Министерство науки и высшего образования Российской Федерации
Федеральное государственное
бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Фундаментальные Науки»

КАФЕДРА ФН-12 «Математическое моделирование»

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ НА ТЕМУ:

Длинная арифметика

Студент: _____ Мациевский И. М.

дата, подпись Ф.И.О.

Преподаватель: _____ Волкова Л. Л.

дата, подпись Ф.И.О.

Москва, 2023

Содержание

Введение

Цель лабораторной работы: составить структуру данных для хранения целых чисел с повышением ёмкости хранения по сравнению с базовым типом данных.

Для достижения поставленной цели требуется решить следующие **задачи**.

1. Реализовать 6 операторов сравнения.
2. Реализовать оператор присваивания.
3. Реализовать операторы сложения и вычитания двух структур длинных чисел в четверичной системе счисления.
4. Реализовать оператор четверичного сдвига вправо и влево.
5. Реализовать оператор остатка от деления структуры длинных чисел.

1 Аналитическая часть

Длинная арифметика — это методика работы с числами, превышающими максимальные значения, предоставляемые базовыми типами данных языка программирования. Часто используется в задачах, требующих работы с очень большими числами, такие задачи часто встречаются, например, в науке.

Для хранения длинных чисел в лабораторной работе реализован класс $QuaternaryNumber$, объектом класса является массив цифр от 0 до 3 и дополнительный флаг $isNegative$, принимающий значение $true$ в случае, когда число положительно и значение $false$, когда число отрицательно.

Для работы с объектами класса перегружены операторы сложения, вычитания, остатка от деления, 6 операторов сравнения, оператор присваивания, реализованы операторы четверичного сдвига вправо и влево.

2 Конструкторская часть

Операторы сравнения

1. Оператор '==' сначала сравнивает знаки двух чисел, если знаки одинаковы, то сравнивает два числа поразрядно, возвращает $true$ в случае равенства чисел, иначе $false$.

2. Оператор '!=' сравнивает два числа сначала по их знакам и если знаки различны, сравнивает числа поразрядно, возвращает $false$ в случае равенства чисел, иначе $true$.

3. Оператор '<' сначала сравнивает знаки двух чисел, если знаки одинаковы, то сравнивает два числа поразрядно, возвращает $false$, если первое (по левую сторону от оператора) число больше второго или равно ему, иначе $true$.

4. Оператор '>' сначала сравнивает знаки двух чисел, если знаки одинаковы, то сравнивает два числа поразрядно, возвращает $true$, если первое (по левую сторону от оператора) число больше второго, иначе $false$.

5. Оператор '<=' сначала сравнивает знаки двух чисел, если знаки одинаковы, то сравнивает два числа поразрядно, возвращает $false$, если первое (по левую сторону от оператора) число больше второго, иначе $true$.

6. Оператор '>=' сначала сравнивает знаки двух чисел, если знаки одинаковы, то сравнивает два числа поразрядно, возвращает $true$, если первое (по левую сторону от оператора) число больше второго или равно ему, иначе $false$.

7. Оператор '< <=' сравнивает поразрядно модули двух чисел, возвращает $true$, если первое (по левую сторону от оператора) число по модулю меньше второго, иначе $false$.

8. Оператор '> >=' сравнивает поразрядно модули двух чисел, возвращает $true$, если первое (по левую сторону от оператора) число по модулю больше второго, иначе $false$.

Остальные операторы

1. Оператор '> >' сдвигает цифры числа вправо на заданное количество разрядов, освободившиеся разряды заполняет нулями.

2. Оператор '< <' сдвигает цифры числа влево на заданное количество разрядов, освободившиеся разряды заполняет нулями.

3. Операторы '+' и '-' определяется знак суммы/разности в зависимости от знаков операндов, после этого поразрядно складываются/вычитаются модули двух чисел,

используется дополнительная переменная для операции "заема"из старшего разряда и "добавления"в старший разряд.

