Федеральное государственное автономное образовательное учреждение
высшего образования
Национальный исследовательский университет
«Высшая школа экономики»

Факультет социально-экономических и компьютерных наук

Лабораторная работа №1

по дисциплине «Теоретические основы информатики»

Кодирование информации и представление данных в памяти компьютера

Выполнил: студент Яцишин Л.С., уч. группа ПСАПР-25-2

Пермь, 2025

Содержание

0 Правки

После 06.10.2025. Решено задание №4. Исправленны неточности.

После 04.10.2025. Я просмотрел семинары и файл с примером, и отредактировал работу (всё ещё первые 3 задания), чтобы она соответствовала требованиям. Так же исправил опечатки, ошибки и места, где я, как выяснилось, не доделал или сделал плохо.

1 Задание 1

Сколько раз выполнится цикл в программе, фрагмент кода которой на языке C++ приведён ниже, если переменная A имеет целочисленный тип – целое без знака в формате байта (контроль выхода за допустимый диапазон значений отключён). Какое значение примет переменная A после завершения цикла?
Поясните ответ – выполните трассировку программы («сухую прокрутку» – пошаговое выполнение вручную) – заполните таблицу, структура которой показана ниже, чтобы обосновать свой ответ (покажите, как меняется значение переменной – как выполняются операции – повторите их столько раз, сколько раз они повторятся при выполнении цикла):

```cpp
static unsigned char A;
A = 255;

do { A++;
} while (A != 0);
```

Трассировка

Оператор/ операция	Десятичное значение переменной A: ожидаемое/полученное	Внутреннее представление A	Комментарий
A = 255;	255 / 255	11111111_2 0xFF	Инициализация
A++	256 / 0	00000000_2 0x00	Переполнение
A != 0	итог: ложь	00000000_2 0x00	Условие не выполнилось
Цикл не повторился			

Итого: тело цикла выполнено ровно 1 раз; после завершения A == 0.

Пояснение

Номера битов в разрядной сетке:	[8]	7	6	5	4	3	2	1	0
+		1 0	1 0	1 0	1 0	1 0	1 0	1 0	1 1
	1	0	0	0	0	0	0	0	0

Скриншоты

Рис. 1: static unsigned char A = 255;

Рис. 2: A++;

Выводы

- Трассировка подтверждается отладкой: после единственного прохода A становится 0.
- Причина поведения – определённое поведение беззнаковых типов в C/C++. Арифметика для unsigned типов определяется по модулю 2^N (здесь $N = 8$), поэтому $11111111_2 + 00000001_2 = [1]00000000_2 = 0$ (восьмой разряд отпадает). При этом никакой ошибки не просиходит и процессор просто оставляет флаг переполнения, который можно при надобности обрабатывать.
- Вывод: цикл выполняется один раз, финальное значение A == 0.

2 Задание 2

Задание 2. Сколько раз выполнится цикл в программе, фрагмент кода которой на языке C++ приведён ниже, если переменная A имеет целочисленный тип – целое со знаком в формате байта (контроль выхода за допустимый диапазон значений отключён). Поясните ответ – выполните трассировку программы («сухую прокрутку») – заполните таблицу, чтобы обосновать свой ответ (покажите, как меняется значение переменной – повторите их столько раз, сколько раз они повторятся при выполнении цикла):

```
static signed char A;

A = -127;
while (A < 0) {
  A = A - 1;
}
```

Трассировка

Оператор/ операция	Десятичное значение переменной A: ожидаемое/полученное	Внутреннее представление A	Комментарий
A = -127;	-127 / -127	10000001_2 0x81	Инициализация
A < 0	итог: истина	10000001_2 0x81	Условие выполнилось
A = A - 1;	-128 / -128	10000000_2 0x80	Тело цикла выполнилось
A < 0	итог: истина	10000000_2 0x80	Условие выполнилось
A = A - 1;	-129 / 127	01111111_2 0x7F	Переполнение
A < 0	итог: ложь	01111111_2 0x7F	Условие не выполнилось

Пояснение

Номера битов в разрядной сетке:	7	6	5	4	3	2	1	0
	1	0	0	0	0	0	0	1
−	0	0	0	0	0	0	0	1
	0	0	0	0	0	0	0	1
	0	1	1	1	1	1	1	1

Чтобы получить дополнительный код для отрицательного числа в 8-битном signed char нужно:

1. представить модуль числа как unsigned char;
2. инвертировать все биты;
3. прибавить 1.

