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 №5**

Выполнила:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

студентка группы ИВТ-11 Жукова А.В.

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

доцент, к.т.н. Артамонов Ю.Н.

Котельники 2020

**СОДЕРЖАНИЕ**

|  |  |  |
| --- | --- | --- |
| **1)** | **ВВЕДЕНИЕ**…………………………………………………… | 3 |
| **2)** | **ЧАСТЬ №1. РАЗРАБОТКА ЧИСЛЕННЫХ АЛГОРИТМОВ** | 4 |
| 1.1 | **Суммирование рядов и вычисление элементарных функций** | 4 |
| 1.1.1 | Сумма бесконечного ряда…………………………………… | 4 |
| 1.2 | **Приближенные методы нахождения корней уравнения**… | 15 |
| 1.2.1 | Метод касательных…………………………………………… | 15 |
| 1.2.2 | Метод секущих…………………………………………………… | 16 |
| **3)** | **ЧАСТЬ №2.РАЗРАБОТКА ИГРОВОЙ ПРОГРАММЫ**… | 47 |
| 2.1 | **Случайная генерация выражения** | 61 |
| 2.2 | **Вычисление и вывод выражения** | 62 |
| 2.3 | **Проверка ответа пользователя и игровая часть** | 62 |
| **4)** | **ЗАКЛЮЧЕНИЕ**………………………………………………… | 69 |
| **5)** | **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**……… | 69 |

**ВВЕДЕНИЕ**

Информационные технологии влились в жизни обычных людей не так давно,

однако уже стали неотъемлемой их частью. Скорее всего, найдется мало людей, которые не используют компьютер вовсе, ведь сейчас он нужен почти всегда [10]. Многие используют его для работы, чтобы составлять отчеты, проводить конференции, составлять базы данных. Ученые используют вычислительные мощности компьютера для сложных расчетов, которые всегда присутствуют при изучении какого-либо явления. Для каждой задачи всегда найдется программа, которая решит, или хотя бы поможет решить ее.

Однако за каждой программой, за каждой утилитой или сайтом стоит программист, реализовавший из идеи готовый продукт, который в дальнейшем будет использоваться людьми. Сама программа – это последовательность действий, задаваемых компьютеру, но без сил и стараний специалиста эта последовательность не появится.

Чтобы создавать программы, которые впоследствии будут приносить пользу людям, нужно иметь не только знание принципов работы компьютера, но и знания из области точных наук, а именно: высшей математики, физики, логики и т.д. В этот список входит еще много областей, и каждая, так или иначе, пересекается с программированием.

Создание программ для работы на компьютере начинается с написания их кода на каком-либо языке программирования. Чаще всего это язык программирования высокого уровня. Что же это такое? Это формальный язык, придуманный людьми, имеющий свои правила и особенности. Самая главная его задача – организовать набор правил для структурирования процессов компьютера. Языками высокого уровня называются языки, основу которых составляют абстрактные конструкции, объединившие в себе операции, написанные на машинном коде. Машинный код [7] тяжело обрабатывать человеку, поэтому создание языков программирования высокого уровня сильно облегчило работу программистам.

Разные программисты используют разные методы [5] написания кода и языки, однако в данной курсовой работе мы разберем конкретно программирование на языке высокого уровня Си.

**ЧАСТЬ №1.РАЗРАБОТКА ЧИСЛЕННЫХ АЛГАРИТМОВ**

* 1. **Суммирование рядов и вычисление элементарных функций**

1.1.1 Сумма бесконечного ряда

Одна из важных задач программиста – умение облегчить процесс математических вычислений с помощью грамотно написанного кода. Ведь это может понадобиться не только пользователям, но и самому программисту, так как для решения некоторых задач требуются математические вычисления. Каждый раз вычислять одно и то же самостоятельно – бессмысленное занятие, учитывая огромный потенциал вычислительной мощности компьютера, даже самого слабого.

В данной теме мы рассмотрим, как можно рационализировать вычисление суммы бесконечных рядов. Но для начала стоит понять, что это такое.

По сути своей сумма бесконечного ряда [8] – это поочередно просуммированные члены ряда, которые последовательно изменяются по какой-либо закономерности. Члены ряда – это числа, а закономерность, по которой они изменяются, обычно определяется формулой N-ого члена последовательности или, как его еще называют, общего члена последовательности. Чтобы узнать значение любого члена последовательности, достаточно подставить в формулу его порядковый номер (нумерация зачастую начинается с нуля, однако может начинаться и с единицы). Чем больше членов ряда просуммировано, тем «ближе» сумма ряда к конечному числу, к которому она сходится. То есть хоть ряд и бесконечный, он «стремится» к какому-то конкретному числу, которое может быть как целое, так и иррациональное.