4. Оператор '%' использует операцию вычитания для нахождения остатка от деления.

5. Оператор '=' копирует значения цифр и флага знака из одного объекта в другой.

Вспомогательные функции и методы

1. *isEqual* сначала сравнивает знаки двух чисел, если знаки одинаковы, то сравнивает два числа поразрядно, возвращает $true$ в случае равенства чисел, иначе $false$.

2. *isEqual_abs* сравнивает модули двух чисел поразрядно, возвращает $true$ в случае равенства чисел по модулю, иначе $false$.

3. *isGreaterThan_abs* сравнивает модули двух чисел поразрядно, возвращает $true$, если первое число по модулю больше второго, иначе $false$.

4. *isGreaterThan* сначала сравнивает знаки двух чисел, если знаки одинаковы, сравнивает модули двух чисел поразрядно, возвращает $true$, если первое число больше второго, иначе $false$.

5. *isLessThan_abs* сравнивает модули двух чисел поразрядно, возвращает $true$, если первое число по модулю меньше второго, иначе $false$.

6. *isLessThan* сначала сравнивает знаки двух чисел, если знаки одинаковы, сравнивает модули двух чисел поразрядно, возвращает $true$, если первое число меньше второго, иначе $false$.

7. *print*() выводит число на экран, учитывая его знак и форматирование.

3 Технологическая часть

Для реализации выбран язык C++. На листинге 1 представлена реализация программы (Реализация 1)

