Филиал «Котельники» государственного бюджетного

образовательного учреждения высшего образования

Московской области «Университет «Дубна»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**по курсовой работе по дисциплине**

**«Программирование на языке высокого уровня»**

**ВАРИАНТ No15**

Выполнил:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

студент группы ИВТ-11 Рождественский Д. М.

Проверил:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

доцент, к.т.н. Артамонов Ю. Н.

Котельники – 2018

**Оглавление**

[Раздел 1 - Разработка численных алгоритмов. Часть 1 - Суммирование рядов и вычисление элементарных функций. 3](#_Toc514700627)

[Раздел 1. Часть 2 Приближенные методы нахождения корней уравнения.](#_Toc514700628) 7

Раздел 2. Разработка игровой программы…… 24

Заключение. 36

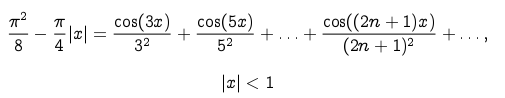
Список использованных источников. 37

Приложения 38

**Раздел 1 — Разработка численных алгоритмов. Часть 1 — суммирование рядов и вычисление элементарных функций.**

**Задача номер 5**

Дано:



Необходимо реализовать вычисление по этому представлению и проверить насколько быстро сходится процесс вычислений (сколько слагаемых надо взять для получения результата с заданной точностью при различных значениях х).

Решение:

#include <stdio.h> // Подключение стандартной библиотеки ввода\вывода stido.h

#include <stdlib.h> // Подключение библиотеки stdio.h

#include <math.h> // Подключение математической библиотеки math.h (для работы некотопрых вещей нужно компилировать с ключом -lm)

float sumR(float, int); // Прототип функции

main(){ // Главная функция - main выполняется при старте программы

float x, lPart, E; // Объявление переменных с плавающей точкой

int n=1; // int - целочисленное число

printf("Введите х такое, что |x| < 1\n x = "); // Вывод строки

while(1){ // Бесконечный цикл, который завершается, если правильно введены данные (если неверные - выводится сообщение об ошибке и требуется ввести заново)

scanf("%f", &x); // Ввод значения типа float с клавиатуры (присваивается переменной х)

if(abs(x) < 1) // Если |x| < 1, то цикл прерывается

break;

printf("Неверное значение х, введите ещё раз\n x = "); // Если |x| >= 1, то выводится сообщение об ошибке и требуется ввести х заново

}

printf("Введите точность вычисления выражения\nE = ");

scanf("%f", &E); // Вводится точность вычислений

E=fabs(E); // Обрабатывается ошиба при E<0

lPart = powf(M\_PI, 2)/8 - M\_PI/4\*abs(x); // Левая часть выражения

while(abs(lPart) - abs(sumR(x, n)) > E) n++; // Проверяется сколько слагаемых надо взять, чтобы вычислить выражение с заданной точностью

printf("Нужно взять %d слагаемых\n", n); // Выводится количество слагаемых

return 0; // Возвращение 0 в конце функции main используется для проверки правильности работы программы

}

float sumR(float x, int n){ // Функция, вычисляющая значение правой части при заданных х и n

float s=0; // Сумма

int i; // Переменная, используемая как счётчик в цикле

for(i=0; i<n; i++){ // Цикл, выполняющийся пока i<n, i меняется на +1 каждую итерацию (а изначально i = 0)

s+=cos(3\*x/180)/powf(2\*n+1, 2); // К уже имеющейся сумме прибавляются слагаемые

}

return s; // Функция возвращает сумму элементов

}

**Описание входных данных**

Тип входных данных:

float

Ограничения:

|x|<1

Обработка ошибочного ввода:

Цикл ввода числа «х» будет запускаться заново с выводом сообщения об ошибке, пока не будут введены верные данные.

Если введено E меньшее 0, то ему будет присвоено его значение, взятое по абсолютной величине.

**Описание выходных данных**

Выводится число типа int, означающее количество слагаемых, которое необходимо взять для заданной точности.

**Блок-схема реализуемого алгоритм**

Начало

N=1

Ввод х

|x|<1

Нет

Да

Ввод E

E=|E|



|lPart|-|sumR(x,n)| > E

n++

Вывод n

Конец

Да

Блок-схема 1.1.1

Блок-схема функции sumR(float, float)

float x; float n;

s=0

I=0; i<n; i++;



return s;

Блок-схема 1.1.2

**Расчётная таблица соответствия входных и выходных данных**

|  |  |  |
| --- | --- | --- |
| Входные данные | | Вывод |
| x | E | Количество слагаемых |
| 0.1 | 0.000000001 | 10 |
| 0.0000002 | 0.000000001 | 9 |
| -0.0003 | 0.000000001 | 9 |
| 0 | 0.000000001 | 9 |

Таблица 1.1.1

**Выводы по результатам тестирования программного приложения на расчётных примерах:**

При большем |x| с одинаковой точностью требуется большее количество слагаемых.

**Раздел 1 — Разработка численных алгоритмов. Часть 2 — приближённые методы нахождения корней уравнения.**

Дано:

Уравнения:

1.2.1

1.2.2

1.2.3

1.2.4

1.2.5

1.2.6

Надо реализовать метод деления пополам и метод Ньютона для решения этих уравнений.

**Решение:**

**#include <stdio.h> // Стандартная библиотека ввода/вывода итд stdio.h**

**#include <stdlib.h> // Подключение библиотеки stdlib.h**

**#include <math.h> // Библиотека с математическими функциями math.h, для работы некоторых функций нужно компилировать с ключом -lm**

**#include <time.h> // Подключение библиотеки time.h**

**double method\_1(int, double, double, double, double, double); // Прототип функции вычисляющей корни по заданному методу**

**double method\_2(int, double, double, double, double); // Прототип функции вычисляющей корни по заданному методу**

**double f(int, double, double, double); // Прототип функции, возвращающей результаты уравнения**

**double df(int, double, double, double); // Прототип функции, возвращающей значение производной функции в данной точке**

**void inputAndFails(int \*, int \*, double \*, double \*, double \*, double \*, double \*, double \*); // Прототип функции, в которой осуществляется ввод данных и проверка их на ошибки**

**// Функция main() выполняется при старте программы**

**main(){**

**srand(time(NULL)); // Генерация случайных чисел будет зависеть от времени**

**// Объявление переменных типа int для выбора метода и уравнения**

**// и типа float, содержащих значения, необходимые для вычислений**

**int findMethod, eqNum;**

**double k1, k2, E, a, b, result, xn;**

**// Запуск функции inputAndFails() с ссылками на переменные, объявленные ранее, чтобы она могла их менять**

**inputAndFails(&findMethod, &eqNum, &k1, &k2, &E, &a, &b, &xn);**

**// Если пользователь выбрал метод 1, то присваивает значение, возвращаемое функцией method\_1, переменной result**

**// В ином случае присваивает переменной result значение, возвращаемое функцией method\_2()**

**if(findMethod == 1)**

**result = method\_1(eqNum, k1, k2, E, a, b);**

**else**

**result = method\_2(eqNum, k1, k2, E, xn);**

**// Выводит результат**

**printf("x = %f\n",result);**

**// Возвращает значение 0, используется как проверка на ошибки**

**return 0;**

**}**

**// Функция, возвращающая значение, полученное с помощью 1 метода (либо -1 в случае слишком большого количества итераций)**

