд-01-24

Студ.билет: 1132249576

## Задача лабораторной работы:

Реализовать рассмотренные алгоритмы программно: Сложение неотрицательных целых чисел, Вычитание неотрицательных целых чисел, Умножение неотрицательных целых чисел столбиком, Быстрый столбик, Деление многоразрядных целых числе

## Описание хода выполнения лабораторной работы:

```
# Функция преобразования массива цифр в строку в системе счисления b
function digits_to_str(dig::Vector{Int}, b::Int)::String
    # Если массив пустой или состоит только из нулей — число равно 0
    if isempty(dig) || all(==(0), dig)
        return "0"
    end
    # Собираем строку из цифр, начиная со старшего разряда (reverse)
    # Если цифра ≥10, преобразуем в букву A–Z (для оснований >10)
    join( [d < 10 ? string(d) : string(Char('A' + d - 10)) for d in reverse(dig)], "")
    # Результат: строка, представляющая число в основании b, без ведущих нулей
end

# Функция преобразования строки в массив цифр (младший разряд — индекс 1)
function str_to_digits(s::String, b::Int)::Vector{Int}
    # Убираем пробелы и приводим к нижнему регистру для удобства
    s = strip(lowercase(s))
    # Пустая строка или "0" → представляем как [0]
    if s == "" || s == "0" return [0] end
    digits = Int[]  # Создаём пустой массив для цифр
    # Проходим по символам строки справа налево (младшие разряды первые)
    for c in reverse(s)
        # Преобразуем символ в цифру: 0–9 или A–Z → 10–35
        d = isdigit(c) ? c - '0' : (uppercase(c) - 'A' + 10)
        # Проверяем, что цифра допустима в данном основании
        if d < 0 || d >= b
            error("Недопустимая цифра '$c' в основании $b")
        end
        push!(digits, d)  # Добавляем цифру в массив (младшая первая)
    end
    # Удаляем ведущие нули (кроме случая, когда число 0)
    while length(digits) > 1 && digits[end] == 0
        pop!(digits)
    end
    digits  # Результат : массив цифр , младший разряд — digits[1]
end

# Удаление ведущих нулей из массива цифр
function trim(dig::Vector{Int})::Vector{Int}
    # Пока больше одного разряда и старший разряд нулевой — удаляем его
    while length(dig) > 1 && dig[end] == 0
        pop!(dig)
    end
    dig  # Результат: массив без ведущих нулей (кроме [0] для нуля)
end

# Алгоритм 1: Сложение неотрицательных чисел (по лабораторной)
```