Бывает и такое, что ряд сходится к бесконечности или вообще не сходится, но так как мы рассматриваем поиск суммы бесконечного ряда до конкретного члена ряда, такого количества теории будет достаточно.

Сумму бесконечного ряда можно записать с помощью вот такой общей формулы:

где **Sn** – сумма бесконечного ряда, **a1,a2,a3** и **аn** – члены ряда.

Как уже было сказано, при суммировании членов ряда мы все ближе приближаемся к более точной сумме. Таким образом, можно найти ее до определенной точности [4]. Точность – это то, насколько близко найденное значение к истинному значению, в данном случае к числу, к которому сходится ряд.

Рассмотрим бесконечный ряд, имеющий представление такого вида:

1) Дано:

где **n** – порядковый номер члена ряда, а левая часть уравнения является представлением числа, к которому сходится ряд.

2) Требуется найти: сумму ряда до заданной точности при различных значениях **х**, вычислив при этом количество членов ряда, требуемых для достижения заданной точности

3)Решение: далее будет представлен листинг программного кода с комментариями для данной задачи.

|  |
| --- |
| #include<stdio.h>  #include<math.h>  #include<stdlib.h>  #include<time.h>  double func(double, double, int \*);  //функция нахождения суммы бесконечного ряда до заданной точности, принимает //значение x, переменную toch отвечающую за точность  //и указатель на переменную i, отвечающую за кол-во итераций  double func(double x, double toch, int \*i)  {  int n=0, t=1;//вводятся переменные n-для подсчета количества членов ряда и t для //изменения членов ряда в соответствии с формулой  double schet=0;//вводится переменная для подсчета суммы бесконечного ряда  while (fabs(pow(x, t)/(float)t)>toch)//пока последний добавленный член дроби //больше точности по модулю...  {  schet+=(pow(x, t)/(float)t);//в переменную для подсчета суммы по формуле //записывается новый член дроби  n++;//количество членов ряда увеличивается на единицу  t=(4\*n)+1;//в переменную t по формуле записывается степень (она же //является и дробной частью) следующего члена ряда  (\*i)=n;//разименовываем указатель i и присваиваем ему количество членов //ряда,чтобы i изменилась и в main  printf("Cумма=%.10f точность=%f кол-во членов ряда:=%d\n", schet,toch,n);//вывод на экран для наглядности значений найденной с каждой //итерацией суммы ряда, точности и количества членов ряда  }  return schet;//после прохождения цикла функция возвращает полученную сумму //ряда до заданной точности  }  int main()  {  double x=1, toch, otv,ist;//вводятся переменные x - отвечающая за введенный //пользователем x соответственно, toch-отвечающая за точность, otv отвечающая за //итоговый ответ и ist для подсчета истинного значения  int vib, chis=-1, i=0;//так же вводится переменная vib, для выбора дальнейших //действий, chis,отвечающая за количество знаков после запятой для вычисления //точности и переменная i, отвечающая за количество членов ряда  while(vib!=2)//пока переменная выбора не равна двум...  {  while ((x>=1)||(x<=-1))//пока x не попадает в допустимый диапазон значений...  {  printf("Введите значение x:\n");//на экран выводится предложение ввести х  scanf("%lf", &x);//с клавиатуры считывается х  if ((x>=1)||(x<=-1)) printf("Введена недопустимая точность, попробуйте еще раз\n");//если х не попадает в //допустимый диапазон значений, выводится сообщение //попробовать еще раз  else break;//иначе если х попадает в допустимый диапазон, цикл прерывается  }  while (chis<=-1)//пока количество знаков после запятой будет меньше или равно 1  {  printf("Введите точность(до скольки знаков после запятой):\n");//выводится //предложение ввести точность, до определенного знака после запятой, не включая его  scanf("%d", &chis);//считывается количество знаков после запятой  if (chis<=-1) printf("Введено неверное значение попробуйте еще раз\n");//если //количество знаков после запятой не вошло в допустимый диапазон, выводится //предложение попробовать еще раз  else toch=1.0/(pow(10.0, chis));//иначе переменной toch(отвечающей за //точность) присваивается приемлемый вид для вычислений в соответствии с //количеством заданных знаков после запятой  }  ist=(1.0/4.0\*log((1.0+x)/(1.0-x)))+(1.0/2.0\*atan(x));//вычисление истинного //значения  otv=func(x, toch,&i);//переменной ответ присваивается значение, возвращаемое //функцией func, в соответствии с введенными значениями точности и x  printf("При точности до %d знаков после запятой, изначальном х:%f, сумма:%.10f, кол-во членов ряда:%d\n", chis,x,otv,i);//далее на экран выводится //точность, изначальный х, найденная сумма и количество членов ряда для ее //достижения  printf("Истинное значение:%.10f\n",ist);//вывод на экран истинного значения для //сравнения  printf("Хотите продолжить? 1-да 2-нет\n");//на экран выводится предложение //закончить вычисления или попробовать еще раз  scanf("%d",&vib);//считывается выбор пользователя с клавиатуры  if(vib!=1&&vib!=2)//если пользователь случайно ввел выбор, которого не //существует  {  printf("Такой операции не существует, попробуйте еще раз!\n");//на экран //выводится сообщение, что такой операции не существует  while(vib!=1&&vib!=2)//пока выбор не окажется из заданных...  {  printf("Хотите продолжить? 1-да 2-нет\n");//на экран выводится //предложение закончить вычисления или попробовать еще раз  scanf("%d",&vib);//считывается выбор пользователя с клавиатуры  }  }  else if(vib==2) {printf("Конец программы.\n"); vib=2;}//если же выбор //пользователя оказался вторым, то выводится сообщение о завершении программы, //а переменной vib присваивается 2, чтобы выйти из цикла выбора  x=1;//переменной х присваивается ее "базовое значение", чтобы при выборе //продолжать вычисления программа работала верно  chis=-1;//то же самое повторяется и с переменной chis  }  return 0;//функция main фозвращает 0, оповещающий о конце работы программы  } |