**double method\_1(int eN, double k1, double k2, double E, double a, double b){**

**// Объявление локальных переменных типа double**

**double c, fc,fa;**

**// Объявление локальной переменной iter и присвоение ей значения 0**

**int iter=0;**

**// Цикл do{}while() выполняется минимум 1 раз, а затем пока выполняется условие**

**do{**

**// Формула получения точки "с"**

**c=(a+b)/2;**

**// Если с равно а или b, то цикл прерывается**

**if(c == a || c == b)**

**break;**

**// переменным fc и fa присваивается значение функции в точках c и a соответственно**

**fc=f(eN, k1, k2, c);**

**fa=f(eN, k1, k2, a);**

**// Если fc и fa >= 0 или fc и fa <= 0, a=c, в противном случае b=c**

**if(fc>=0 && fa>=0 || fc<=0 && fa<=0)**

**a=c;**

**else**

**b=c;**

**iter++; // Счётчик итераций**

**// Если проходит слишком много итераций, функция завершается со значением -1**

**if(iter==200)**

**return -1;**

**} while(fabs(b-a) > E); // Условие цикла: |b-a| > E**

**// Выводит число итераций**

**printf("iter = %d\n", iter);**

**// Возвращает значение с**

**return c;**

**}**

**// Функция, возвращающая значение, полученное с помощью 2 метода (либо -1 в случае слишком большого количества итераций)**

