Государственное образовательное учреждение высшего образования

«Уральский федеральный университет имени первого Президента России Б.Н. Ельцина»

090302 200008 006

**«Технологии обработки информации»**

Семестр 4

**ОТЧЁТ**

по лабораторным работам

Лабораторная работа №2



Преподаватель: Плотников В.Ю.

Студент : Киселев В.А.

Группа : ФТ — 200008

Екатеринбург 2022 г.

Оглавление

[Цель работы 3](#_Toc107518685)

[Введение 3](#_Toc107518686)

[Постановка задачи 3](#_Toc107518687)

[Средства разработки 4](#_Toc107518688)

[Описание алгоритма 4](#_Toc107518689)

[Реализация 4](#_Toc107518690)

[С++ 4](#_Toc107518691)

[Python 6](#_Toc107518692)

[JavaScript 8](#_Toc107518693)

[Результаты 10](#_Toc107518694)

[C++ 10](#_Toc107518695)

[Python 10](#_Toc107518696)

[JavaScript 10](#_Toc107518697)

[Анализ результатов 10](#_Toc107518698)

[Выводы 10](#_Toc107518699)

# Цель работы

Экспериментально установить пределы представимых значений для вещественных типов данных в следующих языках программирования: C++ (для типов float, double и long double), Python и JavaScript. На основании полученных результатов сделать заключение о способе представления и размере. Оформить отчёт о проделанной работе.

# Введение

Есть два способа представления вещественных чисел с фиксированной и с плавающей запятой.

Число с фиксированной запятой — формат представления вещественного числа в памяти ЭВМ в виде целого числа. При этом само число x и его целочисленное представление  связаны формулой , где  — цена (вес) младшего разряда.

Но вещественные числа обычно представляются в виде чисел с плавающей запятой. Числа с плавающей запятой — один из возможных способов представления действительных чисел, который является компромиссом между точностью и диапазоном принимаемых значений, его можно считать аналогом экспоненциальной записи чисел, но только в памяти компьютера.

Число с плавающей запятой состоит из набора отдельных двоичных разрядов, условно разделенных на так называемые знак, порядок и мантиссу. В наиболее распространённом формате (стандарт IEEE 754) число с плавающей запятой представляется в виде набора битов, часть из которых кодирует собой мантиссу числа, другая часть — показатель степени, и ещё один бит используется для указания знака числа (0 — если число положительное, 1 — если число отрицательное). При этом порядок записывается как целое число в коде со сдвигом, а мантисса — в нормализованном виде, своей дробной частью в двоичной системе счисления.

Знак — один бит, указывающий знак всего числа с плавающей точкой. Порядок и мантисса — целые числа, которые вместе со знаком дают представление числа с плавающей запятой в следующем виде:

, где  — знак,  — основание,  — порядок, а  — мантисса. Десятичное число, записываемое как , где  — число в полуинтервале ,  — степень, в которой стоит множитель 10; в нормализированной форме модуль  будет являться мантиссой, а  — порядком, а  будет равно  тогда и только тогда, когда  принимает отрицательное значение

# Постановка задачи

1) Разработать алгоритм

2) Реализация алгоритма

3) Провести эксперимент

4) Анализ результатов

5) Выводы

# Средства разработки

1) CLion 2021.3.3

2) PyCharm Professional 2021.3.2

3) WebStorm 2021.3.2

4) Google Chrome 98.0.4758.102

# Описание алгоритма

Для нахождения верхней границы положительных чисел нам необходимо сперва приблизиться к бесконечности. Чтобы это сделать используем постоянное умножение числа на 2, до тех пор, пока правая граница не станет равна бесконечности.

Далее еще больше приблизимся к бесконечности умножая шаг на 3, пока правая граница не станет равна бесконечности.

Дальше будем прибавлять шаг, который был до того, как правая граница ушла в бесконечность, и если правая граница будет равна бесконечности уменьшим шаг в 2 раза, если граница не стала равна бесконечности сместимся к правой границе. Будем выполнять это действие пока число не перестанет изменяться при прибавлении шага.