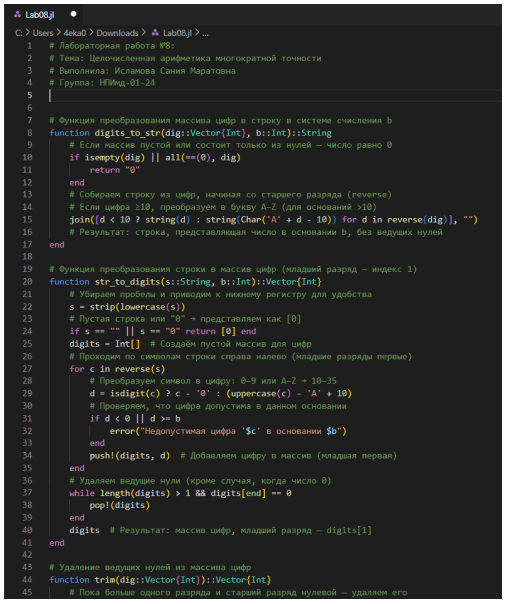
```
function add_big(u::Vector{Int}, v::Vector{Int}, b::Int)::Vector{Int}
    n = max(length(u), length(v))  # Определяем максимальную разрядность
    w = zeros(Int, n + 1)         # Результирующий массив на один разряд больше (для переноса)
    k = 0                         # Перенос, изначально 0
    for j = 1:n                   # Проходим по разрядам от младшего к старшему
        uj = j <= length(u) ? u[j] : 0  # Берем цифру из u или 0, если разряд кончился
        vj = j <= length(v) ? v[j] : 0  # Аналогично для v
        s = uj + vj + k           # Сумма цифр + перенос
        w[j] = s % b              # Записываем младшую часть суммы в текущий разряд
        k = s ÷ b                 # Новый перенос в старший разряд
    end
    w[n+1] = k                    # Записываем финальный перенос (w ₀ в алгоритме)
    trim(w)                       # Убираем ведущие нули
    # Результат: сумма u + v в основании b
end

# Алгоритм 2: Вычитание u - v (u ≥ v ≥ 0)
function sub_big(u::Vector{Int}, v::Vector{Int}, b::Int)::Vector{Int}
    n = length(u)                 # Разрядность берём по первому числу (u ≥ v)
    w = zeros(Int, n)             # Результирующий массив
    k = 0                         # Заём из старшего разряда, изначально 0
    for j = 1:n
        uj = u[j]                 # Цифра из уменьшаемого
        vj = j <= length(v) ? v[j] : 0  # Цифра из вычитаемого или 0
        s = uj - vj - k           # Разность с учётом займа
        if s < 0                  # Если получилась отрицательная цифра
            s += b                # Занимаем из старшего разряда (добавляем основание)
            k = 1                 # Устанавливаем заём для следующего разряда
        else
            k = 0                 # Заём не нужен
        end
        w[j] = s                  # Записываем цифру результата
    end
    trim(w)                       # Убираем ведущие нули
    # Результат: разность u - v в основании b
end

# Алгоритм 3: Умножение "столбиком"
function mul_classic(u::Vector{Int}, v::Vector{Int}, b::Int)::Vector{Int}
    n, m = length(u), length(v)   # Разрядности множимого и множителя
    w = zeros(Int, n + m)         # Результат до n+m разрядов
    for j = 1:m                   # По разрядам множителя v (от младшего)
        v[j] == 0 && continue     # Если цифра 0 — пропускаем (оптимизация)
        k = 0                     # Перенос для текущего "столбика"
        for i = 1:n               # По разрядам множимого u
            t = u[i] * v[j] + w[i+j-1] + k  # Произведение + уже накопленное + перенос
            w[i+j-1] = t % b      # Записываем в текущий разряд
            k = t ÷ b             # Перенос в следующий разряд
        end
        w[n+j] = k                # Записываем оставшийся перенос
    end
    trim(w)                       # Убираем ведущие нули
    # Результат: произведение u × v в основании b
end

# Алгоритм 4: Умножение "быстрым столбиком"
function mul_fast(u::Vector{Int}, v::Vector{Int}, b::Int)::Vector{Int}
    n, m = length(u), length(v)
```

```
    w = zeros(Int, n + m)         # Результат до n+m разрядов
    t = 0                         # Накопитель промежуточной суммы
    for s = 0:n+m-2               # s — диагональ в "столбике"
        low = max(0, s - m + 1)   # Нижняя граница индекса i
        high = min(s, n - 1)      # Верхняя граница индекса i
        for i = low:high          # Суммируем все произведения на этой диагонали
            t += u[i+1] * v[s-i+1]
        end
        w[n+m-s-1] = t % b        # Записываем цифру в соответствующий разряд
        t ÷= b                    # Переносим остаток в следующую диагональ
    end
    w[1] = t                      # Последний перенос в старший разряд
    trim(w)
    # Результат: произведение u × v (тот же, что и в mul_classic, но другой алгоритм)
end

# Алгоритм 5: Деление с остатком
function div_big(u_::Vector{Int}, v_::Vector{Int}, b::Int)
    u = copy(u_)                  # Копируем, чтобы не изменять оригинал
    v = trim(copy(v_))            # Копируем и убираем ведущие нули у делителя
    n, t = length(u), length(v)   # Разрядности делимого и делителя
    if t == 0 || all(==(0), v)    # Проверка деления на ноль
        error("Деление на ноль")