**double method\_2(int eqNum, double k1, double k2, double E, double xn){**

**// Объявление локальных переменных типа double**

**double ifEnd, x,rx[3];**

**// Объявление локальной переменной iter и присвоение ей значения 0**

**int iter=0;**

**do{**

**// Формула получения точки х**

**x = xn - f(eqNum, k1, k2, xn)/df(eqNum, k1, k2, xn);**

**// массив rx состоит из значений, используемых в if, чтобы if выглядел читаемо**

**rx[0]=fabs(x-xn);**

**rx[1]=fabs(f(eqNum, k1, k2, xn-E));**

**rx[2]=fabs(f(eqNum, k1, k2, xn+E));**

**// Проверяется условие конца цикла и в случае необходимости (если одинаковые знаки на концах отрезка), E уменьшается**

**if(rx[0]<E && ((rx[1]>0 && rx[2]>0) || (rx[1]<0 && rx[2]<0))))**

**E/=10;**

**// Если проходит слишком много итераций, функция завершается со значением -1**

**if(iter==200)**

**return -1;**

**// присваивает переменной xn значение переменной x**

**xn=x;**

**iter++; // Счётчик итераций**

**}while(rx[0]>E); // Условие цикла |x-xn|>E**

**// Выводит число итераций**

**printf("iter = %d\n", iter);**

**// Возвращает значение с**

**return x;**

**}**

**// Функция, возвращающее значение производной в данной точке для нужной функции**

**double df(int eqNum, double k1, double k2, double x){**

**// Опретор switch обрабатывает несколько случаев - при разных значениях eqNum будут выполнятся разные действия**

**switch(eqNum){**

**// Если eqNum == 1, то возвращается значение производной в точке первой функции**

**case 1:**

**return 1/(x\*logf(2));**

**case 2:**

**return 4\*powf(x,3)+k1\*3\*x\*x-k2;**

**case 3:**

**return 5\*powf(x,4)+2\*k1\*x;**

**case 4:**

**return 2\*x+k1;**

**case 5:**

**return 2/(x\*logf(2));**

**case 6:**

**return 3\*x\*x+k1;**

**}**

**}**

**// Функция, возвращающее значение выбранной функции в данной точке**

**double f(int eqNum, double k1, double k2, double x){**

**switch(eqNum){**

**// Если eqNum == 1, то возвращается значение функции в точке первой функции**

**case 1:**

**return log2f(k1\*x) - k2;**

**case 2:**

**return powf(x,4) + k1\*powf(x,3) - k2\*x;**

**case 3:**

**return powf(x,5) + k1\*powf(x,2) - k2;**

**case 4:**

**return powf(x, 2) + k1\*x + k2;**

**case 5:**

**return log2f(x\*x\*k1) + k2;**

**case 6:**

**return powf(x,3)+k1\*x+k2;**

**}**

**}**

**// Функция, отвечающая за ввод данных пользователем**

**void inputAndFails(int \*fM, int \*eN, double \*k1, double \*k2, double \*E, double \*a, double \*b, double \*xn){**

**// Объявление локальных переменных типа double**

**double cache,fa,fb;**

**// Предлагает пользователю выбрать метод и считывает введённые им данные**

**printf("Введите номер способа, которым хотите решить это уравнение (1, 2)\n");**

**scanf("%d", fM);**

**// Если пользователь ввёл неверные данные, то метод выбирается случайным образом**

**if(\*fM!=1 && \*fM!=2){**

**printf("Неверный номер, будет сгенерирован случайный\n");**

**\*fM = rand()%2+1; // Присвоение переменной fM случайного значения от 1 до 2**

**printf("Способ %d\n", \*fM);**

**}**

**// Выбор уравнения**

**printf("Введите номер уравнения (1, 2, 3 - уравнения 3, 5, 6 из заданий соответсвенно\n4, 5, 6 - мои уравнения)\n");**

**scanf("%d", eN);**

**// Если пользователь ввёл неверные данные, то уравнение выбирается случайным образом**

**if(\*eN<1 || \*eN>6){**

**printf("Неверный номер, будет сгенерирован случайный\n");**

**\*eN = rand()%6+1; // переменной o присваивается случайное значение от 1 до 6 вклюичтельно**

**printf("Будет решаться уравнение %d\n", \*eN);**

**}**

**// Ввод k1**

**while(1){ // Бесконечный цикл**

**printf("Введите коэфиценты уравнение\n k1 = "); // Вывод строкт**

**scanf("%lf", k1); // Ввод числа с плавающей точкой с клавиатуры**

**if(\*k1==0 && (\*eN==1 || \*eN == 5)){ // Если пользователеь ввёл неверно k1, выводит сообщение об ошибке**

**printf("k1 не может равняться 0\n");**

**}**

**else // Если к1 введено правильно, цикл прерывается**

**break;**

**}**

**// Ввод k2**

**printf(" k2 = "); // Вывод строки**

**scanf("%lf", k2); // Ввод с клавиатуры**

**// Ввод E**

**printf("Введите точность решения уравнение E\n E = ");**

**scanf("%lf", E);**

**\*E = fabs(\*E); // На случай, если будет введено E<0**

**// Ввод a, b или xn, в зависимости от метода**

**if(\*fM == 2){ // Если 2 метод - то вводится xn**

**printf("Введите xn\n");**

**scanf("%lf", xn);**

**} else { // Если 1 метод - вводится a и b**

**// Бесконечный цикл**

**while(1){**

**// Ввод a и b**

**printf("Введите a и b\n a = ");**

**scanf("%lf", a);**

**printf(" b = ");**

**scanf("%lf", b);**

**// Если a > b, то меняю их местами**

**if(\*a > \*b){**

**cache = \*a;**

**\*a = \*b;**

**\*b = cache;**

**}**

**// Если выбрано уравнение 1 или 5, то проверяется случай, когда введены a и b в местах, где функция не существует**

**if((\*eN == 1 || \*eN == 5) && (\*k1<0 && (\*a>=0 || \*b>=0) || \*k1>0 && (\*a<=0 || \*b<=0))){**

**printf("Должно соблюдаться условие:\nЕсли к1 < 0, a и б < 0\nЕсли к1 > 0, то а и б > 0\n");**

**continue; // Переходит к следующей итерации цикла без выполнения строк ниже**

**}**

**// Если a == b, запускается новая итерация цикла**

**if(\*a == \*b){**

**printf("Они не могут быть равны\b");**

**continue;**

**}**

**// Локальным переменным fa и fb присваивается значение функции в точках a и b соответсвенно, чтобы if был читаемей**

**fa=f(\*eN, \*k1, \*k2, \*a);**

**fb=f(\*eN, \*k1, \*k2, \*b);**

**if((fa > 0 && fb > 0) || (fa < 0 && fb < 0)){ // Если fa и fb имеют один знак, то переходит к следующей итерации цикла**

**printf("Неподходящие а, b, либо уравнение не имеет решений\n");**

**continue;**

**}**

**break; // Прерывает цикл**

**}**

**}**

**}**

**Описание входных данных**

Тип входных данных: int для выбора метода и уравнения и double для задания всех значений.

Порядок:

1. Метод (переменные fM и findMethod в разных функциях).
2. Номер уравнения (eN и eqNum в разных функциях).
3. Коэффициенты при уравнении a и b (k1 и k2 соответственно).
4. Точность (E).
5. Начальные значения.
   1. 1 способ — концы отрезка [a,b] (переменные a и b).
   2. 2 спсоб — xn (переменная xn).

Обработка ошибочного ввода:

Если пользователь ввёл:

1. Неверный номер способа решения и/или уравния, то будет использован случайный.
2. Коэффициенты для уравнения 1 или 5, появится сообщение об этом и потребуется ввести данные заново.
3. Отрицательную точность, возьмётся её модуль.
4. a>b, они поменяются местами.
5. Неверное а или b, появится сообщение об ошибке и потребуется ввести данные заново.

Все введённые данные будут в таблицах 1.2.1-1.2.12

**Описание выходных данных**

Выводится число итераций тип int и результат тип double.

Уравнение 1:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Метод | k1 | k2 | E | a | b | xn | iter | x |
| 1 | 1 | 5 | 0.000001 | 0.00001 | 100 |  | 27 | 32.000023 |
| 2 | 1 | 5 | 0.000001 |  |  | 10 | 6 | 32.000003 |

Таблица 1.2.1.

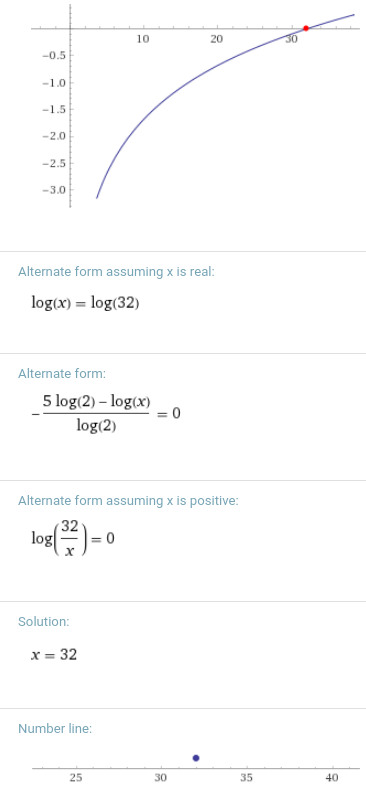


Рисунок 1.2.1.

Уравнение 2:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Метод | k1 | k2 | E | a | b | xn | iter | x |
| 1 | 0 | -3 | 0.0001 | 1 | 2 |  | 14 | 1.442200 |
| 2 | 0 | -3 | 0.0001 |  |  | 2 | 5 | 1.442250 |

Таблица 1.2.2.

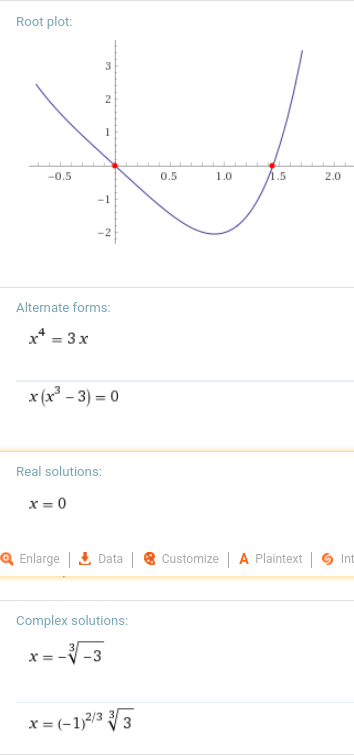


Рисунок 1.2.2.

Уравнение 3:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Метод | k1 | k2 | E | a | b | xn | iter | x |
| 1 | 3 | 9 | 0.0000001 | 1 | 2 |  | 24 | 1.309723 |
| 2 | 3 | 9 | 0.0000001 |  |  | 2 | 200 | -1 |

Таблица 1.2.3.

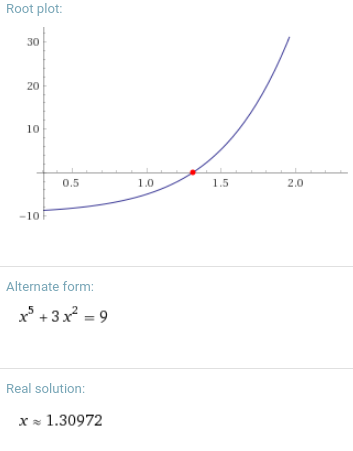


Рисунок 1.2.3.

Уравнение 4:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Метод | k1 | k2 | E | a | b | xn | iter | x |
| 1 | -1 | -3 | 0.001 | 1 | 4 |  | 12 | 2.302979 |
| 2 | -1 | -3 | 0.001 |  |  | 4 | 5 | 2.302776 |

Таблица 1.2.4.