Для нахождения нижней границы положительных чисел будем смещаться от единицы на шаг равный 0.5, если число стало равно нулю или меньше его, то уменьшим шаг в два раза, если изменения не произошло сместимся ближе к нулю. Будем продолжать это делать до тех пор, пока число не перестанет изменяться.

Для отрицательных чисел выполним те же действия, только наоборот.

# Реализация

## С++

#include <iostream>  
#include <limits>  
  
using namespace std;  
  
template <typename T> T search\_positive\_limit\_max () {  
 T leftBorderOfNumber;  
 T rightBorderOfNumber = (T) 2;  
 long double step = 2;  
  
 while (true) {  
 leftBorderOfNumber = rightBorderOfNumber;  
 rightBorderOfNumber \*= step;  
 if (rightBorderOfNumber == INFINITY) {  
 rightBorderOfNumber = leftBorderOfNumber + step;  
 break;  
 }  
 }  
  
 while (true) {  
 leftBorderOfNumber = rightBorderOfNumber;  
 step \*= 3;  
 rightBorderOfNumber += step;  
 if (rightBorderOfNumber == INFINITY) break;  
 }  
  
 while (true) {  
 rightBorderOfNumber = leftBorderOfNumber + step;  
 if (rightBorderOfNumber == leftBorderOfNumber) break;  
 (rightBorderOfNumber == INFINITY) ? (step /= 2) : (leftBorderOfNumber = rightBorderOfNumber);  
 }  
  
 return leftBorderOfNumber;  
}  
  
  
template <typename T> T search\_positive\_limit\_min () {  
 T leftBorderOfNumber;  
 T rightBorderOfNumber = (T) 1;  
 long double step = 0.5;  
  
 while (true) {  
 leftBorderOfNumber = rightBorderOfNumber - step;  
 if (rightBorderOfNumber == leftBorderOfNumber) break;  
 (leftBorderOfNumber <= 0) ? (step /= 2) : (rightBorderOfNumber = leftBorderOfNumber);  
 }  
  
 return rightBorderOfNumber;  
}  
  
  
template <typename T> T search\_negative\_limit\_max() {  
 T leftBorderOfNumber = (T) -2;  
 T rightBorderOfNumber = (T) -1;  
 long double step = 2;  
  
 while (true) {  
 rightBorderOfNumber = leftBorderOfNumber;  
 leftBorderOfNumber \*= step;  
 if (leftBorderOfNumber == -INFINITY) {  
 leftBorderOfNumber = rightBorderOfNumber - step;  
 break;  
 }  
 }  
  
 while (true) {  
 rightBorderOfNumber = leftBorderOfNumber;  
 step \*= 3;  
 leftBorderOfNumber -= step;  
 if (leftBorderOfNumber == -INFINITY) break;  
 }  
  
 while (true) {  
 leftBorderOfNumber = rightBorderOfNumber - step;  
 if (rightBorderOfNumber == leftBorderOfNumber) break;  
 (leftBorderOfNumber == -INFINITY) ? (step = step / 2) : (rightBorderOfNumber = leftBorderOfNumber);  
 }  
  
 return rightBorderOfNumber;  
}  
  
  
template <typename T> T search\_negative\_limit\_min() {  
 T leftBorderOfNumber = (T) -1;  
 T rightBorderOfNumber;  
 long double step = 0.5;  
  
 while (true) {  
 rightBorderOfNumber = leftBorderOfNumber + step;  
 if (rightBorderOfNumber == leftBorderOfNumber) break;  
 (rightBorderOfNumber >= 0) ? (step /= 2) : (leftBorderOfNumber = rightBorderOfNumber);  
 }  
  
 return leftBorderOfNumber;  
}  
  
template <typename T> void fine\_print () {  
 cout << typeid(T).name() << " max(+): " << search\_positive\_limit\_max <T> () << "; min(+): " << search\_positive\_limit\_min <T> ()  
 << "; max(-): " << search\_negative\_limit\_max <T> () << "; min(-): " << search\_negative\_limit\_min <T> () << endl;  
}  
  
int main()  
{  
 fine\_print <float> ();  
 fine\_print <double> ();  
 fine\_print <long double> ();  
 return 0;  
}

