Лабораторная работа №1	10	2023
Представление чисел	Лихтар Анна Викторовна	

Инструментарий и требования к работе: Необходимо написать программу, которая позволяет выполнять арифметические действия с дробными числами в форматах с фиксированной и плавающей точкой. Программа должна использовать только целочисленные вычисления и типы данных. Используемый язык программирования – C++, Версия – C++17.

Ссылка на репозиторий: https://github.com/skkv-mkn/mkn-comp-arch-2023-fixed-floating-likhhtar

Результат работы на тестовых данных:

```
Проверка отчёта - check_pdf: False
Округление к 0 (0) - toward_zero: True
Округление к ближайшему чётному (1) - toward_nearest_even: True
Округление к +inf (2) - toward_pinf: True
Округление к -inf (3) - toward_ninf: True
Тест на вывод - print: Фиксированная и плавающая точка
Тест с + (фиксированная точка) - fixed_op: True
Тест с * (плавающая точка) - floating_op: False
Тест с * (плавающая точка с половинной точностью) - half_op: False
Тест на специальные случаи с плавающей точкой - floating_special: True
```

Входные аргументы	Результат
16.12 0 0x17360	23.210
8.8 1 0xdc9f + 0xd736	-76.168
f 0 0xB9CD542	0x1.39aa84p-104

Числа с фиксированной точкой

На вход даны данные в формате A.B, где A и B представляют собой целое и дробное количество битов соответственно. Для анализа и обработки этого формата данных были реализованы две функции:

1. Функция splitAB разделяет строку формата A.B на два числа A и B, определяя количество битов в целой и дробной частях соответственно.

```
bool splitAB(const std::string& input, size_t& A, size_t& B) {
    size_t dotPos = input.find('.');
    if (dotPos != std::string::npos) {
        try {
            A = std::stoi(input.substr(0, dotPos));
            B = std::stoi(input.substr(dotPos + 1));
            return true;
        } catch (const std::invalid_argument& e) {
            return false;
        }
    } else {
        return false;
    }
}
```

2. Функция hexToBinary преобразует числа, представленные в шестнадцатеричной системе счисления, в их двоичное представление.

```
std::string hexDigitToBinary(char hexDigit) {
    switch (hexDigit) {
        case '0': return "0000";
        case '1': return "0001";
        case '2': return "0010";
        case '3': return "0011";
        case '4': return "0100";
        case '5': return "0101";
        case '6': return "0110";
        case '7': return "0111";
        case '8': return "1000";
        case '9': return "1001";
        case 'A': case 'a': return "1010";
        case 'B': case 'b': return "1011";
        case 'C': case 'c': return "1100";
        case 'D': case 'd': return "1101";
        case 'E': case 'e': return "1110";
        case 'F': case 'f': return "1111";
        default: return "";
    }
}
std::string hexToBinary(std::string hexValue) {
    if (hexValue.substr(0, 2) == "0x") {
        hexValue = hexValue.substr(2);
    }
    std::string binaryValue;
    for (char hexDigit : hexValue) {
        std::string binaryDigit = hexDigitToBinary(hexDigit);
```

```
            binaryValue += binaryDigit;
        }
        return binaryValue;
    }
```

После обработки входных данных выполняется анализ на предмет наличия операции. Если обнаружена операция, то выполняются математические операции (сложение, вычитание, умножение или деление) в зависимости от заданной операции.

Хранение чисел

Изначально числа представлены в виде двоичных строк. Для выполнения операций они преобразуются в объекты типа bitset. Результат операции сохраняется в виде двоичной строки. Функция toFormat возвращает знак числа, целую часть в формате long long и дробную часть в виде строки.

Сложение

Вызывается функция, на вход которой подаются две строки, который представляют собой числа, заданные на входе, в двоичной системе счисления, а также размер целой и дробной частей соответственно, а на выход – результат сложения.

```
std::string sumAB(std::string first, std::string second, size_t x, size_t y) {
    first = negativeAdding(first, x, y, 32);
    second = negativeAdding(second, x, y, 32);

    std::bitset<32> firstNum(first);
    std::bitset<32> secondNum(second);

    std::bitset<32> sum = firstNum.to_ulong() + secondNum.to_ulong();
    std::string result = sum.to_string();
    result = result.substr(32 - x - y,  x + y);

    return result;
}
```