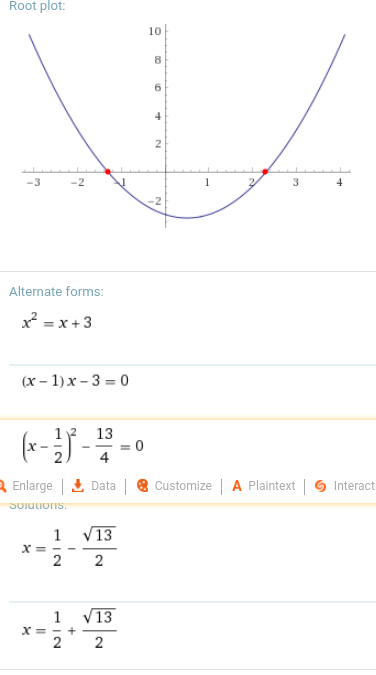
****

Рисунок 1.2.4

Уравнение 5:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Метод | k1 | k2 | E | a | b | xn | iter | x |
| 1 | 1 | -3 | 0.000001 | 2 | 4 |  | 21 | 2.828427 |
| 2 | 1 | -3 | 0.000001 |  |  | 4 | 5 | 2.828427 |

Таблица 1.2.5.

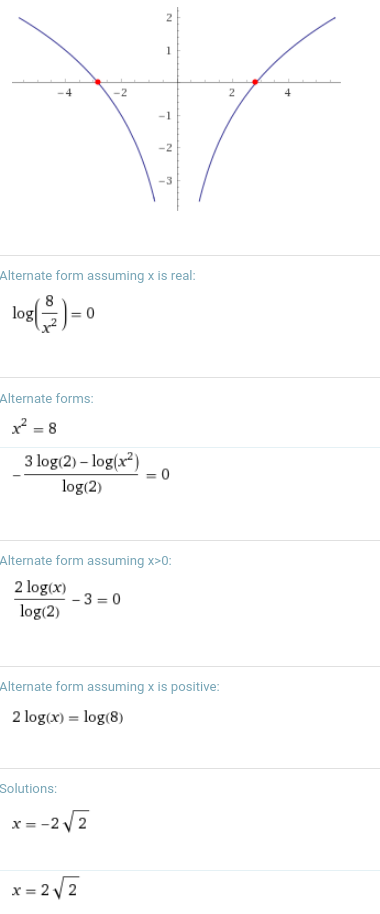
****

Рисунок 1.2.5.

Уравнение 6:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Метод | k1 | k2 | E | a | b | xn | iter | x |
| 1 | 5 | 4 | 0.0001 | -1.5 | -0.5 |  | 14 | -0.724060 |
| 2 | 5 | 4 | 0.0001 |  |  | -0.5 | 4 | -0.724076 |

Таблица 1.2.6.

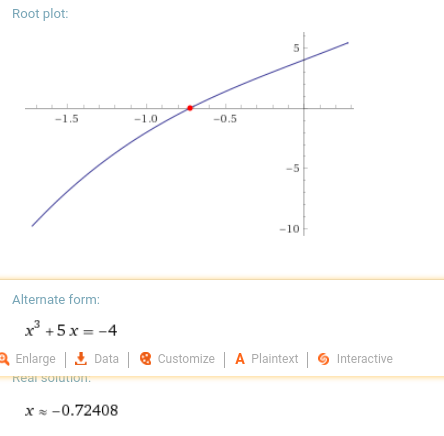


Рисунок 1.2.6.

**Блок-схемы**

Начало

findMethod == 1

result = method\_1(eqNum, k1, k2, E, a, b)

result = method\_2(eqNum, k1, k2, E, xn)

Вывод результата

Конец

Да

Нет

Ввод findMethod,

eqNum, k1, k2, E, a, b, xn

Блок-схема 1.2.1.

int eqNum, double k1, double k2, double E, double xn

|b-a|>E

result = method\_1(eqNum, k1, k2, E, a, b)

result = method\_2(eqNum, k1, k2, E, xn)

Вывод числа итераций

Конец

Нет

Да

x = xn - f(eqNum, k1, k2, xn)/df(eqNum, k1, k2, xn);

rx[0]=fabs(x-xn);

rx[1]=fabs(f(eqNum, k1, k2, xn-E));

rx[2]=fabs(f(eqNum, k1, k2, xn+E));

rx[0]<E && (rx[1]>0 && rx[2]>0)

или

rx[0]<E && (rx[1]<0 && rx[2]<0)

E/=10

Нет

Да

iter==200

return -1

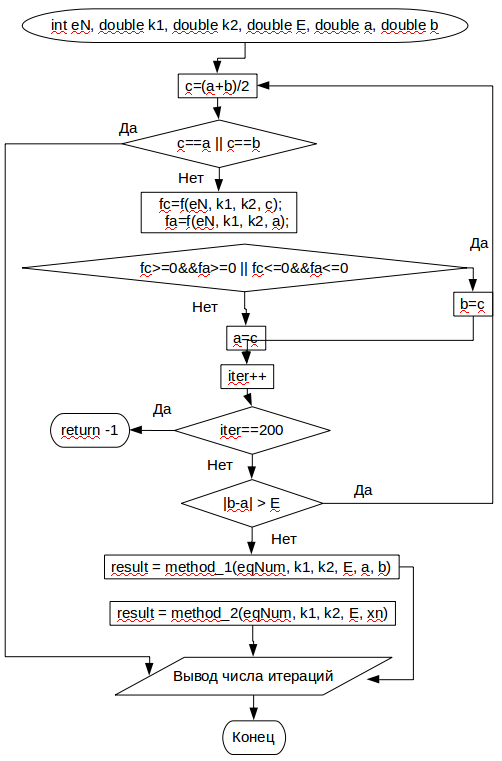
Да

Нет

x=xn;

iter++;

Блок-схема 1.2.2. (Метод 2)

****

Блок-схема 1.2.3.

**Расчётая таблица соответствия входных и выходных данных**

Обозначения:

k1, k2 — коэффициенты в уравнениях.

E — точность.

a и b — концы отрезка [a,b].

xn — начальная точка для 2 способа.

iter — число итераций.

х — результат.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Входные данные | | | | | | | | Вывод | |
| Метод | Ур-е | k1 | k2 | E | a | b | xn | iter | x |
| 1 | 1 | 1 | 5 | 0.000001 | 0.00001 | 100 |  | 27 | 32.000023 |
| 2 | 1 | 1 | 5 | 0.000001 |  |  | 10 | 6 | 32.000003 |
| 1 | 2 | 0 | -3 | 0.0001 | 1 | 2 |  | 14 | 1.442200 |
| 2 | 2 | 0 | -3 | 0.0001 |  |  | 2 | 5 | 1.442250 |
| 1 | 3 | 3 | 9 | 0.0000001 | 1 | 2 |  | 24 | 1.309723 |
| 2 | 3 | 3 | 9 | 0.0000001 |  |  | 2 | 200 | -1 |
| 1 | 4 | -1 | -3 | 0.001 | 1 | 4 |  | 12 | 2.302979 |
| 2 | 4 | -1 | -3 | 0.001 |  |  | 4 | 5 | 2.302776 |
| 1 | 5 | 1 | -3 | 0.000001 | 2 | 4 |  | 21 | 2.828427 |
| 2 | 5 | 1 | -3 | 0.000001 |  |  | 4 | 5 | 2.828427 |
| 1 | 6 | 5 | 4 | 0.0001 | -1.5 | -0.5 |  | 14 | -0.724060 |
| 2 | 6 | 5 | 4 | 0.0001 |  |  | -0.5 | 4 | -0.724076 |

**Выводы по результатам тестирования программного приложения на расчётных примерах:**

В среднем метод касательных точнее метода деления отрезка пополам и к тому же требует меньшее число итераций, поэтому в большинстве случаев целесообразнее использовать его.

