Лабораторная работа по типу float
Выполнила Высоцкая Ира Б03-502

0. unsigned int -> binary

```
void print_binary(unsigned value) {
  auto mask = 1u << 31;
  unsigned n;
  for (auto i = 1u; i <= 32u; i++) {
    std::cout << ((value & mask) != 0);
    mask = mask >> 1;
    if (i % 4 == 0)
      std::cout << ' ';
    if (i % 8 == 0)
      std::cout << ' ';
  }
  std::cout << std::endl;
}
```

1. float->binary

```
union float_unsigned {
  unsigned Unsigned;
  float Float;
};
```

в main мы пишем:

```
float_unsigned fu;
std::cin >> fu.Float;
print_binary(fu.Unsigned);
```

2. Переполнение мантиссы

```
void overflow_mantissa() {
  float_unsigned n;
  n.Float = 10.0;
  for (int i = 0; i < 20; i++) {
    std::cout << "десятичное: " << n.Float << ", двоичное: ";
    print_binary(n.Unsigned);
    std::cout << std::endl;
    n.Float *= 10;
  }
}
```

Что мы видим на выводе:

```
десятичное: 10.00, двоичное: 0100 0001  0010 0000  0000 0000  0000 0000
десятичное: 100.00, двоичное: 0100 0010  1100 1000  0000 0000  0000 0000
десятичное: 1000.00, двоичное: 0100 0100  0111 1010  0000 0000  0000 0000
десятичное: 10000.00, двоичное: 0100 0110  0001 1100  0100 0000  0000 0000
десятичное: 100000.00, двоичное: 0100 0111  1100 0011  0101 0000  0000 0000
десятичное: 1000000.00, двоичное: 0100 1001  0111 0100  0010 0100  0000 0000
десятичное: 10000000.00, двоичное: 0100 1011  0001 1000  1001 0110  1000 0000
десятичное: 100000000.00, двоичное: 0100 1100  1011 1110  1011 1100  0010 0000
десятичное: 1000000000.00, двоичное: 0100 1110  0110 1110  0110 1011  0010 1000
десятичное: 10000000000.00, двоичное: 0101 0000  0001 0101  0000 0010  1111 1001
десятичное: 99999997952.00, двоичное: 0101 0001  1011 1010  0100 0011  1011 0111
десятичное: 999999995904.00, двоичное: 0101 0011  0110 1000  1101 0100  1010 0101
десятичное: 9999999827968.00, двоичное: 0101 0101  0001 0001  1000 0100  1110 0111
десятичное: 100000000376832.00, двоичное: 0101 0110  1011 0101  1110 0110  0010 0001
десятичное: 999999986991104.00, двоичное: 0101 1000  0110 0011  0101 1111  1010 1001
десятичное: 10000000272564224.00, двоичное: 0101 1010  0000 1110  0001 1011  1100 1010
десятичное: 99999998430674944.00, двоичное: 0101 1011  1011 0001  1010 0010  1011 1100
десятичное: 999999984306749440.00, двоичное: 0101 1101  0101 1110  0000 1011  0110 1011
десятичное: 9999999980506447872.00, двоичное: 0101 1111  0000 1010  1100 0111  0010 0011
десятичное: 10000000200408773427 2.00, двоичное: 0110 0000  1010 1101  0111 1000  1110 1100
```

Степень десятки у нас перестает выводиться, когда хотим получить 10**11, т.е. начинается переполнение мантиссы

3. Бесконечный цикл

По результатам п2 видим, что начинать с совсем маленьких чисел нет смысла, но при этом при запуске с 10**10 у меня сразу запустился бесконечный цикл. Поэтому начальное и конечное числа выбрала такие, как в коде

```
void infinite_cycle() {
  float_unsigned n;
  n.Float = 10000000.0;
  for (float i = 0; i < 6777230; i++) {
    n.Float += 1;
    std::cout << "десятичное: " << n.Float << ", двоичное: ";
    print_binary(n.Unsigned);
  }
}
```

