

Шаблон отчёта по лабораторной работе №8

**Дисциплина: Математические основы защиты информации и
информационной безопасности**

Миличевич Александра

Содержание

Цель работы	5
Арифметика в системах счисления: Сложение , Вычитание, Умножение и деление	5
1. Функция addition(num1, num2, base)	5
2. Функция subtraction(num1, num2, base)	7
3. Функция multiplication(num1, num2, base)	9
4. Функция multiplication_alternative(num1, num2, base)	11
5. Функция division(dividend, divisor, base)	13
Вывод	15

Список иллюстраций

1	addition	7
2	substruction	9
3	multiplication	11
4	multiplication alternative	12
5	division	14

Список таблиц

Цель работы

Арифметика в системах счисления: Сложение , Вычитание, Умножение и деление

Этот документ описывает реализацию алгоритмов сложения и вычитания неотрицательных целых чисел в заданной системе счисления.

1. Функция `addition(num1, num2, base)`

Эта функция выполняет сложение двух неотрицательных целых чисел в заданной системе счисления.

Описание:

- **Вход:**

- `num1 (str)`: Первое число в виде строки.
- `num2 (str)`: Второе число в виде строки.
- `base (int)`: Основание системы счисления.

- **Выход:**

- `list`: Список цифр, представляющий сумму двух чисел.

Как работает:

1. **Инициализация:**

- Определяется разрядность чисел `n` (длина `num1`, предполагается, что `num1` и `num2` имеют одинаковую длину).
- Переменная `carry` (перенос) устанавливается в 0.
- Создается пустой список `result` для хранения цифр результата.

2. Цикл по разрядам:

- Цикл `for` проходит по разрядам чисел от младшего к старшему (справа налево).
- На каждой итерации:
 - Вычисляется сумма цифр на текущем разряде и перенос:
`digit_sum = int(num1[i]) + int(num2[i]) + carry.`
 - Младшая цифра результата добавляется в список `result`:
`result.append(digit_sum % base).`
 - Вычисляется новый перенос: `carry = digit_sum // base.`

3. **Разворот результата:** Список `result` разворачивается, так как цифры добавлялись в обратном порядке.

4. **Возврат результата:** Возвращается список `result`, представляющий сумму чисел.

```

import math

# --- Алгоритм 1: Сложение неотрицательных целых чисел ---
def addition(num1, num2, base):
    """
    Выполняет сложение двух неотрицательных целых чисел в заданной системе счисления.

    Args:
        num1 (str): Первое число в виде строки.
        num2 (str): Второе число в виде строки.
        base (int): Основание системы счисления.

    Returns:
        list: Список цифр, представляющий сумму двух чисел.
    """
    n = len(num1) # Разрядность чисел (предполагаем, что num1 и num2 имеют одинаковую длину)
    carry = 0     # Перенос
    result = []   # Список для хранения цифр результата

    for i in range(n - 1, -1, -1): # Идем по разрядам от младшего к старшему
        digit_sum = int(num1[i]) + int(num2[i]) + carry # Складываем цифры и перенос
        result.append(digit_sum % base)                 # Добавляем младшую цифру результата
        carry = digit_sum // base                       # Вычисляем новый перенос

    result.reverse() # Разворачиваем результат, так как цифры добавлялись в обратном порядке.
    return result

```

Рис. 1: addition

2. Функция subtraction(num1, num2, base)

Эта функция выполняет вычитание двух неотрицательных целых чисел в заданной системе счисления. Предполагается, что $\text{num1} \geq \text{num2}$.

Описание:

- **Вход:**

- num1 (str): Уменьшаемое в виде строки.
- num2 (str): Вычитаемое в виде строки.
- base (int): Основание системы счисления.

- **Выход:**

- list: Список цифр, представляющий разность двух чисел.

Как работает:

1. Инициализация:

- Определяется разрядность чисел `n` (длина `num1`).
- Переменная `borrow` (заем) устанавливается в 0.
- Создается пустой список `result` для хранения цифр результата.

2. Цикл по разрядам:

- Цикл `for` проходит по разрядам чисел от младшего к старшему (справа налево).
- На каждой итерации:
 - Вычисляется разность цифр на текущем разряде и заем:


```
digit_diff = int(num1[i]) - int(num2[i]) + borrow.
```
 - Если `digit_diff` отрицателен:
 - * К `digit_diff` добавляется основание, чтобы получить неотрицательную цифру: `digit_diff += base.`
 - * Устанавливается заем в -1: `borrow = -1.`
 - Иначе (если `digit_diff` неотрицателен):
 - * Устанавливается заем в 0: `borrow = 0.`
 - Цифра результата добавляется в список `result`: `result.append(digit_diff).`

3. **Разворот результата:** Список `result` разворачивается.