Однако оба этих метода не учитывают случаев, когда есть 2 или более корня (только если самому задавать разные данные).

**Раздел 3 — Разработка игровой программы.**

**Формулировка:**

Устный счет: используя арифметические операции умножения, сложения, вычитания, деления, возведения в степень, программа случайно генерирует арифметическое выражение, используя в случае необходимости скобки, вычисляет его значение и предлагает человеку на время такжевыполнить вычисление. Предложите и реализуйте не менее трех вариантов расширения функциональности этой игры(например, регулируется длина выражения, используемые арифметические операции, предусматривается система бонусов, если игрок угадывает решение и т. п.).

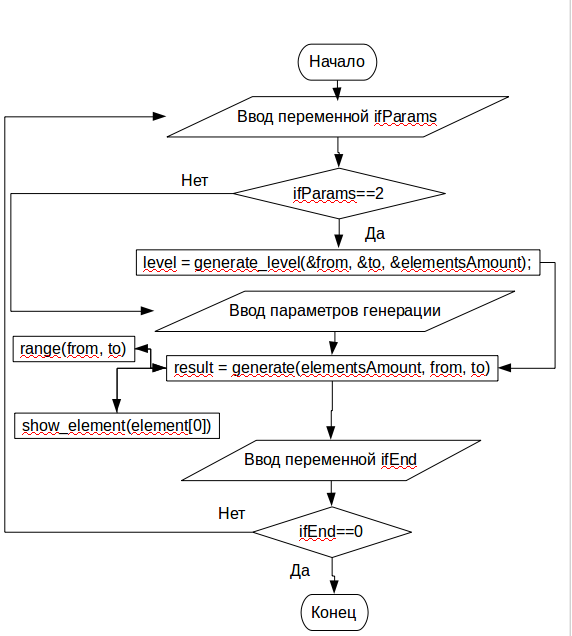
Расширения функциональности:

1. Есть 6 уровней сложности.
2. Есть возможность задавать вручную:
   1. Количество чисел в выражении.
   2. Отрезок на числовой оси, в пределах которого будут генерируемые числа.
3. Играть можно сколько угодно раз подряд; есть счётчик количества игр.
4. Чем сложнее выражение решает пользователь, тем больше ему дают очков, за проигрыш в любом случае снимается только 1 очко.

**Декомпозиця задачи на подзадачи:**

1. Присвоение параметрам для генерации значений:
   1. Генерация параметров в зависимости от уровня сложности.
   2. Задание пользователем параметров вручную.
2. Генерация самого выражения.
3. Реализация зачисления очков, подсчёта количества игр, возможности играть сколько угодно раз.

**Описание взаимодействия подпрограмм при работе основной программы в виде блок-схемы**



Блок-схема 2.1.1.

generate\_level()

int \*fP, int \*tP, int \*elAmP

Выбор уровня сложности

(переменная level)

level

\*fP=0;\*tP=10;\*elAmP=2;

\*fP=0;\*tP=100;\*elAmP=3;

\*fP=-100;\*tP=100;\*elAmP=3;

\*fP=-500;\*tP=500;\*elAmP=5;

\*fP=-1000;\*tP=1000;\*elAmP=10;

\*fP=-1000;\*tP=1000;\*elAmP=100;

0

1

2

3

4

5

\*fP=rand()%100-50;\*tP=rand()%100+51;

\*elAmP=rand()%6+2;

default

Return level

Блок-схема 2.1.2.

/\* Выбор уровня сложности - от 0 до 5.

В зависимости от выбранного уровня будут заданы from, to и elementsAmount.

Функция вернёт выбранный уровень сложности для подсчёта получаемых игроком очков.

\*/

int generate\_level(int \*fP, int \*tP, int \*elAmP){

int level;

printf("Выберите уровень сложности\n 0 - very easy\n 1 - easy\n 2 - medium\n 3 - hard\n 4 - impossible\n 5 - you can't do this\n");

scanf("%d", &level);

switch(level){

default:

level = 0;

\*fP=rand()%100-50;

\*tP=rand()%100+51;

\*elAmP=rand()%6+2;

break;

case 0:

\*fP=0;

\*tP=10;

\*elAmP=2;

break;

case 1:

\*fP=0;

\*tP=100;

\*elAmP=3;

break;

case 2:

\*fP=-100;

\*tP=100;

\*elAmP=3;

break;

case 3:

\*fP=-500;

\*tP=500;

\*elAmP=5;

break;

case 4:

\*fP=-1000;

\*tP=1000;

\*elAmP=10;

break;

case 5:

\*fP=-1000;

\*tP=1000;

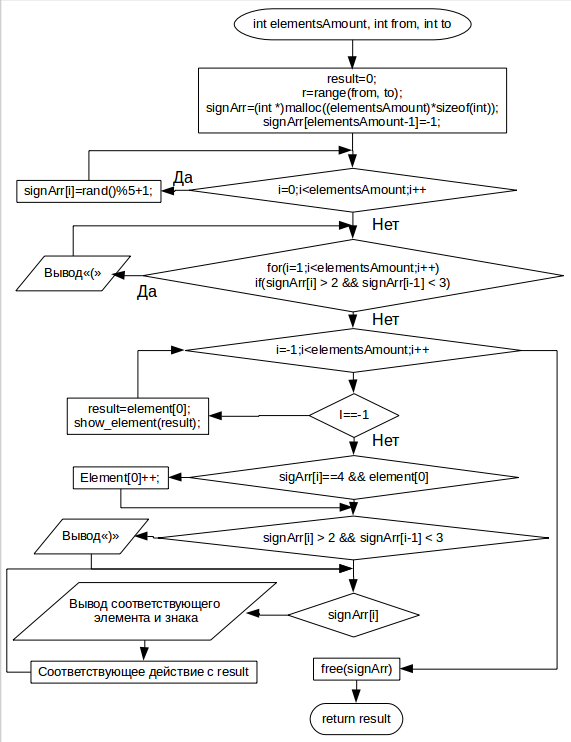
\*elAmP=100;

printf("Ну ладно, попробуем (Подсказка: если не можете найти 0, посчитайте минусы)\n\n");

}

return level;

}

generate()

Блок-схема 2.1.3.