Тогда на части вывода видим:

```
десятичное: 16777185.00, двоичное: 0100 1011  0111 1111  1111 1111  1110 0001
десятичное: 16777186.00, двоичное: 0100 1011  0111 1111  1111 1111  1110 0010
десятичное: 16777187.00, двоичное: 0100 1011  0111 1111  1111 1111  1110 0011
десятичное: 16777188.00, двоичное: 0100 1011  0111 1111  1111 1111  1110 0100
десятичное: 16777189.00, двоичное: 0100 1011  0111 1111  1111 1111  1110 0101
десятичное: 16777190.00, двоичное: 0100 1011  0111 1111  1111 1111  1110 0110
десятичное: 16777191.00, двоичное: 0100 1011  0111 1111  1111 1111  1110 0111
десятичное: 16777192.00, двоичное: 0100 1011  0111 1111  1111 1111  1110 1000
десятичное: 16777193.00, двоичное: 0100 1011  0111 1111  1111 1111  1110 1001
десятичное: 16777194.00, двоичное: 0100 1011  0111 1111  1111 1111  1110 1010
десятичное: 16777195.00, двоичное: 0100 1011  0111 1111  1111 1111  1110 1011
десятичное: 16777196.00, двоичное: 0100 1011  0111 1111  1111 1111  1110 1100
десятичное: 16777197.00, двоичное: 0100 1011  0111 1111  1111 1111  1110 1101
десятичное: 16777198.00, двоичное: 0100 1011  0111 1111  1111 1111  1110 1110
десятичное: 16777199.00, двоичное: 0100 1011  0111 1111  1111 1111  1110 1111
десятичное: 16777200.00, двоичное: 0100 1011  0111 1111  1111 1111  1111 0000
десятичное: 16777201.00, двоичное: 0100 1011  0111 1111  1111 1111  1111 0001
десятичное: 16777202.00, двоичное: 0100 1011  0111 1111  1111 1111  1111 0010
десятичное: 16777203.00, двоичное: 0100 1011  0111 1111  1111 1111  1111 0011
десятичное: 16777204.00, двоичное: 0100 1011  0111 1111  1111 1111  1111 0100
десятичное: 16777205.00, двоичное: 0100 1011  0111 1111  1111 1111  1111 0101
десятичное: 16777206.00, двоичное: 0100 1011  0111 1111  1111 1111  1111 0110
десятичное: 16777207.00, двоичное: 0100 1011  0111 1111  1111 1111  1111 0111
десятичное: 16777208.00, двоичное: 0100 1011  0111 1111  1111 1111  1111 1000
десятичное: 16777209.00, двоичное: 0100 1011  0111 1111  1111 1111  1111 1001
десятичное: 16777210.00, двоичное: 0100 1011  0111 1111  1111 1111  1111 1010
десятичное: 16777211.00, двоичное: 0100 1011  0111 1111  1111 1111  1111 1011
десятичное: 16777212.00, двоичное: 0100 1011  0111 1111  1111 1111  1111 1100
десятичное: 16777213.00, двоичное: 0100 1011  0111 1111  1111 1111  1111 1101
десятичное: 16777214.00, двоичное: 0100 1011  0111 1111  1111 1111  1111 1110
десятичное: 16777215.00, двоичное: 0100 1011  0111 1111  1111 1111  1111 1111
десятичное: 16777216.00, двоичное: 0100 1011  1000 0000  0000 0000  0000 0000
десятичное: 16777216.00, двоичное: 0100 1011  1000 0000  0000 0000  0000 0000
десятичное: 16777216.00, двоичное: 0100 1011  1000 0000  0000 0000  0000 0000
десятичное: 16777216.00, двоичное: 0100 1011  1000 0000  0000 0000  0000 0000
десятичное: 16777216.00, двоичное: 0100 1011  1000 0000  0000 0000  0000 0000
десятичное: 16777216.00, двоичное: 0100 1011  1000 0000  0000 0000  0000 0000
десятичное: 16777216.00, двоичное: 0100 1011  1000 0000  0000 0000  0000 0000
десятичное: 16777216.00, двоичное: 0100 1011  1000 0000  0000 0000  0000 0000
десятичное: 16777216.00, двоичное: 0100 1011  1000 0000  0000 0000  0000 0000
десятичное: 16777216.00, двоичное: 0100 1011  1000 0000  0000 0000  0000 0000
десятичное: 16777216.00, двоичное: 0100 1011  1000 0000  0000 0000  0000 0000
десятичное: 16777216.00, двоичное: 0100 1011  1000 0000  0000 0000  0000 0000
десятичное: 16777216.00, двоичное: 0100 1011  1000 0000  0000 0000  0000 0000
десятичное: 16777216.00, двоичное: 0100 1011  1000 0000  0000 0000  0000 0000
десятичное: 16777216.00, двоичное: 0100 1011  1000 0000  0000 0000  0000 0000
irunka:~/mipt/informatics/informatics/лаба по float (2) (main)$
```