    end
    q = zeros(Int, n - t + 1)     # Массив для частного (максимальная длина)
    for i = n:-1:t+1              # По разрядам делимого от старшего
        hi = i <= n ? u[i] : 0    # Текущий старший разряд
        mi = i-1 >= 1 ? u[i-1] : 0 # Следующий разряд
        lo = i-2 >= 1 ? u[i-2] : 0 # Ещё один для точной оценки
        # Оценка цифры частного
        qhat = hi >= v[t] ? b - 1 : (hi * b + mi) ÷ v[t]
        v1 = v[t] * b + (t >= 2 ? v[t-1] : 0)  # Для проверки переполнения
        # Корректируем оценку вниз, если слишком большая
        while qhat > 0 && qhat * v1 > hi * b*b + mi * b + lo
            qhat -= 1
        end
        borrow = 0                # Заём при вычитании
        for j = 1:t               # Вычитаем qhat × v × b^(i-t)
            pos = i - t + j
            if pos > length(u)    # Если нужно — расширяем массив u
                resize!(u, pos)
                u[pos] = 0
            end
            temp = qhat * v[j] + borrow
            u[pos] -= temp % b
            borrow = temp ÷ b
            if u[pos] < 0         # Если отрицательно — занимаем
                u[pos] += b
                borrow += 1
            end
        end
        pos_carry = i + 1
        if pos_carry <= length(u)
            u[pos_carry] -= borrow  # Вычитаем заём из старшего разряда
            if u[pos_carry] < 0    # Если переполнение — корректируем
                u[pos_carry] += b
                qhat -= 1
            end
```

```
        end
        q[i - t] = qhat          # Записываем цифру частного
    end
    r = length(u) >= t ? u[1:t] : u  # Остаток — младшие t разрядов
    trim(q), trim(r)             # Убираем ведущие нули
    # Результат: кортеж (частное, остаток)
end


println("Лабораторная работа №8: Арифметика многократной точности")
println("0 — Выход")
println("1 — Сложение")
println("2 — Вычитание")
println("3 — Умножение столбиком")
println("4 — Умножение быстрым столбиком")
println("5 — Деление с остатком\n")

# Бесконечный цикл — программа работает, пока пользователь не выберет выход
while true
    print("Выберите алгоритм (0 для выхода): ")
    input = strip(readline())                # Считываем и убираем лишние пробелы
    input == "0" && (println("До свидания!"); break)  # Выход по 0

    # Преобразуем ввод в число, если ошибка — сообщаем и продолжаем цикл
    alg = try parse(Int, input) catch
        println("Неверный выбор \n"); continue
    end
    if !(1 <= alg <= 5)                      # Проверяем диапазон
        println("Выберите от 1 до 5\n"); continue
    end

    print("Основание b (2–36): ")
    b = try parse(Int, readline()) catch     # Считываем основание
        println("Неверное основание\n"); continue
    end
    if !(2 <= b <= 36)                       # Проверяем допустимость
        println("b должно быть от 2 до 36\n"); continue
    end

    print("Первое число (в системе $b): ")
    s1 = readline()                          # Ввод первого числа как строки
    print("Второе число (в системе $b): ")
    s2 = readline()                          # Ввод второго числа

    try
        # Преобразуем строки в массивы цифр
        u = str_to_digits(s1, b)
        v = str_to_digits(s2, b)

        # Выполняем выбранный алгоритм
        if alg == 1
            res = add_big(u, v, b)
            println("Сумма : $(digits_to_str(res, b))\n")
        elseif alg == 2
            # Проверка условия u ≥ v для вычитания
            if length(u) < length(v) || (length(u) == length(v) && u < v)
                println("Ошибка: первое число должно быть ≥ второго\n")
            else
```

```
            res = sub_big(u, v, b)
            println("Разность : $(digits_to_str(res, b))\n")
        end
    elseif alg == 3
        res = mul_classic(u, v, b)
        println("Произведение (столбиком ): $(digits_to_str(res, b))\n")
    elseif alg == 4
        res = mul_fast(u, v, b)
        println("Произведение (быстро ): $(digits_to_str(res, b))\n")
    elseif alg == 5
        q, r = div_big(u, v, b)
        println("Частное : $(digits_to_str(q, b))")
        println("Остаток : $(digits_to_str(r, b))\n")
    end
  catch e
    # Ловим все ошибки ввода (недопустимые цифры и т.д.)
    println ( "Ошибка : $(sprint(showerror, e))\n")
  end
end
```