// Генерация выражения c заданными параметрами: количество чисел в выражении; отрезок на числовой оси, в пределах которого будут генерируемые числа.

float generate(int elementsAmount, int from, int to){

float result=0, element[2];

int i, \*signArr, r;

// Переменной R присваивается число, получаемое в функции range()

r=range(from, to);

// под signArr выделяется пямять = количеству элементов в массиве, умноженному на размер типа int

signArr = (int \*)malloc((elementsAmount)\*sizeof(int));

// Это будет использоваться для последней итерации цикла

signArr[elementsAmount-1]=-1;

/\* Генерация чисел от 1 до 5

1 - сложить

2 - вычесть

3 - умножить

4 - разделить

5 - возвести в степень (мб извлечь корень)

\*/

for(i = 0; i < elementsAmount-1; i++)

signArr[i] = rand()%5+1;

// Вывод скобок в начале выражения, чтобы действия шли в правильном математическом порядке

for(i = 1; i < elementsAmount; i++){

if(signArr[i] > 2 && signArr[i-1] < 3)

printf("(");

}

// Вывод всех элементов выражения

for(i = -1; i < elementsAmount; i++){

// Генерируется элемент от from до to, для этого используется формула:

element[0]=rand()%r - abs(from);

// Вывод первого элемента выражения

if(i==-1){

result=element[0];

if(signArr[0]>2)

printf("(");

show\_element(result);

continue;

}

// Если следующее действие деление, то это помогает избежать случаев деления на 0

if(signArr[i] == 4 && element[0] == 0 && i>-1)

element[0]++;

// Закрывающая скобка для правильного математического порядка действий

if(signArr[i] > 2 && signArr[i-1] < 3)

printf(")");

switch(signArr[i]){

case 1:

printf(" + ");

result+=element[0];

break;

case 2:

printf(" - ");

result-=element[0];

break;

case 3:

printf(" \* ");

result\*=element[0];

break;

case 4:

printf(" / ");

result/=element[0];

break;

case 5:

// Ввиду того, что довольно сложно считать в уме большие степени, а также того, что в результате получатся огромные числа,

// я сделал только степени 0, 0.5, 2

switch(rand()%3){

case 0:

element[0]=0;

case 1:

element[0]=0.5;

case 2:

element[0]=2;

}

printf(" ^ ");

result=powf(result, element[0]);

break;

case -1:

printf(" = ? \n");

}

// Обрабатывается последняя итерация

if(signArr[i]==-1)

break;

show\_element(element[0]);

// Это используется для расставления скобок

element[1]=element[0];

}

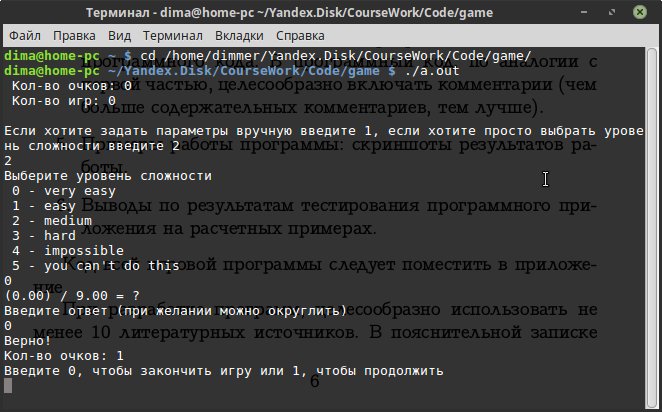
// освобождается память и возвращается результат

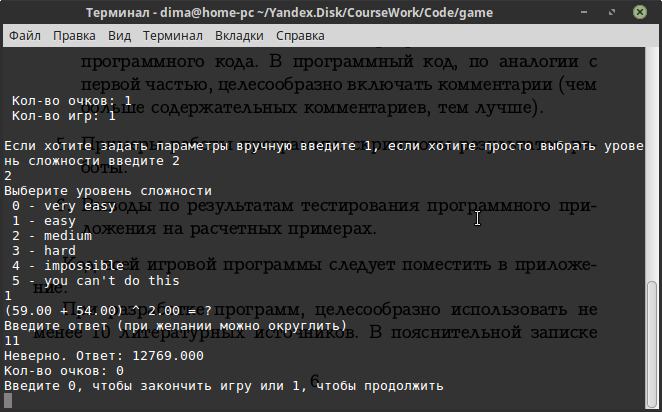
free(signArr);

return result;

}

**Примеры работы программы**

Рисунок 2.1.1.

Рисунок 2.1.2.