Для обеспечения корректного перевода, строки проверяются на знак числа. Если первый бит строки длины A+B равен 1, то число считается отрицательным, и к нему добавляются единицы для сохранения знака.

```
std::string negativeAdding (std::string first, size_t x, size_t y, size_t t) {
    if(first.length() == x + y && first[0] == '1') {
        for(size_t i = 0; i < t - x - y; i++) {
            first = "1" + first;
        }
    }
```

```
    return first;
}
```

Затем бинарные строки преобразуются в объекты bitset и складываются, результат хранится в последних A+B битах.
После того, как программа получила результат сложения, вызывается функция, принимающая на вход результат сложения, тип округления и количество битов в целой и дробной частях, и выдает знак сложения и целую и дробные части после округления.

```
std::pair<bool, std::pair<long long, std::string>> toFormat(std::string
result, std::string format, size_t x, size_t y) {
    std::pair<std::string, std::string> parts = splitNumber(result, x, y);
    long long intPart = binaryToDecimal(parts.first, x, true);
    bool sign = true;
    if (intPart < 0) {
        sign = false;
    }
    std::string fracPart;
    fracPart =
vectorToString(fractionalPart(intToVector(binaryToDecimal(parts.second, y,
false)), y));

    if (!sign) {
        std::vector<long long> additional = {1};
        for (size_t i = 0; i < y; i++) {
            additional.push_back(0);
        }
        std::vector<long long> frac = subtractVectors(additional,
stringToVector(fracPart));
        if(frac.size() == y) {
            fracPart = vectorToString(frac);
            intPart++;
        }
    }

    return {sign, rounding(intPart, fracPart, format, y, 3, sign)};
}
```

Сначала нужно разбить строку на две подстроки, одна будет означать целую часть, вторая – дробную.

```
std::pair<std::string, std::string> splitNumber(std::string& str, size_t A,
size_t B) {
    size_t len = str.length();
    if (len < A + B) {
        str.insert(0, A + B - len, '0');
    }
    str = str.substr(str.length() - A - B, A + B);
    return std::make_pair(str.substr(0, A), str.substr(A, B));
}
```

Затем необходимо перевести целую часть в десятичное число. Для этого вызовем следующую функцию. На ее вход подается бинарная строка целой части, количество битов целой части, а также значение bool, которое определяет, может ли получится отрицательный ответ. И если такое возможно, то будем смотреть на первый бит строки (если ее длина равна x), и отнимать от полученного значения (binary[0] – ‘0’) * 2^{x-1}.

```
long long binaryToDecimal(const std::string& binary, size_t x, bool integer) {
    long long decimal = 0;
    long long power = 1;
    for (size_t i = binary.length(); i > 0; i--) {
        long long t = power * (binary[i - 1] - '0');
        power *= 2;
        if(i + x - 1 == binary.length() && integer) {
            t *= -1;
        }
        decimal += t;
    }
    return decimal;
}
```

Вернемся к пониманию функции toFormat. Будем хранить знак целой части в bool sign. Далее найдем десятичную строчную запись дробной части с использованием функции fractionalPart. Об этой функции поговорим позже. Если целая часть была отрицательной, то выполняется коррекция для того, чтобы число было равно сложению целой и дробной частей. Для этого было использовано вычитание векторов и прибавление 1 к целой части.
Функция возвращает знак и результат округления. Об округлении позже.

Вычитание

Ничем принципиально не отличается от сложения. Только вместо функции сложения вызывается функция вычитания.

```
std::string diffAB(std::string first, std::string second, size_t x, size_t y)
{
    first = negativeAdding(first, x, y, 32);
    second = negativeAdding(second, x, y, 32);

    std::bitset<32> firstNum(first);
    std::bitset<32> secondNum(second);

    std::bitset<32> diff = firstNum.to_ulong() - secondNum.to_ulong();
    std::string result = diff.to_string();
    result = result.substr(32 - x - y, 32);

    return result;
}
```

Произведение

Вызывается функция для умножения двух строк:

```
std::string multAB(std::string first, std::string second, size_t x, size_t y)
{
    first = negativeAdding(first, x, y, 64);
    second = negativeAdding(second, x, y, 64);

    std::bitset<64> firstNum(first);
    std::bitset<64> secondNum(second);
    std::bitset<64> product = static_cast<unsigned long
long>(firstNum.to_ullong() * secondNum.to_ullong());
    std::string result = product.to_string();
    result = result.substr(64 - x - y - y, x + y + y);

    return result;
}
```

