Лабораторная работа №5
«Циклы с вложением»

Теоретическое введение

Ряды

1

Ряд — это сумма бесконечного числа слагаемых. Например

$$\sum_{k=0}^{\infty}\frac{1}{2^k}=\frac{1}{1}+\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\frac{1}{16}+\cdots$$

Очевидно, для вычисления такой «бесконечной» суммы потребуется также бесконечное время. Поэтому перед тем, как вычислять, зададимся некоторой величиной допустимой «погрешности» ε и будем вычислять слагаемые последовательно друг за другом до тех пор, пока очередное слагаемое не станет меньше этой «погрешности». Получится примерно следующий алгоритм для вычисления суммы ряда:

```
1) положить sum = 0
2) вычислить следующее слагаемое
3) прибавить его к переменной sum
4) если полученное слагаемое больше ε — вернуться к шагу 2
        иначе — конец вычислений
```

При реализации этого алгоритма следует помнить, что члены ряда («слагаемые») иногда могут иметь и отрицательный знак — и поэтому с величиной ε следует сравнивать их *модуль*, а не само значение (см. функцию fabs()).

2

Довольно часто в записи формулы ряда присутствует какой-нибудь дополнительный параметр (например, x), значение которого существенно влияет на результат суммирования данного ряда. А если при разных значениях x получаются разные результаты — значит сумма нашего ряда — это уже *функция* величины x. При такой постановке задачи часто требуется «*табулировать*» эту функцию. То есть — в табличной форме представить соответствие: x — f(x), где f(x) это сумма ряда при заданном значении x. Сделать это можно при помощи следующего короткого алгоритма:

```
for(double x=x_min; x<=x_max; x+=x_step){
     1) вычислить сумму ряда при текущем значении x
     2) вывести на экран (в одну строку) x и полученную сумму
}
```

В этом алгоритме x_min, x_max и x_step — это начальное и конечное значение x, а также шаг его изменения соответственно. О том, как реализовать пункт (1) данного алгоритма, см. выше.

3

При последовательном вычислении членов ряда нередко оказывается, что следующий член проще вычислить на основе уже имеющегося предыдущего, чем вычислять его «с нуля». Например (см. ряд вверху страницы), если нам уже известно значение $a_k=\frac{1}{2^k}$, то из него легко получить a_{k+1}:

$$a_{k+1}=\frac{1}{2^{k+1}}=\frac{1}{2^k}\cdot\frac{1}{2}=a_k\cdot\frac{1}{2}$$

Если обозначить отношение k+1-го члена ряда к k-му через r_k ($r_k = \frac{a_{k+1}}{a_k}$, а для приведенного выше примера ряда $r_k = \frac{1}{2}$ при любом значении k), то получим следующий, оптимизированный, алгоритм вычисления суммы ряда:

```
1) положить sum = 0
2) положить k = 0
3) вычислить первое (т.е. «нулевое») слагаемое a_0
4) прибавить его к переменной sum
5) вычислить r_k
6) вычислить следующее слагаемое: a_{k+1} = a_k · r_k
7) k = k + 1
8) если полученное слагаемое больше ε — вернуться к шагу 4
        иначе — конец вычислений
```

Простые числа

Простыми называют натуральные числа, которые не имеют целых делителей, кроме самого себя и числа 1. На простых числах, например, основана небезызвестная криптографическая система RSA. Нас же будет интересовать, как, имея некоторое число, проверить его на простоту. Следующий фрагмент псевдокода отвечает на этот вопрос:

```
ввод N — числа для проверки
int prime = 1;          // «флаг», обозначающий, что число - простое
for(int i=2; i<N; i++){
     если N делится нацело на i — положить prime = 0, конец вычислений
}
если теперь проверить значение переменной prime, и оно окажется равным
нулю – значит число составное.
если же не окажется – значит простое!
```

Задания для самостоятельного решения

Задание 1. Напишите программу для вычисления числа *e* по формуле:

$$e = \sum_{k=0}^{\infty} \frac{1}{k!}$$

Вычисление суммы продолжать до тех пор, пока очередное слагаемое по модулю не станет меньше $\varepsilon = 10^{-4}$.

Задание 2. Напишите программу для вычисления значения экспоненциальной функции e^x в заданной точке *x* по формуле:

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$$

Вычисление суммы продолжать до тех пор, пока очередное слагаемое по модулю не станет меньше $\varepsilon = 10^{-4}$. Конкретное значение *x* вводится с клавиатуры.

Задание 3. Табулировать экспоненциальную функцию e^x для значений $x \in [0,2]$ с шагом 0.1. Результат вывести в виде следующей таблицы:

x	sum	exp	delta
0	1	1	0
0.1	1.1052	1.105171	0.000029
0.2	_____	_____	_____
…	_____	_____	_____
2	_____	_____	_____

(`x` — значение аргумента функции; `sum` — вычисленное Вашей программой значение экспоненты; `exp` — значение экспоненты данного аргумента, полученное при помощи библиотечной функции `exp();` `delta` — разность между вычисленным с помощью ряда и «библиотечным» значением).

Величину ε вводить с клавиатуры.

Задание 4. Выполните свой вариант индивидуального задания №5.