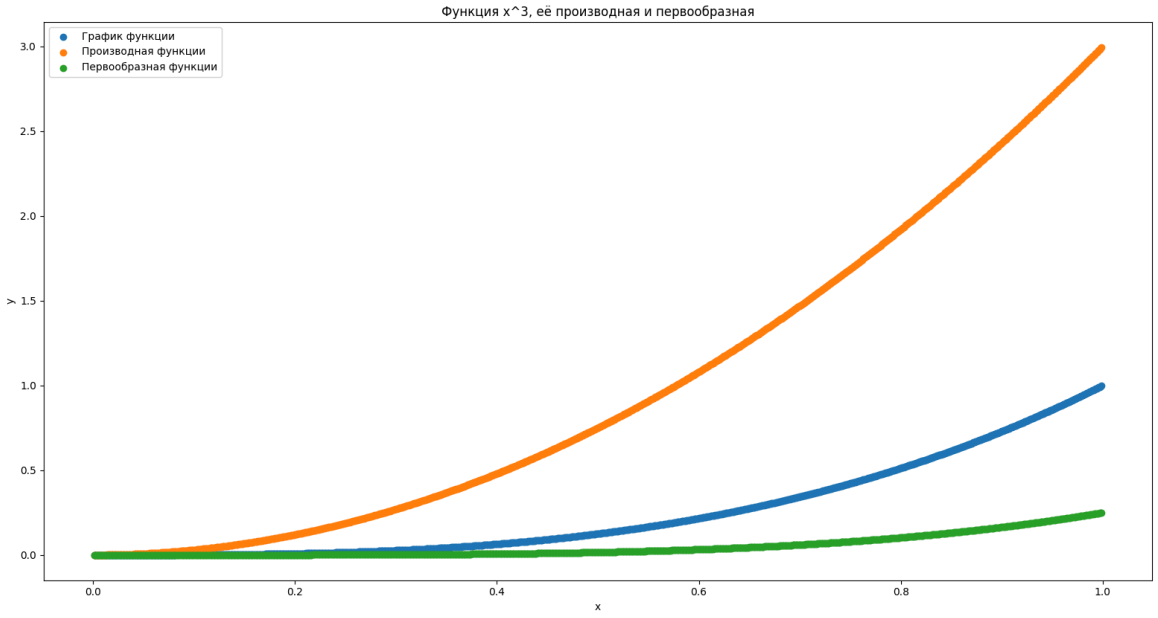
у меня бесконечный цикл запустился при значении 16777216.0

4. Численное интегрирование

Я хочу взять функцию $f = x^3$. Тогда её площадь от 0 до 1 будет равна

$$\int_0^1 x^3 dx = \frac{1}{4} = 0,25$$

Построили графики функции, её производной и её первообразной (для первообразной брала C=0):