Основное отличие от сложения и вычитания заключается в том, что размер bitset теперь установлен на 64, так как произведение может не уместиться в 32 бита. Кроме того, на этом этапе функция возвращает не последние A+B битов, а последние A+B+B, так как при умножении произошел сдвиг на B бит влево. В следующей функции число сначала округлится до A+B битов, а затем до A+3 битов. Эта функция получает результат умножения, тип округления и размеры целой и дробной частей.

```
std::pair<bool, std::pair<long long, std::string>> toMultFormat(std::string
result, std::string format, size_t x, size_t y) {
    std::pair<bool, std::pair<long long, std::string>> firstPart =
toFormat(result, format, x + y, y, 0);
    result = decimalToBinary(firstPart.second.first, x + y);
    return toFormat(result, format, x, y, 3);
}
```

В первой части число разбивается на «целую» и «дробную» части. В целой части хранится число, состоящее из правильных целой и дробной частей числа, а в дробной – знаки после запятой, начинающиеся с B+1 бита. Далее, так же, как и в сложении/вычитании, происходит округление, но в этот раз до 0-го символа дробной части. Таким образом, округление после операции произведения сделано. Начинается вторая часть. Полученная «целая» часть переводится в битовую строку, а затем вызывается функция toFormat. Таким образом, второе округление было выполнено.

Деление

Вызывается функция для деления двух строк.

```
std::string quoAB(std::string first, std::string second, size_t x, size_t y) {
    for (size_t i = 0; i < y; i++) {
        first += "0";
    }
    long long fir = binaryToDecimal(first, x + y + y, true);
    long long sec = binaryToDecimal(second, x + y, true);
    bool sign = true;
    if ((fir < 0 && sec > 0) || (fir > 0 && sec < 0)) {
        sign = false;
    }
    fir = abs(fir); sec = abs(sec);
    first = decimalToBinary(fir, x + y);
    second = decimalToBinary(sec, x + y);
    std::bitset<64> firstNum(first);
    std::bitset<64> secondNum(second);

    if(secondNum.to_ullong() == 0) return "false";
    std::bitset<64> product = static_cast<unsigned long>(firstNum.to_ullong()
/ secondNum.to_ullong());
    if (!sign) {
        product.flip();
    }
    std::string result = product.to_string();
    result = result.substr(64 - x - y, x + y);

    return result;
}
```

Первоначально в функции добавляются B битов в конец первой строки для компенсации сдвига вправо при делении. Затем бинарные строки преобразуются в десятичные числа, и определяется знак результата в зависимости от знаков делимого и делителя. Модули чисел снова преобразуются в двоичные строки, затем обе строки преобразуются в объекты bitset. В случае деления на нуль, функция возвращает «false». Далее выполняется деление, и, если необходимо, инвертируются биты результата. Результат преобразуется в строку и обрезается до нужной длины.

Затем вызывается функция toFormat от результата. Таким образом, произойдет округление, и на выход поступят знак, целая и дробная части ответа.

Операция отсутствует

Если операция не задана, то вызывается функция toFormat от входного числа.

Округление

Реализовано округление в десятичной системе счисления.

```
std::pair<long long, std::string> rounding(long long intPart, std::string
input, std::string format, size_t x, size_t y, bool sign) {
    std::string result = input;
    if(format == "0") {
        result = result.substr(0, y);
    }
    else if (format == "1") {
        return nearestEven(intPart, result, x, y);
    }
    else if (format == "2") {
        return towardInfinity(intPart, result, x, y, sign);
    }
    else if (format == "3") {
        return towardNegInfinity(intPart, result, x, y, sign);
    }
    return std::make_pair(intPart, result);
}
```

1. Если округление к 0, то нужно просто убрать все биты, начиная с y.
2. Если округление к ближайшему четному, то вызывается следующая функция.

```
std::pair<long long, std::string> nearestEven(long long intPart,
std::string result, size_t x, size_t y) {
    long long zero = 0;
    if(result[y] < '5'){
        zero = -1;
    }
    else if(result[y] > '5') {
        zero = 1;
    }
    else {
        for(size_t i = y + 1; i < x; i++) {
            if(result[i] != '0') {
                zero = 1;
                break;
            }
        }
    }
    result = result.substr(0, y);
    if(zero == 1 || (zero == 0 && ((result[y - 1] - '0') % 2))) {
        result = vectorToString(addVectors(stringToVector(result),
{1}));
        if(result.length() > y) {
            result = result.substr(1, y);
            intPart += 1;
        }
    }
    return std::make_pair(intPart, result);
}
```

Если бит под номером Y меньше 5, то нужно сделать округление к нулю.
Если бит под номером Y больше 5, то нужно убрать все биты, начиная с Y, и к оставшейся дробной части добавить 1.
Если бит под номером Y равен 5, то проверяются следующие биты. Если не все их них равны 0, или они все равны 0 и значение бита с номером Y – 1 нечетное, то происходит то же, что и с Y >5. Если все они равны 0 и значение бита с номером Y-1 четное, то нужно сделать округление к нулю.