## Результат реализации рассмотренного алгоритма




Lab08.jl

C: > Users > 4eka0 > Downloads > Lab08.jl > ...

```julia
# Алгоритм 3: Умножение "столбиком"
function mul_classic(u::Vector{Int}, v::Vector{Int}, b::Int)::Vector{Int}
    n, m = length(u), length(v)   # Разрядности множимого и множителя
    w = zeros(Int, n + m)         # Результат до n+m разрядов
    for j = 1:m                   # По разрядам множителя v (от младшего)
        v[j] == 0 && continue     # Если цифра 0 — пропускаем (оптимизация)
        k = 0                     # Перенос для текущего "столбика"
        for i = 1:n               # По разрядам множимого u
            t = u[i] * v[j] + w[i+j-1] + k  # Произведение + уже накопленное + перенос
            w[i+j-1] = t % b      # Записываем в текущий разряд
            k = t ÷ b             # Перенос в следующий разряд
        end
        w[n+j] = k                # Записываем оставшийся перенос
    end
    trim(w)                       # Убираем ведущие нули
    # Результат: произведение u × v в основании b
end

# Алгоритм 4: Умножение "быстрым столбиком"
function mul_fast(u::Vector{Int}, v::Vector{Int}, b::Int)::Vector{Int}
    n, m = length(u), length(v)
    w = zeros(Int, n + m)         # Результат до n+m разрядов
    t = 0                         # Накопитель промежуточной суммы
    for s = 0:n+m-2               # s — диагональ в "столбике"
        low = max(0, s - m + 1)   # Нижняя граница индекса i
        high = min(s, n - 1)      # Верхняя граница индекса i
        for i = low:high          # Суммируем все произведения на этой диагонали
            t += u[i+1] * v[s-i+1]
        end
        w[n+m-s-1] = t % b        # Записываем цифру в соответствующий разряд
        t ÷= b                    # Переносим остаток в следующую диагональ
    end
    w[1] = t                      # Последний перенос в старший разряд
    trim(w)
    # Результат: произведение u × v (тот же, что и в mul_classic, но другой алгоритм)
end
```

Lab08.jl

C: > Users > 4eka0 > Downloads > Lab08.jl > ...

```julia
# Алгоритм 5: Деление с остатком
function div_big(u_::Vector{Int}, v_::Vector{Int}, b::Int)
    u = copy(u_)                  # Копируем, чтобы не изменять оригинал
    v = trim(copy(v_))            # Копируем и убираем ведущие нули у делителя
    n, t = length(u), length(v)   # Разрядности делимого и делителя
    if t == 0 || all(==(0), v)    # Проверка деления на ноль
        error("Деление на ноль")
    end
    q = zeros(Int, n - t + 1)     # Массив для частного (максимальная длина)
    for i = n:-1:t+1              # По разрядам делимого от старшего
        hi = i <= n ? u[i] : 0    # Текущий старший разряд
        mi = i-1 >= 1 ? u[i-1] : 0 # Следующий разряд
        lo = i-2 >= 1 ? u[i-2] : 0 # Ещё один для точной оценки
        # Оценка цифры частного
        qhat = hi >= v[t] ? b - 1 : (hi * b + mi) ÷ v[t]
        vl = v[t] * b + (t >= 2 ? v[t-1] : 0)  # Для проверки переполнения
        # Корректируем оценку вниз, если слишком большая
        while qhat > 0 && qhat * vl > hi * b^b + mi * b + lo
            qhat -= 1
        end
        borrow = 0                # Заём при вычитании
        for j = 1:t               # Вычитаем qhat × v × b^(i-t)
            pos = i - t + j
            if pos > length(u)    # Если нужно — расширяем массив u
                resize!(u, pos)
                u[pos] = 0
            end
            temp = qhat * v[j] + borrow
            u[pos] -= temp % b
            borrow = temp ÷ b
            if u[pos] < 0         # Если отрицательно — занимаем
                u[pos] += b
                borrow += 1
            end
        end
        pos_carry = i + 1
        if pos_carry <= length(u)
            u[pos_carry] -= borrow  # Вычитаем заём из старшего разряда
            if u[pos_carry] < 0   # Если переполнение — корректируем
                u[pos_carry] += b
                qhat -= 1
            end
        end
        q[i - t] = qhat           # Записываем цифру частного
```