Листинг 1 – Исходный код

```
#include <iostream>
#include <cmath>
#include <iomanip>
using namespace std;

const int MAX_DIGITS = 10;

class QuaternaryNumber {
private:
    int digits[MAX_DIGITS];
    bool isNegative;

public:
    QuaternaryNumber() {
        for (int i = 0; i < MAX_DIGITS; ++i) {
            digits[i] = 0;
        }
        isNegative = false;
    }

    QuaternaryNumber(int value) {
        isNegative = value < 0;
        value = abs(value);

        for (int i = 0; i < MAX_DIGITS; ++i) {
            digits[i] = value % 4;
            value /= 4;
        }
    }

    // Операторы сравнения по модулю
    bool operator==(const QuaternaryNumber& other) const {
        return isEqual(other);
    }

    bool operator!=(const QuaternaryNumber& other) const {
```

```
    return !isEqual(other);
}

bool operator<<=(const QuaternaryNumber& other) const {
    return isLessThan_abs(other); //сравнение по модулю
}

bool operator>>=(const QuaternaryNumber& other) const {
    return isGreaterThan_abs(other); //сравнение по модулю
}

bool operator<(const QuaternaryNumber& other) const {
    return isLessThan(other);
}

bool operator>(const QuaternaryNumber& other) const {
    return isGreaterThan(other);
}

bool operator<=(const QuaternaryNumber& other) const {
    return !isGreaterThan(other);
}

bool operator>=(const QuaternaryNumber& other) const {
    return !isLessThan(other);
}

// Оператор присваивания
QuaternaryNumber& operator=(const QuaternaryNumber& other) {
    if (this != &other) {
        for (int i = 0; i < MAX_DIGITS; ++i) {
            digits[i] = other.digits[i];
        }
        isNegative = other.isNegative;
    }
    return *this;
}

// Операции сложения и вычитания
QuaternaryNumber operator+(const QuaternaryNumber& other) const {
QuaternaryNumber result;
```

```
int carry = 0;

// Определяем знак результата
if (isNegative && !other.isNegative) {
    if (*this >>= other) {
        result.isNegative = true;
    } else {
        result.isNegative = false;
    }
} else if (!isNegative && other.isNegative) {
    if (*this >>= other) {
        result.isNegative = false;
    } else {
        result.isNegative = true;
    }
} else if (isNegative && other.isNegative) {
    result.isNegative = true;
} else {
    result.isNegative = false;
}
bool isZero = true;
// Складываем числа по модулю
for (int i = 0; i < MAX_DIGITS; ++i) {
    int sum;

    if (isNegative && !other.isNegative) {
        if (*this >>= other) {
            sum = digits[i] - other.digits[i] - carry;
        }
        else {
        sum = other.digits[i] - digits[i] - carry;
        }
    }
    else if (!isNegative && other.isNegative) {
        if (*this >>= other) {
            sum = digits[i] - other.digits[i] - carry;
        }
        else {
        sum = other.digits[i] - digits[i] - carry;
        }
    }
```

```
        else if (isNegative && other.isNegative) {
            sum = other.digits[i] + digits[i] + carry;
        }
        else {
            sum = digits[i] + other.digits[i] + carry;
        }

        if (sum < 0) {
            sum += 4;
            carry = 1;
        }
        else if (sum > 3) {
            carry = 1;
        }
        else {
            carry = 0;
        }
        if (sum % 4 != 0) {
            isZero = false;
        }
        result.digits[i] = sum % 4;
    }
    if (carry > 0) {
        for (int i = MAX_DIGITS - 1; i >= 0; --i) {
            if (result.digits[i] != 0) {
                result.digits[i - 1] += carry;
                carry = 0;
                break;
            }
        }
    }
    if (isZero) {
        result.isNegative = false;
    }
    return result;
}


    QuaternaryNumber operator-(const QuaternaryNumber& other) const {
     QuaternaryNumber result;
     int carry = 0;
```

```
// Определяем знак результата
if (isNegative && other.isNegative) {
    if (*this >>= other) {
        result.isNegative = true;
    } else {
        result.isNegative = false;
    }
} else if (!isNegative && !other.isNegative) {
    if (*this >>= other) {
        result.isNegative = false;
    } else {
        result.isNegative = true;
    }
} else if (isNegative && !other.isNegative) {
    result.isNegative = true;
} else {
    result.isNegative = false;
}
bool isZero = true;
// Складываем числа по модулю
for (int i = 0; i < MAX_DIGITS; ++i) {
    int sum;

    if (isNegative && other.isNegative) {
        if (*this >>= other) {
            sum = digits[i] - other.digits[i] - carry;
        }
        else {
        sum = other.digits[i] - digits[i] - carry;
        }
    }
    else if (!isNegative && !other.isNegative) {
        if (*this >>= other) {
            sum = digits[i] - other.digits[i] - carry;
        }
        else {
        sum = other.digits[i] - digits[i] - carry;
        }
    }
    else if (isNegative && !other.isNegative) {
```

```
            sum = other.digits[i] + digits[i] + carry;
        }
        else {
            sum = digits[i] + other.digits[i] + carry;
        }

        if (sum < 0) {
            sum += 4;
            carry = 1;
        }
        else if (sum > 3) {
            carry = 1;
        }
        else {
            carry = 0;
        }
        if (sum % 4 != 0) {