## Python

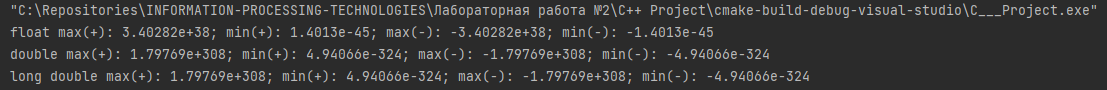
p\_inf = float('inf')  
n\_inf = float('-inf')  
  
  
def search\_positive\_limit\_max():  
 rightBorderOfNumber = 2.0  
 step = 2.0  
 while True:  
 leftBorderOfNumber = rightBorderOfNumber  
 rightBorderOfNumber \*= step  
 if rightBorderOfNumber == p\_inf:  
 rightBorderOfNumber = leftBorderOfNumber + step  
 break  
 while True:  
 step \*= 3.0  
 leftBorderOfNumber = rightBorderOfNumber  
 rightBorderOfNumber += step  
 if rightBorderOfNumber == p\_inf:  
 break  
 while True:  
 rightBorderOfNumber = leftBorderOfNumber + step  
 if rightBorderOfNumber == leftBorderOfNumber:  
 break  
 if rightBorderOfNumber == p\_inf:  
 step /= 2.0  
 else:  
 leftBorderOfNumber = rightBorderOfNumber  
 return leftBorderOfNumber  
  
  
def search\_positive\_limit\_min():  
 rightBorderOfNumber = 1.0  
 step = 0.5  
 while True:  
 leftBorderOfNumber = rightBorderOfNumber - step  
 if rightBorderOfNumber == leftBorderOfNumber:  
 break  
 if leftBorderOfNumber <= 0:  
 step = step / 2.0  
 else:  
 rightBorderOfNumber = leftBorderOfNumber  
 return rightBorderOfNumber  
  
  
def search\_negative\_limit\_max():  
 leftBorderOfNumber = -2.0  
 rightBorderOfNumber = -1.0  
 step = 2.0  
 last\_step = 0  
 while True:  
 leftBorderOfNumber \*= step  
 if leftBorderOfNumber == n\_inf:  
 leftBorderOfNumber = rightBorderOfNumber - step  
 break  
 while True:  
 last\_step = step  
 step \*= 3.0  
 rightBorderOfNumber = leftBorderOfNumber  
 leftBorderOfNumber -= step  
 if leftBorderOfNumber == n\_inf:  
 if step == p\_inf:  
 step = last\_step  
 break  
 while True:  
 leftBorderOfNumber = rightBorderOfNumber - step  
 if rightBorderOfNumber == leftBorderOfNumber:  
 break  
 if leftBorderOfNumber == n\_inf:  
 step /= 2.0  
 else:  
 rightBorderOfNumber = leftBorderOfNumber  
 return rightBorderOfNumber  
  
  
def search\_negative\_limit\_min():  
 leftBorderOfNumber = -1.0  
 step = 0.5  
 while True:  
 rightBorderOfNumber = leftBorderOfNumber + step  
 if rightBorderOfNumber == leftBorderOfNumber:  
 break  
 if rightBorderOfNumber >= 0:  
 step /= 2.0  
 else:  
 leftBorderOfNumber = rightBorderOfNumber  
 return leftBorderOfNumber  
  
  
print('(+) max: ', search\_positive\_limit\_max(), ' min: ', search\_positive\_limit\_min())  
print('(-) max: ', search\_negative\_limit\_max(), ' min: ', search\_negative\_limit\_min())

## JavaScript

function search\_positive\_limit\_max() {  
 let leftBorderOfNumber = 1  
 let rightBorderOfNumber = 2  
 let step = 2  
  
 while (true){  
 leftBorderOfNumber = rightBorderOfNumber  
 rightBorderOfNumber \*= step  
 if (rightBorderOfNumber === ***Infinity***) {  
 rightBorderOfNumber = leftBorderOfNumber + step  
 break  
 }  
 }  
 while (true) {  
 step \*= 3  
 leftBorderOfNumber = rightBorderOfNumber  
 rightBorderOfNumber += step  
 if (rightBorderOfNumber === ***Infinity***) {  
 break  
 }  
 }  
 while (true) {  
 rightBorderOfNumber = leftBorderOfNumber + step  
 if (rightBorderOfNumber === leftBorderOfNumber) {  
 break  
 }  
 if(rightBorderOfNumber === ***Infinity***) {  
 step /= 2  
 } else {  
 leftBorderOfNumber = rightBorderOfNumber  
 }  
 }  
 return leftBorderOfNumber  
}  
  