3. Если округление к +∞, то вызывается функция:

```
std::pair<long long, std::string> towardInfinity(long long intPart,
std::string result, size_t x, size_t y, bool sign) {
    if (sign){
        bool zero = true;
        for(size_t i = y; i < x; i++) {
            if(result[i] != '0') {
                zero = false;
                break;
            }
        }
        result = result.substr(0, y);
        if(!zero) {
            result =
vectorToString(addVectors(stringToVector(result), {1}));
            if(result.length() > y) {
                result = result.substr(1, y);
                intPart += 1;
            }
        }
    }
    else {
        result = result.substr(0, y);
    }
    return std::make_pair(intPart, result);
}
```

Если число отрицательное, то нужно сделать округление к 0.
Если число положительное и, начиная с Y бита, все биты – 0, то нужно сделать округление к нулю.
Если число положительно и оставшиеся биты ненулевые, то убираются биты, начиная с Y, и добавляется 1 к оставшейся дробной части.

4. Если округление к -∞, то вызывается функция:

```
std::pair<long long, std::string> towardNegInfinity(long long
intPart, std::string result, size_t x, size_t y, bool sign) {
    if(sign) {
        result = result.substr(0, y);
    }
    else {
        bool zero = true;
        for(size_t i = y; i < x; i++) {
            if(result[i] != '0') {
                zero = false;
                break;
            }
        }
        result = result.substr(0, y);
        if(!zero) {
            result =
vectorToString(addVectors(stringToVector(result), {1}));
            if(result.length() > y) {
                result = result.substr(1, y);
                intPart -= 1;
            }
        }
    }
    return std::make_pair(intPart, result);
}
```

Если число положительное, то нужно сделать округление к 0.

Если число отрицательное и, начиная с Y бита, все биты – 0, то нужно сделать округление к нулю.

Если число отрицательное и оставшиеся биты ненулевые, то убираются биты, начиная с Y, и отнимается 1 от оставшейся дробной части.

Дробная часть

Была реализована функция для преобразования дробной части двоичного числа в десятичное число. Для этого необходимо сначала умножить число, хранящееся в векторе, на 10^n, а затем разделить на 2^n. То есть, необходимо умножить число на 5^n, эта функция реализована с помощью длинной арифметики. Так как нам нужен лишь остаток от деления, то возьмем лишь n последних чисел.

```
std::vector<long long> fractionalPart(std::vector<long long> a, size_t n) {
    std::vector<long long> b; b.push_back(5);
    std::vector<long long> five; five.push_back(5);
    for(size_t i = 0; i + 1 < n; i++) {
        b = multiplyVectors(b, five);
    }

    std::vector<long long> result = multiplyVectors(a, b);
    if(result.size() > n) {
        std::reverse(result.begin(), result.end());
        while(result.size() > n) result.pop_back();
        std::reverse(result.begin(), result.end());
    }
    else if(result.size() < n) {
        std::reverse(result.begin(), result.end());
        while(result.size() < n) result.push_back(0);
        std::reverse(result.begin(), result.end());
    }

    return result;
}
```

Вывод на экран

Функции toFormat и toMultFormat возвращают знак, целую и дробные части. Во-первых, если при делении оказалось, что делитель – 0, то программа выведет “error”. Затем проверяется, чтобы в дробной части было 3 символа, иначе добавляются нули в конец, проверяется, если знак отрицательный, целая часть равна 0 и дробная часть не 0, то выводится “-”. Далее выводится целая и дробная часть через точку.

Числа с плавающей точкой

Различие между half precision и single precision состоит в количестве бит, отведенных под знак, экспоненту и мантиссу. В формате half precision используется 16 бит, где 1 бит отведен под знак, 5 бит - под экспоненту и 10 бит - под мантиссу. В формате single precision используется 32 бита, где 1 бит отведен под знак, 8 бит - под экспоненту и 23 бита - под мантиссу.