Задание 5. Воспользуйтесь программой из «Задания 3» лабораторной работы №4 (поиск максимальной степени числа, не превосходящей заданного N), а также блок-схемой из задания 4 лабораторной работы №2 (запись заданного числа в произвольной системе счисления) для того, чтобы написать полноценную программу перевода числа в систему счисления по основанию k. На входе программы — число для перевода N и основание системы счисления k. На выходе представление введенного числа N в k-чной системе счисления. Программа должна работать при значениях k от 2 до 16 включительно.

Задание 6. Написать программу для проверки натурального числа N на простоту. N вводится с клавиатуры.

Задание 7. Написать программу для поиска всех простых чисел на интервале 2..N. N вводится с клавиатуры.

Оценивание

Содержание отчета

1. Условия заданий.
2. Исходные коды и скриншоты выполнения программ №3,4,5,7.
3. **Исходные коды программ из заданий 1,2,6 можно отдельно не выписывать, а объединить с №3 и №7 соответственно.**

Баллы за задания

Задание	Баллы	
1	обязательное	1
2	обязательное	1
3	обязательное	1
4		3
5		1
6	обязательное	1
7	обязательное	1
Всего		**9**

Бонусы

Досрочная сдача: +1 балл.
Несвоевременная сдача: макс. балл уменьшается на 2 за каждую просроченную неделю.

Приложене 1: индивидуальное задание №5

Дано натуральное число N. Вычислить сумму конечного отрезка ряда:

1) $\sum_{k=1}^{N} \frac{(-1)^k}{(2k+1)^2}$

2) $\sum_{k=1}^{N} \frac{(-1)^k}{(2k+1)k}$

3) $\sum_{k=1}^{N} \frac{(-1)^{k+1}}{(k+1)k}$

4) $\sum_{k=1}^{N} \frac{(-1)^k (k+1)}{k!}$

5) $\sum_{k=1}^{N} \frac{k!}{\frac{1}{2}+\frac{1}{3}+\frac{1}{k+1}}$

Вычислить бесконечную сумму с заданной (с клавиатуры) точностью E (E>0). Считать, что требуемая точность достигнута, если очередное слагаемое оказалось по модулю меньше, чем E:

6) $\sum_{k=1}^{\infty} \frac{1}{(2*k+1)^2}$

7) $\sum_{k=1}^{\infty} \frac{(-1)^k}{(2*k+1)*k}$

8) $\sum_{i=0}^{\infty} \frac{(-2)^i}{i!}$

9) $\sum_{i=1}^{\infty} \frac{1}{i*(i+1)}$

Табулировать функции F и S при заданной точности E, с указанным шагом и диапазоном значений по x:

10) $F(x) = \sum_{k=1}^{\infty} (-1)^{k+1} \frac{2^{2k-1} x^{2k}}{(2k)!}$ $\quad E = 10^{-6}$

$S(x) = F(x) - \sin^2 x;$ $\quad x = 0(1)10$

11) $F(x) = \sum_{k=1}^{\infty} (-1)^{k+1} \frac{3^{2k+1} - 3}{(2k+1)!} x^{2k+1}$ $\quad E = 10^{-6}$

$S(x) = F(x) - 4\sin^3 x$ $\quad x = 0(0.1)0.9$

12) $F(x) = \sum_{k=0}^{\infty} (-1)^{k} \frac{3^{2k} + 3}{(2k)!} x^{2k+1}$ $\quad E = 10^{-6}$

$S(x) = F(x) - 4\cos^3 x$ $\quad x = 0(1)10$

13) $F(v,x) = \sum_{k=0}^{\infty} (-1)^{k} \frac{x^{v+k}}{k!(v+k)}$ $\quad E = 10^{-4}$

$S(v,x) = F(v+1,x) - vF(v,x) + x^v e^{-x}$ $\quad x = 0(0.4)4$, $v = 1;1.7;2.5$

14) $F(x) = \sum_{k=1}^{\infty} (-1)^{k+1} k x^{k-1} \sum_{n=1}^{k} \frac{1}{n}$ $\quad E = 10^{-4}$

$S(x) = F(x) - \frac{1 - \ln(1+x)}{(1+x)^2}$ $\quad x = 0(0.1)0.8$

Приложение 2: пример выполнения задания на ряды

Задание — табулировать функции F и S:

$$F(x)=\sum_{k=1}^{\infty}(-1)^{k+1}\frac{3^{2k+1}-3}{(2k)!}x^{2k} \qquad E=10^{-6}$$

$$S(x)=F(x)-6\sin x\sin 2x \qquad x=0(0.1)2$$

Решение.

Обратите особенное внимание на подсчет суммы ряда в функции S: каждое новое слагаемое вычисляется не «с нуля», а путем домножения уже имеющегося слагаемого (a) на некоторое выражение (см. пункт 3 в «Теоретическм введении» в ряды).

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

double S(double x, double eps = 1e-6){
     double sum = 0;
     int k = 1;
     double a = 12.0 * pow(x, 2);
     while(abs(a)>=eps){
          sum += a;
          a *= -pow(x,2)/(2.0*k+2)/(2.0*k+1)
              *(pow(3.0,2*k+2)-1)/(pow(3.0,2*k)-1);
          k++;
     }//while
     return sum;
}
int main(int argc, char* argv[])
{
     printf("x\t\ts\n");
     for(double x=0; x<=2.05; x+=0.1){
          printf("%lf\t%lf\n", x, S(x) - 6*sin(x)*sin(2*x));
     }
     return 0;
}
```