Это эквивалентно арифметике по модулю 2^8 : -x $\equiv 2^8 - x \pmod{2^8}$. Соответственно:

$$127_{10} = 01111111_2 \xrightarrow{\text{инверсия}} 10000000_2 \xrightarrow{+1} 10000001_2 = 0x81 \quad (-127_{10}).$$

Далее при вычитании второй единицы происходит перенос с 7ого бита (который является показателем знака) и соответственно при интерпретации этого байта как signed char получатеся 127.

Скриншоты

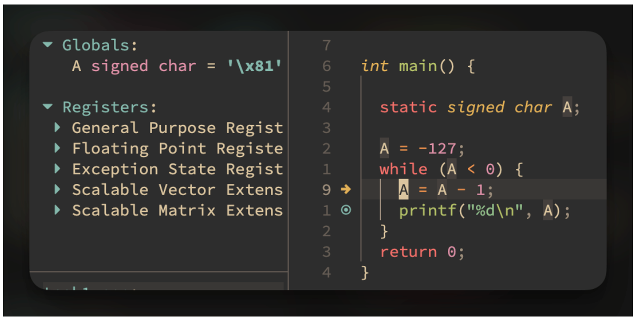

Рис. 3: static signed char A = -127;

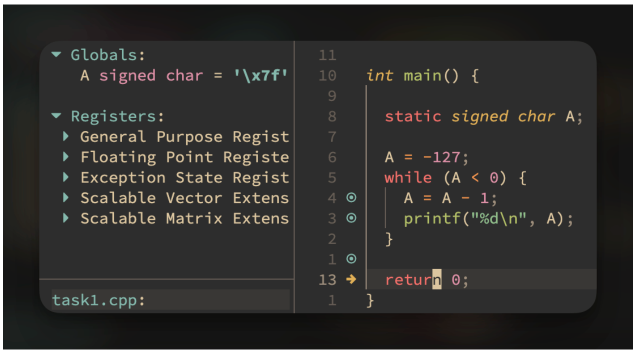

Рис. 4: A - 2;

Выводы

- Трассировка и отладка совпали: тело цикла выполняется дважды. Последовательность значений: $-127 \to -128 \to 127$, после чего условие $A < 0$ ложно и цикл прекращается.
- Причина: тип signed char имеет диапазон $[-128, 127]$. Первое вычитание даёт -128. Второе вычитание для -128 выходит за диапазон и отнимает бит с позиции отведённой для знака.

3 Задание 3

Сколько раз выполнится цикл в программе, фрагмент кода которой на языке Pascal приведён ниже, если переменная S имеет вещественный тип однократной точности, а переменная N – тип целого без знака в формате байта. Какое значение получит переменная S?
Поясните ответ – выполните трассировку программы (пошаговое выполнение вручную – «сухую прокрутку») – заполните таблицу, структура которой приведена ниже, вычисляя значения при выполнении каждого оператора, чтобы обосновать свой ответ (покажите, как меняется значение переменной – повторите их столько раз, сколько раз они повторятся при выполнении цикла, и заполните соответствующие строки в таблице)

```
{NOTE: real – двухкратная точность, single – однократная }
var S: single; N: byte; { Объявление переменных }
begin
  N := 3; { Присваивание переменной N значения }
  S := 1/N; { Присваивание переменной S значения – вычисляется выражение 1/N }
  while S <> 1 do {Выполнять, пока S не равно 1 }
   begin
    writeln(S); {вывести на экран}
    S := S + 1 / N;
   end;
  writeln(S); {вывести на экран}
end.
```

Трассировка

Оператор/ операция	Десятичное значение S	Внутреннее представление S	Комментарий
N := 3;	—	—	Инициализация
S := 1/N;	0.333333	0 01111101 01010101010101010101010	1/3
S <> 1	итог: истина	0 01111101 01010101010101010101010	Условие цикла выполнено
S := S + 1/N;	0.666666	0 01111110 01010101010101010101010	Суммирование двух 1/3
S <> 1	итог: истина	0 01111110 01010101010101010101010	Ещё не 1
S := S + 1/N;	1.0 / 0.999999	0 01111110 11111111111111111111111	Округление даёт 1.0
S <> 1	итог: ложь?	0 01111110 11111111111111111111111	Выход из цикла