4. **Возврат результата:** Возвращается список `result`, представляющий разность чисел.


```

# --- Алгоритм 2: Вычитание неотрицательных целых чисел ---
def subtraction(num1, num2, base):
    """
    Выполняет вычитание двух неотрицательных целых чисел в заданной системе счисления.
    Предполагается, что num1 >= num2.

    Args:
        num1 (str): Уменьшаемое в виде строки.
        num2 (str): Вычитаемое в виде строки.
        base (int): Основание системы счисления.

    Returns:
        list: Список цифр, представляющий разность двух чисел.
    """
    n = len(num1) # Разрядность чисел
    borrow = 0 # Заем
    result = [] # Список для хранения цифр результата

    for i in range(n - 1, -1, -1): # Идем по разрядам от младшего к старшему
        digit_diff = int(num1[i]) - int(num2[i]) + borrow # Вычисляем разность цифр и заем

        if digit_diff < 0: # Если результат отрицательный
            digit_diff += base # Добавляем основание, чтобы получить неотрицательную цифру
            borrow = -1 # Устанавливаем заем в -1
        else:
            borrow = 0 # Иначе, заем равен 0

        result.append(digit_diff) # Добавляем цифру результата

    result.reverse() # Разворачиваем результат
    return result

```

Рис. 2: subtraction

3. Функция multiplication(num1, num2, base)

Эта функция выполняет умножение двух неотрицательных целых чисел в заданной системе счисления.

Описание:

- **Вход:**

- num1 (str): Первое число в виде строки.
- num2 (str): Второе число в виде строки.
- base (int): Основание системы счисления.

- **Выход:**

- list: Список цифр, представляющий произведение двух чисел.

Как работает:

1. Инициализация:

- Определяется разрядность чисел n и m (длины `num1` и `num2` соответственно).
- Создается список `result` длиной $n + m$, заполненный нулями, для хранения результата.

2. Внешний цикл по разрядам второго числа:

- Цикл `for` проходит по разрядам `num2` от младшего к старшему (справа налево).
- Если текущая цифра `num2[j]` равна 0, то итерация пропускается.

3. Внутренний цикл по разрядам первого числа:

- Цикл `for` проходит по разрядам `num1` от младшего к старшему.
- На каждой итерации:
 - Вычисляется произведение цифр на текущих разрядах и добавляется к предыдущему результату и переносу: `product = int(num1[i]) * int(num2[j]) + result[i + j + 1] + carry`.
 - Младшая цифра произведения записывается в `result[i + j + 1]`: `result[i + j + 1] = product % base`.
 - Вычисляется новый перенос: `carry = product // base`.

4. Добавление переноса:

Оставшийся перенос `carry` добавляется к `result[j]`.

5. Удаление ведущих нулей:

Убираются ведущие нули из `result` (если есть).

6. Возврат результата:

Возвращается список `result`, представляющий произведение чисел.

```

# --- Алгоритм 3: Умножение неотрицательных целых чисел ---
def multiplication(num1, num2, base):
    """
    Выполняет умножение двух неотрицательных целых чисел в заданной системе счисления.

    Args:
        num1 (str): Первое число в виде строки.
        num2 (str): Второе число в виде строки.
        base (int): Основание системы счисления.

    Returns:
        list: Список цифр, представляющий произведение двух чисел.
    """
    n = len(num1) # Разрядность первого числа
    m = len(num2) # Разрядность второго числа
    result = [0] * (n + m) # Создаем список для хранения результата, заполненный нулями

    for j in range(m - 1, -1, -1): # Идем по разрядам второго числа от младшего к старшему
        if int(num2[j]) == 0: # Если текущая цифра равна 0, пропускаем
            continue

        carry = 0 # Перенос
        for i in range(n - 1, -1, -1): # Идем по разрядам первого числа от младшего к старшему
            product = int(num1[i]) * int(num2[j]) + result[i + j + 1] + carry # Вычисляем произведение и добавляем перенос
            result[i + j + 1] = product % base # Записываем младшую цифру в результат
            carry = product // base # Вычисляем новый перенос

        result[j] += carry # Добавляем оставшийся перенос

    # Убираем ведущие нули
    while len(result) > 1 and result[0] == 0:
        result.pop(0)

    return result

```

Рис. 3: multiplication

4. Функция multiplication_alternative(num1, num2, base)

Эта функция выполняет альтернативный алгоритм умножения двух неотрицательных целых чисел в заданной системе счисления.

Описание:

• Вход:

- num1 (str): Первое число в виде строки.
- num2 (str): Второе число в виде строки.
- base (int): Основание системы счисления.

• Выход:

- list: Список цифр, представляющий произведение двух чисел. #####

Как работает:

1. Инициализация:

- Определяется разрядность чисел n и m (длины num1 и num2 соответственно).

- Создается список `result` длиной $n + m + 2$, заполненный нулями, для хранения результата.
- Переменная `carry` устанавливается в 0.

2. Внешний цикл по суммам разрядов:

- Цикл `for` проходит по всем возможным суммам разрядов.
- На каждой итерации вычисляется `temp_sum` как сумма произведений соответствующих цифр из `num1` и `num2`.
- Текущая цифра результата вычисляется как $(temp_sum + carry) \% base$ и записывается в `result`.
- `carry` обновляется как $(temp_sum + carry) // base$.