            isZero = false;
        }
        result.digits[i] = sum % 4;
    }
    if (carry > 0) {
        for (int i = MAX_DIGITS - 1; i >= 0; --i) {
            if (result.digits[i] != 0) {
                result.digits[i - 1] += carry;
                carry = 0;
                break;
            }
        }
    }
   if (isZero) {
       result.isNegative = false;
   }
    return result;
}


   // Другие операции
   QuaternaryNumber operator<<(int shift) const {
       QuaternaryNumber result = *this;
       for (int i = 0; i < shift; ++i) {
```

```
            for (int j = MAX_DIGITS - 1; j > 0; --j) {
                result.digits[j] = result.digits[j - 1];
            }
            result.digits[0] = 0;
        }
        return result;
    }

    QuaternaryNumber operator>>(int shift) const {
        QuaternaryNumber result = *this;
        for (int i = 0; i < shift; ++i) {
            for (int j = 0; j < MAX_DIGITS - 1; ++j) {
                result.digits[j] = result.digits[j + 1];
            }
            result.digits[MAX_DIGITS - 1] = 0;
        }
        return result;
    }

    QuaternaryNumber operator%(const QuaternaryNumber& divisor) const {
        QuaternaryNumber dividend = *this;
        if (dividend.isNegative == divisor.isNegative) {
            while (dividend >= divisor) {
                dividend = dividend - divisor;
            }
        }
        else if (dividend.isNegative) {
            while (dividend.isNegative) {
                dividend = dividend + divisor;
            }
        }
        else {
            while (!dividend.isNegative) {
                dividend = dividend + divisor;
            }
        }
        return dividend;
    }

    // Вспомогательные функции
    bool isEqual(const QuaternaryNumber& other) const {
```

```
    if (isNegative != other.isNegative) {
        return false;
    }
    for (int i = 0; i < MAX_DIGITS; ++i) {
        if (digits[i] != other.digits[i]) {
            return false;
        }
    }
    return true;
}

bool isEqual_abs(const QuaternaryNumber& other) const {
        for (int i = 0; i < MAX_DIGITS; ++i) {
            if (digits[i] != other.digits[i]) {
                return false;
            }
        }
        return true;
    }

bool isGreaterThan_abs(const QuaternaryNumber& other) const {
        for (int i = MAX_DIGITS - 1; i >= 0; --i) {
            if (digits[i] > other.digits[i]) {
                return true;
            } else if (digits[i] < other.digits[i]) {
                return false;
            }
        }
        return false;
    }

bool isGreaterThan(const QuaternaryNumber& other) const {
    if (isNegative and !other.isNegative) {
        return false;
    }
    else if (!isNegative and other.isNegative) {
        return true;
    }
    else if (!isNegative and !other.isNegative) {
        for (int i = MAX_DIGITS - 1; i >= 0; --i) {
            if (digits[i] > other.digits[i]) {
```

```
                return true;
            } else if (digits[i] < other.digits[i]) {
                return false;
            }
        }
        return false;
    }
    else {
        for (int i = MAX_DIGITS - 1; i >= 0; --i) {
            if (digits[i] > other.digits[i]) {
                return false;
            } else if (digits[i] < other.digits[i]) {
                return true;
            }
        }
        return true;
    }
}

bool isLessThan_abs(const QuaternaryNumber& other) const {
    return !isEqual_abs(other) && !isGreaterThan_abs(other);
}

bool isLessThan(const QuaternaryNumber& other) const {
    return !isEqual(other) && !isGreaterThan(other);
}

// Вывод числа
void print() const {
    if (isNegative) {
        cout << '-';
    }

    for (int i = MAX_DIGITS - 1; i >= 0; --i) {
        cout << digits[i];
    }

    cout << endl;
}
};
```

```
int main() {
    int x, y;
    cout << "Введите первое число" << endl;
    cin >> x;
    cout << "Введите второе число" << endl;
    cin >> y;
    QuaternaryNumber a(x);
    QuaternaryNumber b(y);

    QuaternaryNumber sum = a + b;
    QuaternaryNumber diff = a - b;
    QuaternaryNumber leftShift = a << 2;
    QuaternaryNumber rightShift = a >> 1;
    QuaternaryNumber modulo = a % b;

    // Проверка операторов сравнения
    bool isEqual = (a == b);
    bool isNotEqual = (a != b);
    bool isLessThan = (a < b);
    bool isGreaterThan = (a > b);
    bool isLessOrEqual = (a <= b);
    bool isGreaterOrEqual = (a >= b);

    // Проверка оператора присваивания
    QuaternaryNumber assigned = a;

    cout << "a: ";
    a.print();

    cout << "b: ";
    b.print();

    cout << "a + b: ";
    sum.print();

    cout << "a - b: ";
    diff.print();

    cout << "a << 2: ";
    leftShift.print();
```

```
    cout << "a >> 1: ";
    rightShift.print();

    cout << "a % b: ";
    modulo.print();

    cout << "a == b: " << boolalpha << isEqual << endl;
    cout << "a != b: " << boolalpha << isNotEqual << endl;
    cout << "a < b: " << boolalpha << isLessThan << endl;
    cout << "a > b: " << boolalpha << isGreaterThan << endl;
    cout << "a <= b: " << boolalpha << isLessOrEqual << endl;
    cout << "a >= b: " << boolalpha << isGreaterOrEqual << endl;

    cout << "Assigned: ";
    assigned.print();

    return 0;
}
```