Перевод десятичного числа в двоичную систему и нормализация:

$$\frac{1}{3}_{10} = 0.\overline{01}_2 = 0.010101010101\ldots_2$$

Для нормализации сдвигаем точку так, чтобы первое представимое число имело вид $1.\ldots$; первая единица стоит в позиции 2^{-2}, поэтому получаем:

$$0.0101010101\ldots_2 = 1.0101010101\ldots_2 \times 2^{-2}.$$

Мантисса (24 разряда: скрытая 1 + 23 дробных разряда): Нормализованная мантисса (бесконечная) —

$$M = 1.010101010101010101010101\ldots_2\,.$$

Нам нужно сохранить 23 дробных бита (скрытая 1 не хранится). Итоговая хранимая мантисса (скрытая 1 + дробная часть):

$$\underbrace{0\;1\;0\;1\;0\;1\;0\;1\;0\;1\;0\;1\;0\;1\;0\;1\;0\;1\;0\;1\;0\;1\;0}_{23}.$$

Порядок (со смещением): Истинный порядок $E = -2$, значит

$$E_{\text{смещённый}} = -2 + 127 = 125.$$

В двоичном виде (8 бит):

$$125_{10} = 01111101_2.$$

Знак: $S = 0$ (положительное число).

Итоговое 32-битное представление:

$$\underbrace{0}_{\text{знак}}\;\underbrace{01111101}_{\text{порядок}}\;\underbrace{01010101010101010101010}_{\text{дробная часть}} = \boxed{0{,}333333_{10}}.$$

Операция сложения: $S := S + 1/3$. Алгоритм процессора в двоичном виде:

1. Выравнивание порядков: не требуется, одинаковые числа.
2. Сложение мантисс (включая скрытую 1):

$$\begin{array}{r} 01.01010101010101010101010_2 \\ +01.01010101010101010101010_2 \\ \hline = 10.10101010101010101010100_2 \end{array}$$

   В результате появился перенос в старший разряд: форма $10.10\ldots$ — не нормализовано.
3. Нормализация: сдвигаем точку и увеличиваем порядок на 1:

$$10.10101010101010101010100_2 = 1.01010101010101010101010_2 \times 2^1.$$

   Поскольку у слагаемых порядок был -2, после увеличения он становится -1, что соответствует правильному виду для 2/3: $2/3 = 0.10101010\ldots_2 = 1.0101010\ldots_2 \times 2^{-1}$.

Итоговое записанное значение (после сложения):

$$0\;\;01111110\;\;01010101010101010101010$$

Примечание. Я не доконца разобрался почему в сухом прогоне получилось именно $0\;01111110\;11111111111111111111111_2 = 0.999999_{10}$, а не $0\;01111111\;00000000000000000000000_2 = 1_{10}$. При реальном запуске выполнение прекращается, хотя условие поставленно на чёткое равенство, а не на вхождение в диапазон. Может быть дело в округлении, которое где-то не делается или делается не так.

Перевод на C++ и сравнение

Переведите программу на язык C++, выбрав подходящие типы данных и операторы языка C++. Выполните «сухую прокрутку» программы на C++ (заполните трассировочную таблицу, заменив операторы языка Pascal на соответствующие операторы языка C++). Какие результаты получены? С какой точностью (сколько десятичных знаков) могут быть эти числа записаны в памяти компьютера в указанном формате? Проверьте, используя справочную систему Microsoft (информацию по типам данных в C++).
В чём разница в описаниях типов данных, чем различаются правила выполнения операций на языках C++ и Pascal? Объясните ответ. Проверьте результаты, выполнив программу на Pascal.

```cpp
#include <cstdio>

int main() {
  unsigned char N = 3;
  float S;

  S = 1.0f / N;

  int iters = 0;
  while (S != 1.0f) {
    printf("iters=%d, S=%.9E\n", iters, S);
    S = S + 1.0f / N;
    ++iters;
  }

  printf("iters=%d, S=%.9E\n", iters, S);
}
```

Теоретически, поскольку я постарался сделать как можно более прямой перевод между языками, трассировочная таблица Pascal и C++ должна совпадать (отличия только в синтаксисе).