3. Удаление ведущих нулей: Убираются ведущие нули из `result` (если есть).

4. Возврат результата: Возвращается список `result`, представляющий произведение чисел.

```
# --- Алгоритм 4: Умножение неотрицательных целых чисел (альтернативный) ---
def multiplication_alternative(num1, num2, base):
    """
    Выполняет умножение двух неотрицательных целых чисел в заданной системе счисления.

    Args:
        num1 (str): Первое число в виде строки.
        num2 (str): Второе число в виде строки.
        base (int): Основание системы счисления.

    Returns:
        list: Список цифр, представляющий произведение двух чисел.
    """
    n = len(num1)
    m = len(num2)
    result = [0] * (n + m + 2)
    carry = 0

    for s in range(m + n - 1, -1, -1):
        temp_sum = 0
        for i in range(s + 1):
            if n - i - 1 < 0 or m - s + i - 1 < 0 or n - i - 1 >= n or m - s + i - 1 >= m:
                continue
            temp_sum += (int(num1[n - i - 1]) * int(num2[m - s + i - 1]))

        result[m + n - s - 1] = (temp_sum + carry) % base
        carry = (temp_sum + carry) // base

    # Убираем ведущие нули
    while len(result) > 1 and result[0] == 0:
        result.pop(0)
    return result
```

Рис. 4: multiplication alternative

5. Функция `division(dividend, divisor, base)`

Эта функция выполняет деление неотрицательных целых чисел в заданной системе счисления.

Описание:

- **Вход:**

- `dividend (str)`: Делимое в виде строки.
- `divisor (str)`: Делитель в виде строки.
- `base (int)`: Основание системы счисления.

- **Выход:**

- `tuple`: Кортеж, содержащий частное (`list`) и остаток (`str`).

Как работает:

1. Инициализация:

- Определяется длина делимого `dividend_len` и делителя `divisor_len`.
- Проверяется, что делитель не равен нулю.
- Проверяется, что делимое больше или равно делителю.
- Создается список `quotient` (частное) и присваивается `remainder` значение `dividend`.

2. Предварительное деление:

- Выполняется проверка, на случай, если делимое больше делителя на целую степень `base`.

3. Основной цикл деления:

- Цикл проходит по разрядам делимого (справа налево).
- На каждой итерации:

- Проверяется, что текущий разряд делимого входит в границы остатка.
- Определяется начальное значение для цифры частного.
- Цикл while уточняет значение цифры частного.
- Обновляется остаток с учетом новой цифры частного.

4. **Возврат результата:** Возвращается кортеж, содержащий частное (quotient) и остаток (remainder).

```
def division(dividend, divisor, base):
    """
    Выполняет деление неотрицательных целых чисел в заданной системе счисления.

    Args:
        dividend (str): Делимое в виде строки.
        divisor (str): Делитель в виде строки.
        base (int): Основание системы счисления.

    Returns:
        tuple: Кортеж, содержащий частное (list) и остаток (str).
    """
    dividend_len = len(dividend) # Длина делимого
    divisor_len = len(divisor) # Длина делителя

    # Проверим, что делитель не ноль
    if int(divisor) == 0:
        return [], "Деление на ноль!"

    # Проверим, что делимое больше или равно делителю
    if int(dividend) < int(divisor):
        return [0], dividend

    # Инициализация частного и остатка
    quotient = [0] * (dividend_len - divisor_len + 1)
    remainder = dividend # Изначально остаток - это делимое

    # Условие деления делимого на делитель
    while int(remainder) >= int(divisor) * (base ** (dividend_len - divisor_len)):
        quotient[dividend_len - divisor_len] += 1
        remainder = int(remainder) - int(divisor) * (base ** (dividend_len - divisor_len))
    remainder = str(remainder) # Конвертируем обратно в строку

    # Основной цикл деления
    for i in range(dividend_len - 1, divisor_len - 2, -1):
        if i >= len(remainder):
            continue

        if i >= len(remainder):
            continue

        if int(remainder[i]) > int(divisor[divisor_len - 1]):
            quotient[i - divisor_len] = base - 1
        else:
            if i - 1 < 0:
                if int(divisor[divisor_len - 1]) != 0:
                    quotient[i - divisor_len] = math.floor(int(remainder[i]) / int(divisor[divisor_len - 1]))
                else:
                    quotient[i - divisor_len] = 0
            else:
                if int(divisor[divisor_len - 1]) != 0:
                    quotient[i - divisor_len] = math.floor((int(remainder[i]) * base + int(remainder[i - 1])) / int(divisor[divisor_len - 1]))
                else:
                    quotient[i - divisor_len] = 0

    # Уточнение частного
    if i - 2 < 0:
        if i - 1 < 0:
            continue
```

Рис. 5: division

Вывод

Эта лабораторная описывает функции `addition`, `subtraction`, `multiplication`, `multiplication_alternative`, и `division`, которые выполняют сложение, вычитание, умножение и деление неотрицательных целых чисел в заданной системе счисления. Эти функции используют стандартные алгоритмы для выполнения этих операций, работая с числами, представленными в виде строк цифр.