Примеры работы. На рисунках 1—8 представлены примеры работы программы для расчета выражений в длинной арифметике.

1. Входные файлы: два положительных числа: Результат приведён на рис. 1.

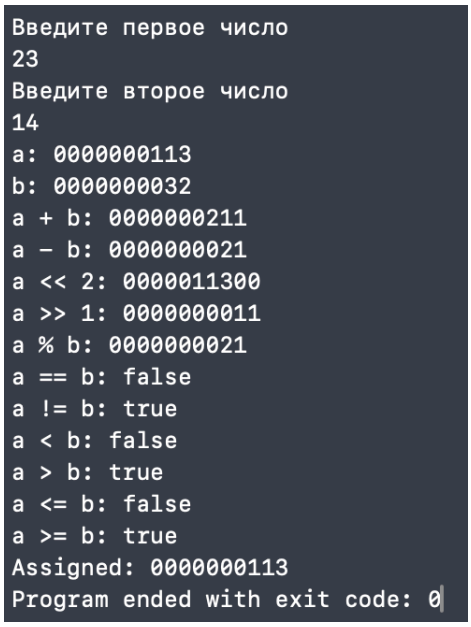

Рис. 1 – Пример работы 1

2. Входные файлы: первое число положительно, второе отрицательно. Результат приведён на рис. 2.

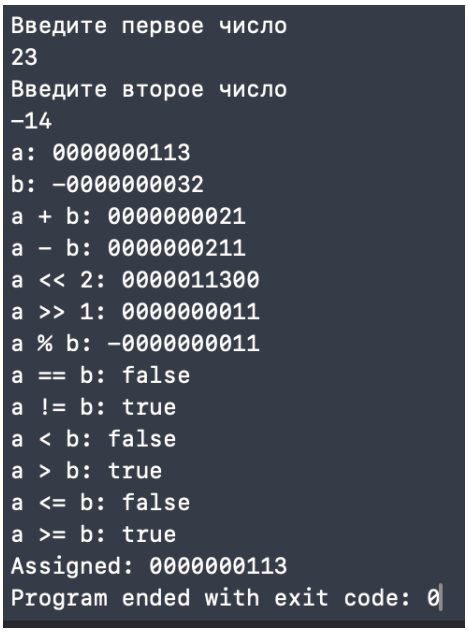

Рис. 2 – Пример работы 2

3. Входные файлы: два отрицательных числа. Результат приведён на рис. 3.

```
Введите первое число
-23
Введите второе число
-14
a: -0000000113
b: -0000000032
a + b: -0000000211
a - b: -0000000021
a << 2: -0000011300
a >> 1: -0000000011
a % b: -0000000113
a == b: false
a != b: true
a < b: true
a > b: false
a <= b: true
a >= b: false
Assigned: -0000000113
Program ended with exit code: 0
```

Рис. 3 – Пример работы 3

4. Входные файлы: первое число отрицательно, второе положительно. Результат приведён на рис. 4.

```
Введите первое число
-23
Введите второе число
14
a: -0000000113
b: 0000000032
a + b: -0000000021
a - b: -0000000211
a << 2: -0000011300
a >> 1: -0000000011
a % b: 0000000011
a == b: false
a != b: true
a < b: true
a > b: false
a <= b: true
a >= b: false
Assigned: -0000000113
Program ended with exit code: 0
```

Рис. 4 – Пример работы 4

5. Входные файлы: числа, равные по модулю и противоположные по знаку. Результат приведён на рис. 5.