  
function search\_positive\_limit\_min() {  
 let leftBorderOfNumber  
 let rightBorderOfNumber = 1  
 let step = 0.5  
 while (true) {  
 leftBorderOfNumber = rightBorderOfNumber - step  
 if(rightBorderOfNumber === leftBorderOfNumber) {  
 break  
 }  
 if(leftBorderOfNumber <= 0) {  
 step /= 2  
 } else {  
 rightBorderOfNumber = leftBorderOfNumber  
 }  
 }  
 return rightBorderOfNumber  
}  
  
  
function search\_negative\_limit\_max() {  
 let leftBorderOfNumber = -2  
 let rightBorderOfNumber = 1  
 let step = 2  
 while (true) {  
 rightBorderOfNumber = leftBorderOfNumber  
 leftBorderOfNumber \*= step  
 if (leftBorderOfNumber === -***Infinity***) {  
 leftBorderOfNumber = rightBorderOfNumber + step  
 break  
 }  
 }  
 while (true) {  
 step \*= 3  
 rightBorderOfNumber = leftBorderOfNumber  
 leftBorderOfNumber -= step  
 if (leftBorderOfNumber === -***Infinity***) {  
 break  
 }  
 }  
 while (true) {  
 leftBorderOfNumber = rightBorderOfNumber - step  
 if (rightBorderOfNumber === leftBorderOfNumber) {  
 break  
 }  
 if (leftBorderOfNumber === -***Infinity***) {  
 step /= 2  
 } else {  
 rightBorderOfNumber = leftBorderOfNumber  
 }  
 }  
 return rightBorderOfNumber  
}  
  
function search\_negative\_limit\_min() {  
 let leftBorderOfNumber = -1  
 let rightBorderOfNumber  
 let step = 0.5  
 while (true){  
 rightBorderOfNumber = leftBorderOfNumber + step  
 if(rightBorderOfNumber === leftBorderOfNumber) {  
 break  
 }  
 if(rightBorderOfNumber >= 0) {  
 step /= 2  
 } else {  
 leftBorderOfNumber = rightBorderOfNumber  
 }  
 }  
 return leftBorderOfNumber  
}  
  
***console***.log("(+): max" + search\_positive\_limit\_max() + " min: " + search\_positive\_limit\_min())  
***console***.log("(-) max: " + search\_negative\_limit\_max() + " min: " + search\_negative\_limit\_min())

# Результаты

## C++



## Python

Изображение выглядит как текст

Автоматически созданное описание

## JavaScript



# Анализ результатов

В результате вычислительного эксперимента мы получили значения верхних и нижних границ для положительных и отрицательных вещественных типов данных.

В языке С++ получились следующие значения.

Переменная типа Float содержит значения в промежутке , а ближайшие к нулю значения равны , переменная типа Double и Long Double содержит значения в промежутке , а ближайшие к нулю значения равны

В языке Python переменная типа Float содержит значения в промежутке [-1.79\*10308; 1.79\*10308], а ближайшие к нулю значения равны -5\*10-324 и 5\*10-324

В языке JavaScript переменная типа Number содержит значения в промежутке [-1.79\*10308; 1.79\*10308], а ближайшие к нулю значения равны -5\*10-324 и 5\*10-324

То, что предельные значения равны для положительных и отрицательных чисел, говорит о том, что во всех этих языках числа представлены в виде числа с мажорантой, и то, что первое число в двоичном представлении отвечает за знак числа.

Так же можно предположить, что Double в C++, Float в Python и Number в JavaScript представлены одинаково

# Выводы

Мы провели вычислительный эксперимент на трех языках программирования: С++, Python, JavaScript. Определили границы представления вещественных чисел на этих языках и нашли ближайшие числа к нулю в этих языках.