Оператор/ операция	Десятичное значение S	Внутреннее представление S	Комментарий
unsigned char N = 3;	—	—	Инициализация
S = 1.0f / N;	0.333333	0 01111101 01010101010101010101010	1/3
S != 1.0f	итог: истина	0 01111101 01010101010101010101010	Условие цикла выполнено
S = S + 1.0f / N;	0.666666	0 01111110 01010101010101010101010	Суммирование двух 1/3
S != 1.0f	итог: истина	0 01111110 01010101010101010101010	Ещё не 1
S = S + 1.0f / N;	1.0 / 0.999999	0 01111110 11111111111111111111111	Округление даёт 1.0
S != 1.0f	итог: ложь?	0 01111110 11111111111111111111111	Выход из цикла

Рис. 5: Выполнение на Pascal

Рис. 6: Выполнение на C++

При корректном выборе одинаковой точности (Pascal: single, C++: float) существенной разницы в поведении программ нет: оба языка выполняют вещественное деление и сложение в указанном формате, и последовательность значений совпадает. Точность представления: формат IEEE-754 single (32 бита) имеет 24 бита значащих, что соответствует $\log_{10}(2^{24}) \approx 7{,}22$ значащим десятичным цифрам — то есть примерно 7 значащих десятичных цифр. Поэтому цифры после седьмой уже не являются точными. (Код на Pascal проверял в онлайн компиляторе Online Pascal Compiler)

Перевод на Python и сравнение

> Переведите программу на язык Python. Какие результаты получены? Сравните их с результатами выполнения программы на C++.

```
N: int = 3
S: float = 1 / N

iters = 0
while S != 1:
    print("iters =", iters, "S =", S)
    S = S + 1 / N
    iters += 1

print("iters =", iters, "S =", S)
```

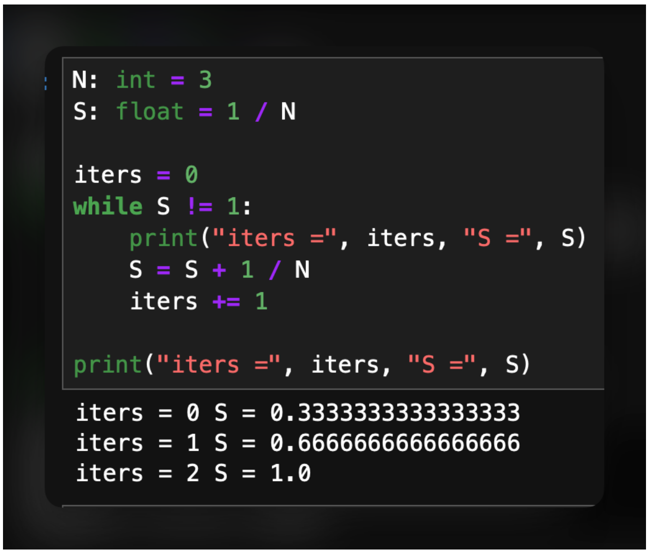

Рис. 7: Выполнение на Python

Сравнение. Результаты выполнения программы на python dпо сути ничем не отличаются от результатов выполнения программы на C++, за исключении точности. В Python не такая гранулированная система типов чисел как в C++ – в стандартном Python float – двухкратная точность, а int теоретически вообще может расти до бесконечности, потому что реализован через массив более маленьких фиксированного размера. На сколько я помню Integer в Haskell имеет такую же идею (в сравнении с Int). Соответственно там, где в C++ при N=7 было использованно float, в Python используется double и из-за накопления другой ошибки цикл никогда не завершается. (Если в C++ заменить тип N на double, то будет такое же поведение.)

Проверка на других значениях N

Как изменятся результаты, если организовать ввод значений переменной N и выполнить программу для других значений? Проведите эксперименты с разными значениями (7, 11). Всегда ди цикл будет выполняться конечное число раз? Чем объясняются полученные результаты? Опишите свои эксперименты в отчёте (заполните трассировочные таблицы, как это было показано выше) и поясните полученные ответы (коды вещественных чисел – их внутреннее представление – можно посмотреть в режиме отладки в 16-ричной системе, переключившись на дизассемблированный код, ...).