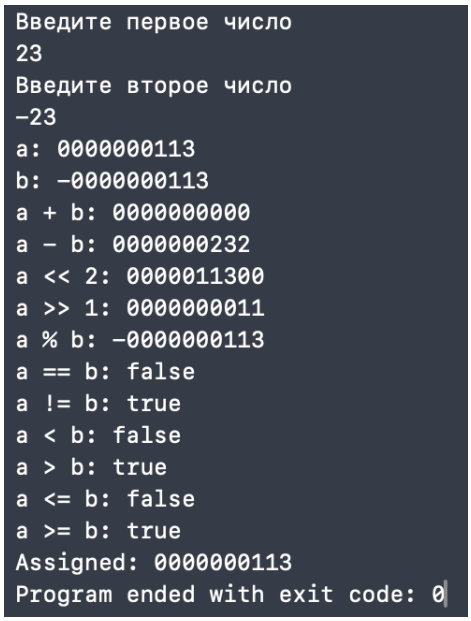

Рис. 5 – Пример работы 5

6. Входные файлы: равные числа. Результат приведён на рис. 6.

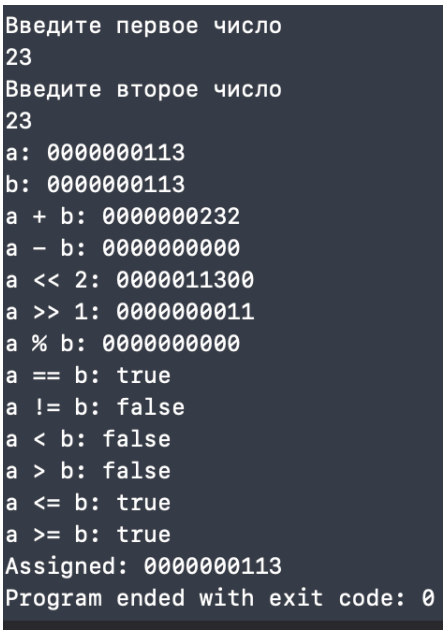

Рис. 6 – Пример работы 6

7. Входные файлы: первое число нуль, второе число положительно. Результат приведён на рис. 7.

```
Введите первое число
0
Введите второе число
23
a: 0000000000
b: 0000000113
a + b: 0000000113
a - b: -0000000113
a << 2: 0000000000
a >> 1: 0000000000
a % b: 0000000000
a == b: false
a != b: true
a < b: true
a > b: false
a <= b: true
a >= b: false
Assigned: 0000000000
Program ended with exit code: 0
```

Рис. 7 – Пример работы 7

8. Входные файлы: первое число нуль, второе число отрицательно: Результат приведён на рис. 8.

```
Введите первое число
0
Введите второе число
-23
a: 0000000000
b: -0000000113
a + b: -0000000113
a - b: 0000000113
a << 2: 0000000000
a >> 1: 0000000000
a % b: -0000000113
a == b: false
a != b: true
a < b: false
a > b: true
a <= b: false
a >= b: true
Assigned: 0000000000
Program ended with exit code: 0
```

Рис. 8 – Пример работы 8

Заключение

Цель достигнута: разработана структура данных для хранения целых чисел с хранением 10 разрядов. В результате выполнения лабораторной работы были выполнены все задачи.

1. Реализовано 6 операторов сравнения.
2. Реализован оператор присваивания.
3. Реализованы операторы сложения и вычитания двух структур длинных чисел в четверичной системе счисления.
4. Реализован оператор четверичного сдвига вправо и влево.
5. Реализован оператор остатка от деления структуры длинных чисел.

Список литературы

1. Неспирный В.Н., Длинная арифметика: научная статья 2010. – 13 с.