После считывания данных, вызывается следующая функция:

```
std::pair<bool, std::pair<long long, std::string>> doF(std::string result,
std::string format, size_t x, size_t y, size_t t) {
    bool sign = true;
    if(result.length() >= x + y + 1 && result[0] == '1'){
        sign = false;
    }
    std::pair<std::string, std::string> parts = splitFrac(result, x, y);
    long long exp = binaryToDecimal(parts.first, x, true) - t;
    std::string mantis = mantissa(parts.second, format, y, sign);
    return {sign, {exp, mantis}};
}
```

В начале функции определяется знак по первому биту входной строки. Затем входная строка разделяется на мантиссу и экспоненту с использованием функции splitFrac. Экспонента вычисляется, и мантисса приводится к нужному формату с помощью функции mantissa.

Функция splitFrac разделяет входную строку следующим образом:

```
std::pair<std::string, std::string> splitFrac(std::string result, size_t x,
size_t y) {
    while (result.length() < x + y + 1) {
        result = "0" + result;
    }
    return {result.substr(1, x), result.substr(x + 1, result.length() - x -
1)};
}
```

Далее, функция mantissa приводит мантиссу к нужному формату:

```
std::string mantissa(std::string result, std::string format, size_t x, bool
sign) {
    std::pair<long long, std::string> res = roundingFloating(result, format,
x, sign);
    if(!sign) res.first *= -1;
    result = decimalToBinary(res.first, x);
    while (result.length() < x) result = "0" + result;
    while (result.length() % 4 != 0) result += "0";
    result = binaryToHex(result);
    return result;
}
```

Функция roundingFloating выполняет округление:

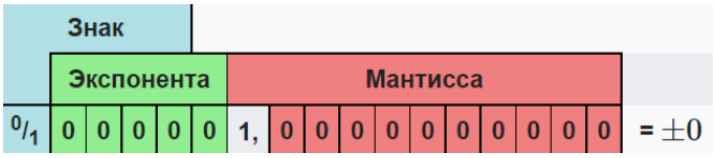
```
std::pair<long long, std::string> roundingFloating(std::string result,
std::string format, size_t x, bool sign) {
    long long intPart = binaryToDecimal(result.substr(0, x), x, false);
    if(!sign) intPart *= -1;
    if (result.length() > x)
        return rounding(intPart, result.substr(x + 1, result.length() - x),
format, x, 0, sign);
    else return {intPart, result};
}
```

После выполнения этих функций мантисса округляется и возвращается в шестнадцатеричной системе счисления. Также учитываются специальные значения (nan, inf, -inf).

Хранение чисел

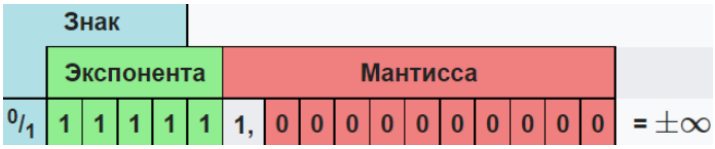
Изначально число представлено в виде двоичной строки. Затем экспонента хранится в виде десятичного числа, а мантисса – в виде шестнадцатиричной строки.

Специальные случаи



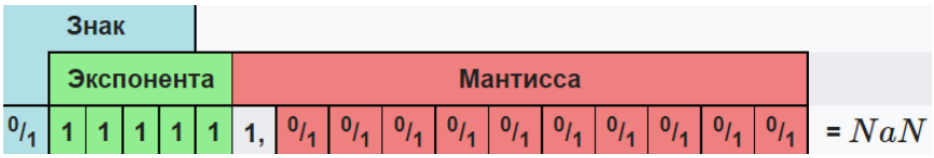
Ноль (со знаком)

Проверяем, если экспонента нулевая, значит это нуль.



Бесконечность (со знаком)

Проверяем, если экспонента состоит из единиц, а мантисса нулевая, значит это бесконечность (положительная или отрицательная, в зависимости от знака).



Неопределенность

Проверяем, если экспонента состоит из единиц, а мантисса ненулевая, значит это неопределенность.

Обрабатываются некоторые операции со специальными случаями:

1. Сумма бесконечности и числа – бесконечность
2. Сумма бесконечностей с одинаковым знаком – бесконечность с соответствующим знаком
3. Сумма бесконечностей с разными знаками – nan
4. Вычитание работает симметрично
5. При умножении на 0 ответ 0
6. Произведение двух бесконечностей с одинаковым знаком – бесконечность
7. 0/0 – nan
8. При делении числа на нуль со знаком получается бесконечность с соответствующим знаком