Lab08.jl

C: > Users > 4eka0 > Downloads > Lab08.jl > ...

```julia
function div_big(u_::Vector{Int}, v_::Vector{Int}, b::Int)
    end
    r = length(u) >= t ? u[1:t] : u  # Остаток — младшие t разрядов
    trim(q), trim(r)              # Убираем ведущие нули
    # Результат: кортеж (частное, остаток)
end


println("Лабораторная работа №8: Арифметика многократной точности")
println("0 — Выход")
println("1 — Сложение")
println("2 — Вычитание")
println("3 — Умножение столбиком")
println("4 — Умножение быстрым столбиком")
println("5 — Деление с остатком\n")

# Бесконечный цикл — программа работает, пока пользователь не выберет выход
while true
    print("Выберите алгоритм (0 для выхода): ")
    input = strip(readline())     # Считываем и убираем лишние пробелы
    input == "0" && (println("До свидания!"); break)  # Выход по 0

    # Преобразуем ввод в число, если ошибка — сообщаем и продолжаем цикл
    alg = try parse(Int, input) catch
        println("Неверный выбор\n"); continue
    end
    if !(1 <= alg <= 5)           # Проверяем диапазон
        println("Выберите от 1 до 5\n"); continue
    end

    print("Основание b (2-36): ")
    b = try parse(Int, readline()) catch    # Считываем основание
        println("Неверное основание\n"); continue
    end
    if !(2 <= b <= 36)            # Проверяем допустимость
        println("b должно быть от 2 до 36\n"); continue
    end

    print("Первое число (в системе $b): ")
    s1 = readline()               # Ввод первого числа как строки
    print("Второе число (в системе $b): ")
    s2 = readline()               # Ввод второго числа
    println()

    try
```

Lab08.jl

C: > Users > 4eka0 > Downloads > Lab08.jl > ...

```julia
        # Преобразуем строки в массивы цифр
        u = str_to_digits(s1, b)
        v = str_to_digits(s2, b)

        # Выполняем выбранный алгоритм
        if alg == 1
            res = add_big(u, v, b)
            println("Сумма: $(digits_to_str(res, b))\n")
        elseif alg == 2
            # Проверка условия u ≥ v для вычитания
            if length(u) < length(v) || (length(u) == length(v) && u < v)
                println("Ошибка: первое число должно быть ≥ второго\n")
            else
                res = sub_big(u, v, b)
                println("Разность: $(digits_to_str(res, b))\n")
            end
        elseif alg == 3
            res = mul_classic(u, v, b)
            println("Произведение (столбиком): $(digits_to_str(res, b))\n")
        elseif alg == 4
            res = mul_fast(u, v, b)
            println("Произведение (быстро): $(digits_to_str(res, b))\n")
        elseif alg == 5
            q, r = div_big(u, v, b)
            println("Частное: $(digits_to_str(q, b))")
            println("Остаток: $(digits_to_str(r, b))\n")
        end
    catch e
        # Ловим все ошибки ввода (недопустимые цифры и т.д.)
        println("Ошибка: $(sprint(showerror, e))\n")
    end
end
```

```
Лабораторная работа №8: Арифметика многократной точности
0 — Выход
1 — Сложение
2 — Вычитание
3 — Умножение столбиком
4 — Умножение быстрым столбиком
5 — Деление с остатком

julia> 1
1
Основание b (2-36): 6
Первое число (в системе 6): 4
Второе число (в системе 6): 2
Сумма: 10

Выберите алгоритм (0 для выхода): 2
Основание b (2-36): 29
Первое число (в системе 29): 23
Второе число (в системе 29): 17
Ошибка: первое число должно быть ≥ второго

Выберите алгоритм (0 для выхода): 2
Основание b (2-36): 30
Первое число (в системе 30): 29
Второе число (в системе 30): 10
Разность: 19

Выберите алгоритм (0 для выхода): 3
Основание b (2-36): 33
Первое число (в системе 33): 32
Второе число (в системе 33): 19
Произведение (столбиком): 3TI

Выберите алгоритм (0 для выхода): 4
Основание b (2-36): 10
Первое число (в системе 10): 2
Второе число (в системе 10): 8
Произведение (быстро): 1

Выберите алгоритм (0 для выхода): 5
Основание b (2-36): 16
Первое число (в системе 16): 15
Второе число (в системе 16): 12
Частное: 0
Остаток: 15

Выберите алгоритм (0 для выхода): 0
До свидания!
```