Оператор/ операция	Десятичное значение S	Внутреннее представление S	Комментарий
unsigned char N = 7;	—	—	Инициализация
S = 1.0f / N;	0.142857	0 01111100 00100100100100100100100	1/7
S != 1.0f	итог: истина	0 01111100 00100100100100100100100	Условие цикла истинно
S = S + 1.0f / N;	0.285714	0 01111101 00100100100100100100100	Сумма двух 1/7
S != 1.0f	итог: истина	0 01111101 00100100100100100100100	Ещё не 1
S = S + 1.0f / N;	0.428571	0 01111101 10110110110110110110110	Третье суммирование
S != 1.0f	итог: истина	0 01111101 10110110110110110110110	Ещё не 1
S = S + 1.0f / N;	0.571428	0 01111110 00100100100100100100100	Четвёртое суммирование
S != 1.0f	итог: истина	0 01111110 00100100100100100100100	Ещё не 1
S = S + 1.0f / N;	0.714285	0 01111110 01101101101101101101101	Пятое суммирование
S != 1.0f	итог: истина	0 01111110 01101101101101101101101	Ещё не 1
S = S + 1.0f / N;	0.857142	0 01111110 10110110110110110110110	Шестое суммирование
S != 1.0f	итог: истина	0 01111110 10110110110110110110110	Ещё не 1
S = S + 1.0f / N;	0.999999	0 01111110 11111111111111111111111	Седьмое суммирование даёт
S != 1.0f	итог: ложь?	0 01111110 11111111111111111111111	Цикл завершается

На сколько я понимаю, тут такая же проблема, как в задании 2. Потому что при реальном запуске программы на седьмом суммировании просиходит остановка.

Оператор/ операция	Десятичное значение S	Внутреннее представление S	Комментарий
unsigned char N = 11;	—	—	Инициализация
S = 1.0f / N;	0.090909	0 01111011 01110100010111010001011	1/11 представлено с усечением мантиссы (truncate)
S != 1.0f	итог: истина	0 01111011 01110100010111010001011	Условие цикла выполнено
S = S + 1.0f / N;	0.181818	0 01111100 01110100010111010001011	Сумма двух усечённых 1/11
S != 1.0f	итог: истина	0 01111100 01110100010111010001011	Ещё не 1
S = S + 1.0f / N;	0.272727	0 01111101 00010111010001011101000	Третье суммирование
S != 1.0f	итог: истина	0 01111101 00010111010001011101000	Ещё не 1
S = S + 1.0f / N;	0.363636	0 01111101 01110100010111010001010	Четвёртое суммирование
S != 1.0f	итог: истина	0 01111101 01110100010111010001010	Ещё не 1
S = S + 1.0f / N;	0.454545	0 01111101 11010001011101000101100	Пятое суммирование
S != 1.0f	итог: истина	0 01111101 11010001011101000101100	Ещё не 1
S = S + 1.0f / N;	0.545454	0 01111110 00010111010001011100111	Шестое суммирование
S != 1.0f	итог: истина	0 01111110 00010111010001011100111	Ещё не 1
S = S + 1.0f / N;	0.636363	0 01111110 01000101110100010111000	Седьмое суммирование
S != 1.0f	итог: истина	0 01111110 01000101110100010111000	Ещё не 1
S = S + 1.0f / N;	0.727272	0 01111110 01110100010111010001001	Восьмое суммирование
S != 1.0f	итог: истина	0 01111110 01110100010111010001001	Ещё не 1
S = S + 1.0f / N;	0.818181	0 01111110 10100010111010001011010	Девятое суммирование
S != 1.0f	итог: истина	0 01111110 10100010111010001011010	Ещё не 1
S = S + 1.0f / N;	0.909090	0 01111110 11010001011101000101011	Десятое суммирование
S != 1.0f	итог: истина	0 01111110 11010001011101000101011	Ещё не 1
S = S + 1.0f / N;	0.999999	0 01111110 11111111111111111111100	Одиннадцатое суммирование
S != 1.0f	итог: истина	0 01111110 11111111111111111111100	Условие цикла всё ещё истинно
S = S + 1.0f / N;	1.090908	0 01111111 00010111010001011100110	Двенадцатое суммирование — значение больше 1
S != 1.0f	итог: истина	0 01111111 00010111010001011100110	Цикл не завершается

1. Всегда ли цикл выполнится конечное число раз и чем это объясняется? Нет. Зависит от того, попадёт ли накопленное значение S ровно в представимое 1.0f. float (IEEE-754 single) имеет конечную точность. Значение 1/N часто не представимо точно, при суммировании идут округления; последовательность S_k = round(k*(1/N)) в типе float может в какой-то момент дать ровно 1.0f (тогда цикл закончится) или никогда не дать ровно 1.0f (тогда цикл не закончится).