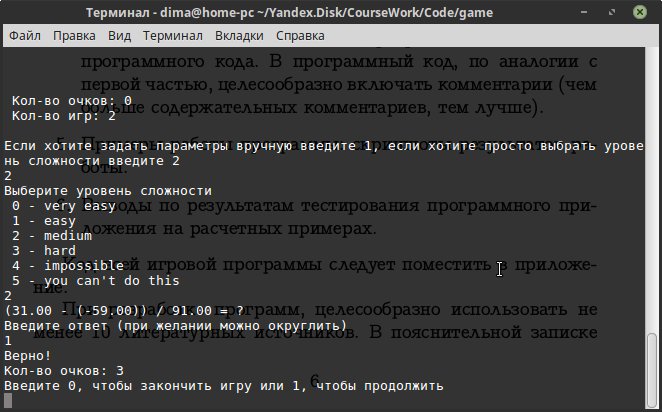
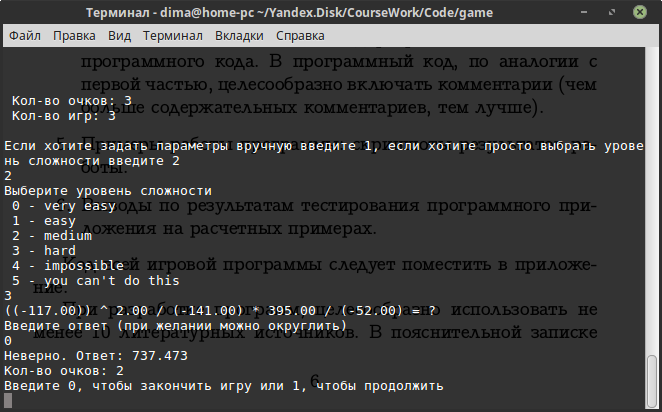
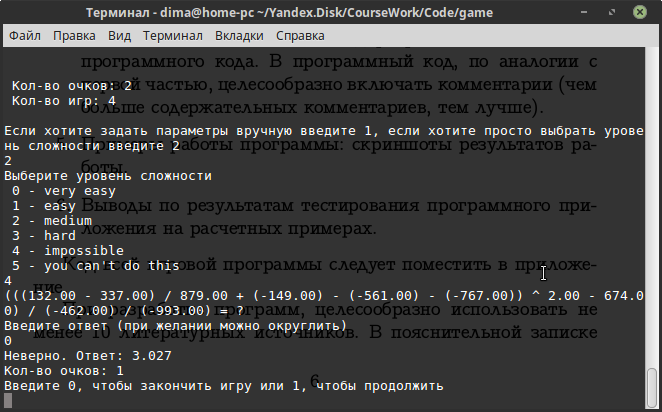
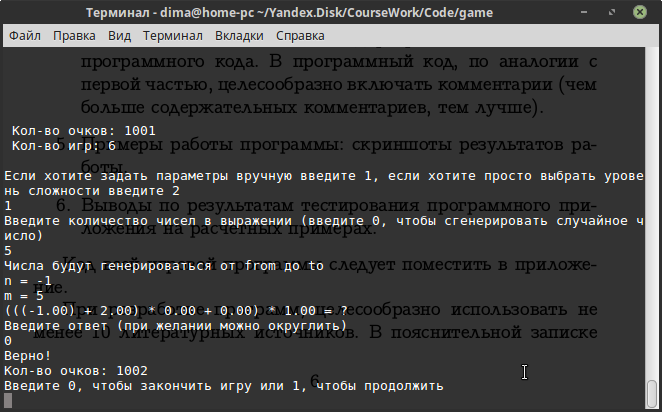
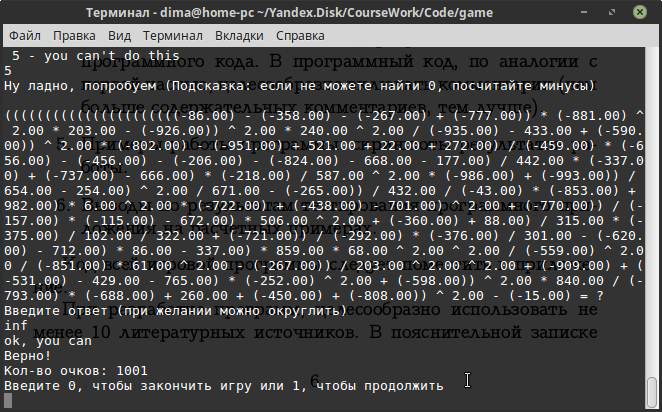
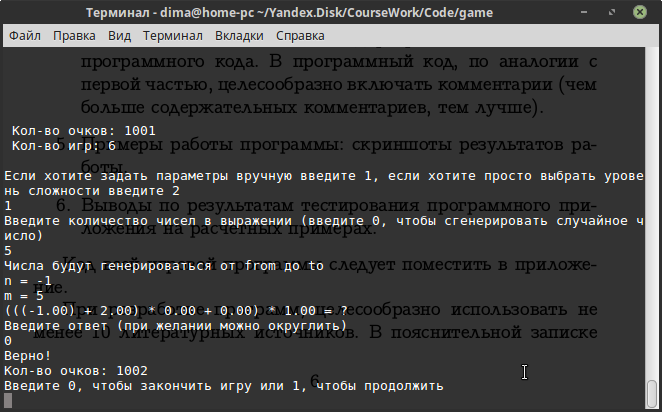


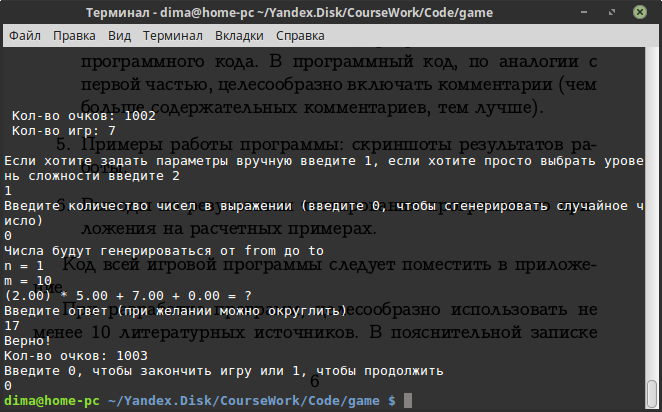
Рисунок 2.1.3.

Рисунок 2.1.4.

Рисунок 2.1.5.

Рисунок 2.1.6.

Рисунок 2.1.7.

Рисунок 2.1.8.

**Выводы:**

Многие выражения довольно сложно точно посчитать в уме, однако возможность округления даёт возможность не использовать калькулятор или листок при игре.

**Заключение**

По итогам исследования выходных данные программы 2 задания 1 части можно сделать вывод, что метод деления пополам гораздо менее эффективен, чем метод касательных.

В программе 1 надо задавать число по модулю меньшее 1, в программе 2 есть достаточно много ограничений, они расписаны выше, в игровой программе ограничения на входные данные — целые числа.

**Список использованных источников**

<https://ru.wikipedia.org/wiki/Метод_Ньютона>

http://cppstudio.com/post/850/

<http://all-ht.ru/inf/prog/c/func/log,logf,logl.html>

<https://www.wolframalpha.com/>

<https://libreoffice.su/draw/osnovyi-rabotyi-libreoffice-draw-sozdanie-blok-shem.html>

<https://studfiles.net/preview/1755574/page:2/>

[https://github.com/junaart/High-Programming/blob/master/Course%20work/book.pdf](https://github.com/junaart/High-Programming/blob/master/Course work/book.pdf)

**Приложения**

Ссылки на программы:

https://github.com/rdm1234/HighProgramming/blob/master/c/Course%20Work/CW\_1\_1.c

https://github.com/rdm1234/HighProgramming/blob/master/c/Course%20Work/CW\_1\_2.3.c

[https://github.com/rdm1234/HighProgramming/blob/master/c/Course%20Work/CW\_game.c](https://github.com/rdm1234/HighProgramming/blob/master/c/Course Work/CW_game.c)

Игровая программа:

// Подключение библиотек

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <math.h>

// Прототипы функций

// generate() отвечает за генерацию и вывод самого выражения, а также возвращает ответ

float generate(int, int, int);

// generate\_level() генерирует входные данные для функции generate() в зависимости от выбранного уровня сложности

int generate\_level(int\*, int\*, int\*);

void generate\_params();

// Значение функции используется в формуле генерации чисел выражения

int range(int, int);

// show\_element() используется для вывод каждого элемента в функции generate() (если число отрицательное, ставит скобки)

void show\_element(float);

// Функция main - "главная функция программ", она выполняется при запуске программы

int main(){

// Генерация случайных чисел теперь зависит от системного времени

srand(time(NULL));

// Объявление переменных

/\* ifParams - переменная, задаваемая польлзователем в начале каждой игры:

если равна 1, то пользователь выбирает уровень сложности;

если равна 2, то пользователь задаёт параметры генерации вручную.

ifEnd - переменная, задаваемая пользователем в конце каждой игры, если равна 0, то игра заканчивается.

games - количество игр.

points - количество очков по итогам всех игр.

level - уровень сложности, который выбрал пользователь

\*/

int ifParams, ifEnd=0, games=0, points=0, level;

/\* from, to - от скольки до скольки будут генерироваться числа

elementsAmount - количество элементов в генерируемом выражении

\*/

int from, to, elementsAmount;

/\* result - ответ выражения

answer - вводимый пользователем ответ

\*/

float result, answer;