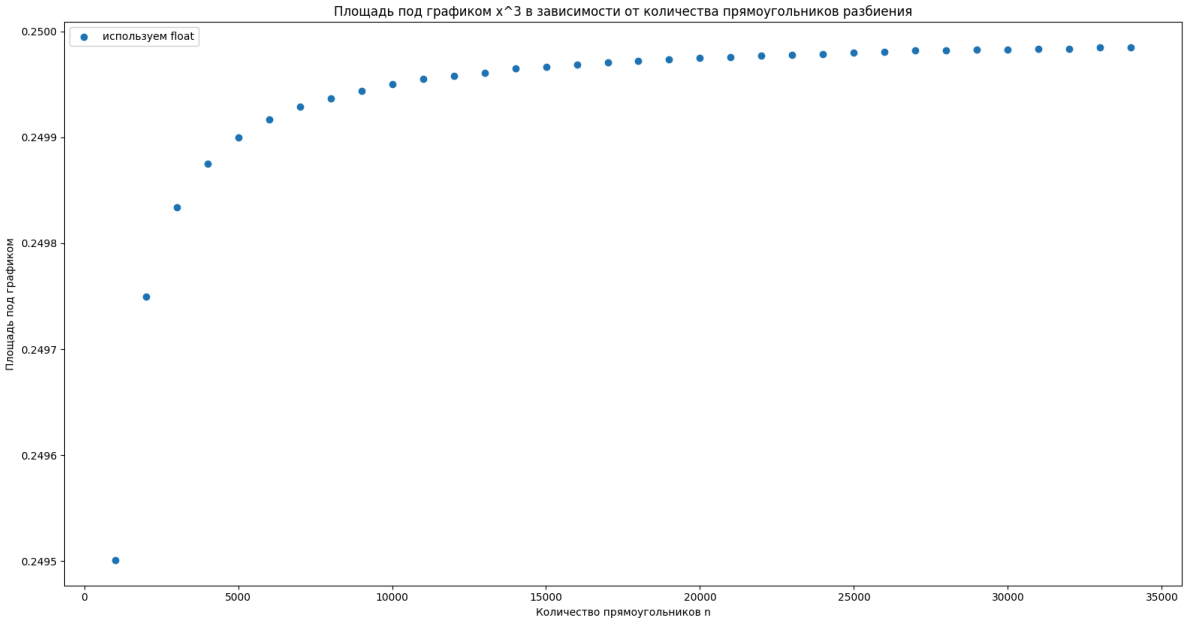


Код для подсчета площади под графиком в зависимости от количества прямоугольников в разбиении:

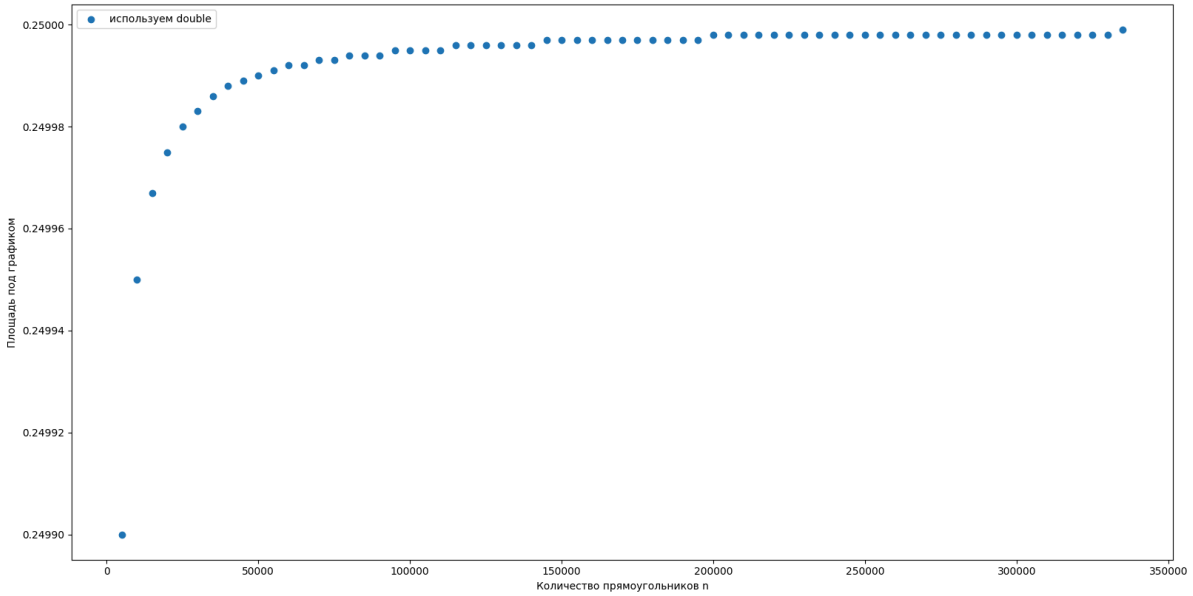
```
float pow(float x) { return x * x * x; }

float square(float n) {
  float step = 1.0f / n;
  float summ = 0;
  for (int i = 0; i < n; i++) {
    summ += pow(step * i) * step;
  }
  return summ;
}
```

Для достижения наиболее близкого к теоретическому значению я хочу увеличить количество прямоугольников разбиения (n)

Используем float: Я увеличивала n до 495001, в процессе значение 0,25 было достигнуто (например, при n=355001, n=375001, n=405001, n=445001, n=450001)



<u>Используем double:</u> При увеличении n до примерно 335001 и более значение площади под графиком перестает меняться (видимо бесконечно зациклилось), так что считаю, что больше брать не надо, при таком n значение площади под графиком будет максимально приближенным к 0,25 и равно 0,249999



На этом примере видно, что double хранит числа более точно: вроде бы с помощью разбиения на прямоугольники мы никогда не сможем получить реальное значение площади под графиком, что видно при использовании double. При использовании float из-за менее точного хранения информации мы можем получить значение 0,25