- N = 3 — завершается (после трёх сумм S достигает ровно 1.0f).
- N = 7 — завершается (после семи сумм S округляется к 1.0f).
- N = 11 — НЕ завершается: на шаге, где ожидалось ровно 1, получается немного больше.

С практической точки зрения есть 2 вывода:

1. Главное смотреть не на название типов и операций, а на стандарт согласно которому они реализованы.
2. При необходимости сравнения двух чисел представленных в формате с "плавающей точкой" нужно сравнивать не на прямую, а на вхождения модуля разности выбранных чисел в определённый маленький диапазон.

4 Задание 4

Разработайте на основе самостоятельно выведенной рекуррентной формулы алгоритм вычисления суммы ряда S(x) для приближённого вычисления функции y = F(x) (x вводится с клавиатуры; выбор функции F(x) для вычислений разложением в ряд S(x) ограничивается предложенным списком (см. ниже), где для каждой функции приведена формула разложения в ряд и вес этой функции в оценке результатов выполнения задания по 10-балльной шкале).
Покажите порядок вывода рекуррентной формулы.
Опишите алгоритм на псевдокоде (см. пример в рекомендациях по выполнению задания и в материалах практических занятий).

Я выбрал $F(x) = \left(1 - \frac{x^2}{2}\right)\cos x - \frac{x}{2}\sin x$, соответственно $S = 1 - \frac{3}{2}x^2 + \ldots + (-1)^n \frac{2n^2+1}{(2n)!}x^{2n}$.

Анализ алгоритма

1. Вывод рекуррентной формулы общего члена. Пусть

$$A_n = (-1)^n \frac{2n^2+1}{(2n)!} x^{2n} \quad (n = 0, 1, 2, \ldots).$$

Тогда «шаг» между соседними слагаемыми равен

$$M_n = \frac{A_n}{A_{n-1}} = \frac{(-1)^n}{(-1)^{n-1}} \cdot \frac{2n^2+1}{2(n-1)^2+1} \cdot \frac{x^2}{(2n)(2n-1)} = -\frac{x^2}{(2n)(2n-1)} \cdot \frac{2n^2+1}{2(n-1)^2+1}.$$

Следовательно, рекуррентная формула:

$$A_n = A_{n-1} \cdot M_n, \qquad M_n = -\frac{x^2}{(2n)(2n-1)} \cdot \frac{2n^2+1}{2(n-1)^2+1}.$$

2. Начальные значения. Первый член ($n = 0$): $A_0 = 1$. Начальная сумма: $S\_0 = A_0 = 1$.

3. Алгоритм вычисления при заданном n.

```
Ввод x;
Ввод n;
A ← 1.0;
S ← 1.0;

Цикл по всем k от 1 до n с шагом 1
  Выполнять:
          A ← A × ( - (x^2)/((2k)(2k - 1)) × ( (2k^2 + 1)/(2(k - 1)^2 + 1) ) );
          S ← S + A;
Конец цикла по k;
Вывод S;
```

Реализация алгоритма на C++

Разработайте на основе выведенной рекуррентной формулы и разработанного алгоритма программу вычисления суммы ряда