// Бесконечный цикл, выход из которого будет осуществляться через оператор break

// Использую его для возможности играть сколько угодно раз подряд, не запуская программу заново, при этом суммируются очки

while(1){

// Сначала выводится количество очков и количество сыгранных игр

printf(" Кол-во очков: %d\n Кол-во игр: %d\n\n", points, games);

// Пользователь выбирает способ генерации выражения

printf("Если хотите задать параметры вручную введите 1, если хотите просто выбрать уровень сложности введите 2\n");

scanf("%d", &ifParams);

// Если он ввёл 2, то он выбирает уровень сложности, иначе он задаёт параметры вручную

if(ifParams==2){

// в функцию передаются ссылки на элементы, поскольку требуется, чтобы она меняла значения элементов, а не их копий

level = generate\_level(&from, &to, &elementsAmount);

}

else {

// Пользователь вводит количество чисел в выражении

printf("Введите количество чисел в выражении (введите 0, чтобы сгенерировать случайное число)\n");

scanf("%d", &elementsAmount);

// Если пользователь ввёл 0, генерируется число от 1 до 10 включительно

if(elementsAmount == 0)

elementsAmount = rand()%10+1;

// Если пользователь воодит количество элементов < 0, то берётся модуль от этого числа

if(elementsAmount < 0)

elementsAmount = abs(elementsAmount);

// Пользователь вводит какие числа будут генерироваться (с проверкой неверно введённых данных - если числа равны или если первое больше второго)

do{

printf("Числа будут генерироваться от from до to\nn = ");

scanf("%d", &from);

printf("m = ");

scanf("%d", &to);

} while(from == to || from > to);

}

/\* Переменной result присваивается значение, возвращаемое функцией generate()

Пользователь может ввести либо точный ответ, либо округлённый по обычным математическим правилам - и тот, и тот зачтётся

Если пользователь ответил верно - ему зачисляться очки в зависимости от сложности выражения (уровень сложности + 1, а если самый сложный +1000)

Если пользователь отвечает неверно, то кол-во очков уменьшается и выводится ответ (Количество очков не может опуститься ниже 0)

\*/

result = generate(elementsAmount, from, to);

printf("Введите ответ (при желании можно округлить)\n");

scanf("%f", &answer);

if(answer==roundf(result) || answer == result){

points+=level+1;

if(level==5){

printf("ok, you can\n");

points+=994;

}

printf("Верно!\n");

level=0;

}

else{

points--;

printf("Неверно. Ответ: %.3f\n", result);

}

if(points<0)

points=0;

// Пользователь выбирает продолжать ли игру

printf("Кол-во очков: %d\nВведите 0, чтобы закончить игру или 1, чтобы продолжить\n", points);

scanf("%d", &ifEnd);

// Если пользователь вводит 0, цикл прерывается с помощью оператора break

if(ifEnd == 0)

break;

games++;

printf("\033[2J");// Очищает экран (как system('clean'))

}

return 0;

}

/\* Выбор уровня сложности - от 0 до 5.

В зависимости от выбранного уровня будут заданы from, to и elementsAmount.

Функция вернёт выбранный уровень сложности для подсчёта получаемых игроком очков.

\*/

int generate\_level(int \*fP, int \*tP, int \*elAmP){

int level;

printf("Выберите уровень сложности\n 0 - very easy\n 1 - easy\n 2 - medium\n 3 - hard\n 4 - impossible\n 5 - you can't do this\n");

scanf("%d", &level);

switch(level){

default:

level = 0;

\*fP=rand()%100-50;

\*tP=rand()%100+51;

\*elAmP=rand()%6+2;

break;

case 0:

\*fP=0;

\*tP=10;

\*elAmP=2;

break;

case 1:

\*fP=0;

\*tP=100;

\*elAmP=3;

break;

case 2:

\*fP=-100;

\*tP=100;

\*elAmP=3;

break;

case 3:

\*fP=-500;

\*tP=500;

\*elAmP=5;

break;

case 4:

\*fP=-1000;

\*tP=1000;

\*elAmP=10;

break;

case 5:

\*fP=-1000;

\*tP=1000;

\*elAmP=100;

printf("Ну ладно, попробуем (Подсказка: если не можете найти 0, посчитайте минусы)\n\n");

}

return level;

}

// Генерация выражения c заданными параметрами: количество чисел в выражении; отрезок на числовой оси, в пределах которого будут генерируемые числа.

float generate(int elementsAmount, int from, int to){

float result=0, element[2];

int i, \*signArr, r;

// Переменной R присваивается число, получаемое в функции range()

r=range(from, to);

// под signArr выделяется пямять = количеству элементов в массиве, умноженному на размер типа int

signArr = (int \*)malloc((elementsAmount)\*sizeof(int));

// Это будет использоваться для последней итерации цикла

signArr[elementsAmount-1]=-1;

/\* Генерация чисел от 1 до 5

1 - сложить

2 - вычесть

3 - умножить

4 - разделить

5 - возвести в степень (мб извлечь корень)

\*/

for(i = 0; i < elementsAmount-1; i++)

signArr[i] = rand()%5+1;

// Вывод скобок в начале выражения, чтобы действия шли в правильном математическом порядке

for(i = 1; i < elementsAmount; i++){

if(signArr[i] > 2 && signArr[i-1] < 3)

printf("(");

}

// Вывод всех элементов выражения

for(i = -1; i < elementsAmount; i++){

// Генерируется элемент от from до to, для этого используется формула:

element[0]=rand()%r - abs(from);

// Вывод первого элемента выражения

if(i==-1){

result=element[0];

if(signArr[0]>2)

printf("(");

show\_element(result);

continue;

}

// Если следующее действие деление, то это помогает избежать случаев деления на 0

if(signArr[i] == 4 && element[0] == 0)

element[0]++;

// Закрывающая скобка для правильного математического порядка действий

if(signArr[i] > 2 && signArr[i-1] < 3)

printf(")");

switch(signArr[i]){

case 1:

printf(" + ");

result+=element[0];

break;

case 2:

printf(" - ");

result-=element[0];

break;

case 3:

printf(" \* ");

result\*=element[0];

break;

case 4:

printf(" / ");

result/=element[0];

break;

case 5:

// Ввиду того, что довольно сложно считать в уме большие степени, а также того, что в результате получатся огромные числа,

// я сделал только степени 0, 0.5, 2

switch(rand()%3){

case 0:

element[0]=0;

case 1:

element[0]=0.5;

case 2:

element[0]=2;

}

printf(" ^ ");

result=powf(result, element[0]);

break;

case -1:

printf(" = ? \n");

}

// Обрабатывается последняя итерация

if(signArr[i]==-1)

break;

show\_element(element[0]);

// Это используется для расставления скобок

element[1]=element[0];

}

// освобождается память и возвращается результат

free(signArr);

return result;

}

void show\_element(float element){

if(element<0)

printf("(");

printf("%.2f", element);

if(element<0)

printf(")");

}

int range(int n, int m){

if(n<0 && m<0)

return abs(n)-abs(m);

else

return m-n;

}