1. а) для заданного количества N слагаемых A_k в сумме ряда S(x), (N вводится с клавиатуры);
2. б) с заданной точностью E (E вводится с клавиатуры) – суммируются слагаемые A_k, абсолютная величина которых не меньше заданного E;
3. в) с максимально возможной точностью (до машинного нуля – до слагаемого A_k, которое становится машинным нулём).

```cpp
#include <cstdio>
#include <cmath>

int main() {
  float Sn = 1.0f, Se = 1.0f, Se0 = 1.0f, A = 1.0f; // или double
  float x, e; // или double
  int n;

  printf("Введите x: ");
  scanf("%f", &x);
  printf("Введите n: ");
  scanf("%d", &n);
  printf("Введите e (точность вычислений): ");
  scanf("%f", &e);

  // а) Sn: сумма по заданному n
  for (int k = 1; k <= n; ++k) {
    A *= ( - (x * x) / ( (2.0f * k) * (2.0f * k - 1.0f) )
          * ( (2.0f * k * k + 1.0f) / ( 2.0f * (k - 1) * (k - 1) + 1.0f ) ) );
    Sn += A;
  }
  printf("\nSn = %g, n = %d, A_N=%gf\n", Sn, n, A);

  // б) Se: сумма до заданной точности e
  A = 1.0f;
  int i = 1;
```

```
  while (abs(A) >= e) {
    A *= ( - (x * x) / ( (2.0f * i) * (2.0f * i - 1.0f) )
          * ( (2.0f * i * i + 1.0f) / ( 2.0f * (i - 1) * (i - 1) + 1.0f ) ) );
    Se += A;
    ++i;
  }
  printf("Количество слагаемых для Se: %d (точность до e = %gf)\n", i, e);
  printf("Se = %g\n", Se);

  // в) Se0: сумма с точностью до машинного нуля
  A = 1.0f;
  i = 1;
  while (abs(A) > 0.0f) {
    A *= ( - (x * x) / ( (2.0f * i) * (2.0f * i - 1.0f) )
          * ( (2.0f * i * i + 1.0f) / ( 2.0f * (i - 1) * (i - 1) + 1.0f ) ) );
    Se0 += A;
    ++i;
  }
  printf("Количество слагаемых для Se0: %d (точность до e = 0.0f)\n", i);
  printf("Se0 = %g\n", Se0);

  return 0;
}
```

Такой же по струтуре код будет с двойной точностью для вещественных чисел (только соответственно типы будут double и вместо .0f будет просто .0).

Результаты

	x	N (а) / CA (б, в)	A_N (а) / E	Тип данных	S(x)	$y = F(x)$
а)	0.1	N = 100 000 000	A_N = 0.0f	float	0.985038	0.98503747...
а)	0.1	N = 100 000 000	A_N = 0.0	double	0.985037	0.98503747...
б)	0.1	CA = 4	E = 0.00001	float	0.985038	0.98503747...
б)	0.1	CA = 5	E = 0.000000001	double	0.985037	0.98503747...
в)	0.1	CA = 13	E = 0.0f	float	0.985038	0.98503747...
в)	0.1	CA = 63	E = 0.0	double	0.985037	0.98503747...
а)	10	N = 10	A_N = 82611.74f	float	1812.65	43.8346104...
а)	10	N = 10	A_N = 82611.74	double	1812.65	43.8346104...
б)	10	CA = 18	E = 0.1	float	43.8252	43.8346104...
б)	10	CA = 18	E = 0.1	double	43.833	43.8346104...
в)	10	CA = 47	E = 0.0f	float	43.8268	43.8346104...
в)	10	CA = 156	E = 0.0	double	43.8346	43.8346104...
а)	100	N = 10	A_N = 8.26174e+23f	float	8.0132e+23	-4285.41376...
а)	100	N = 10	A_N = 82611.74	double	1812.65	-4285.41376...
б)	100	overflow	E = 0.1f	float	–	-4285.41376...
б)	100	CA = 142	E = 0.1	double	4.16736e+29	-4285.41376...
в)	100	overflow	E = 0.0f	float	–	-4285.41376...
в)	100	CA = 375	E = 0.0	double	4.16736e+29	-4285.41376...

Вычисление истинных значений F(x) проводил в онлайн системе Wolfarm Alpha.

Выводы

Видно, что при адекватных значениях x (N/E) программа с необходимой точностью вычисляет значение функции. Если же x взять слишком большой, то может, как в случае с однократной точностью, произойти переполнение (причём кажется переменной int которая отвечает за количество итераций), и программа завершится с ошибкой. Но, что более важно, при большом значении x члены ряда становятся очень большими, знак видимо начинает теряться, быстро накапливается ошибка и происходит сильное расхождение с F(x). Выходит, что если теоретически данный ряд сходится, то на практике такой алгоритм с такими значениями неприменим для аппроксимации F(x).