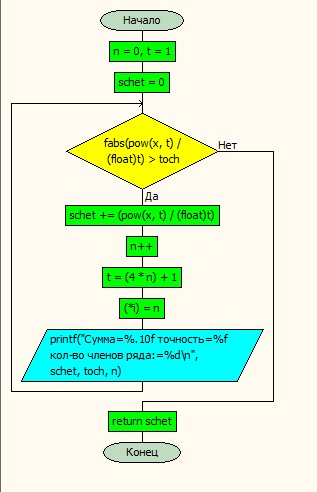
**Листинг 1.1**

Данный код описывает алгоритм поиска суммы ряда до заданного знака после запятой, не включая его, а так же рассчитывает количество нужных для этого членов ряда. В коде предусмотрена возможность вводить значение х столько раз, сколько пожелает пользователь, а так же возможность завершить программу в любой момент. За поиск суммы и количества членов отвечает функция **func**, в функции **main** реализовано несколько ограничений ввода переменных:

1. Ограничение на ввод **х**. Он может быть введен только в диапазоне **-1<x<1**. Если пользователь введет **х**, не соответствующий диапазону, на экран выведется уведомление об этом и будет предложено заново ввести **х**, пока ввод не будет корректен.
2. Ограничение на ввод, до какой точности будут производиться вычисления. Если пользователь введет отрицательное количество знаков после запятой, на экран выведется уведомление о некорректном вводе и будет предложено заново ввести количество знаков, пока ввод не будет корректен.
3. Так как в коде реализована возможность вводить значения **х** и вычислять сумму ряда сколько угодно, существует ограничение на ввод выбора дальнейших действий. Если пользователь ввел некорректную операцию, на экран выведется уведомление об этом и будет предложено выбрать операцию продолжения/завершения программы, пока ввод не будет корректен.

Более подробно с алгоритмом можно ознакомиться, прочитав комментарии, добавленные к коду (см. **Листинг 1.1.1**). Далее будут представлены блок-схемы данного программного кода.

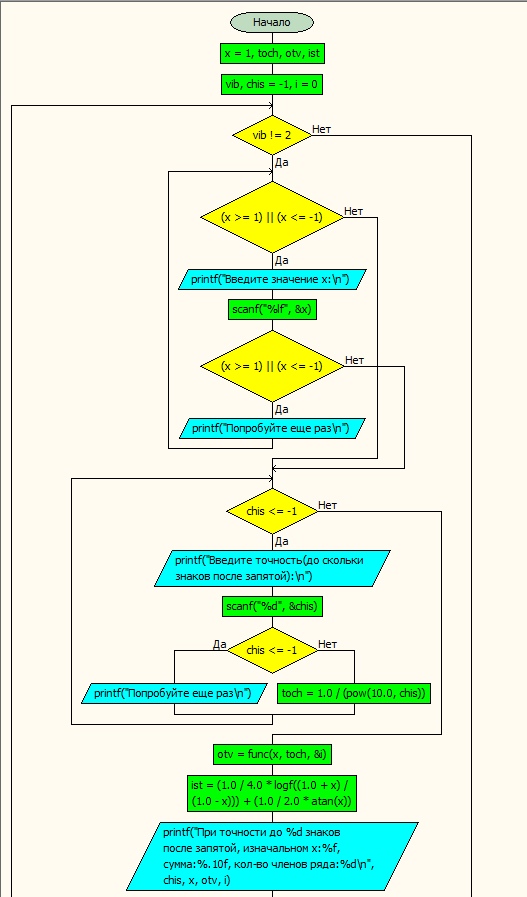
Блок-схема функции **func**:

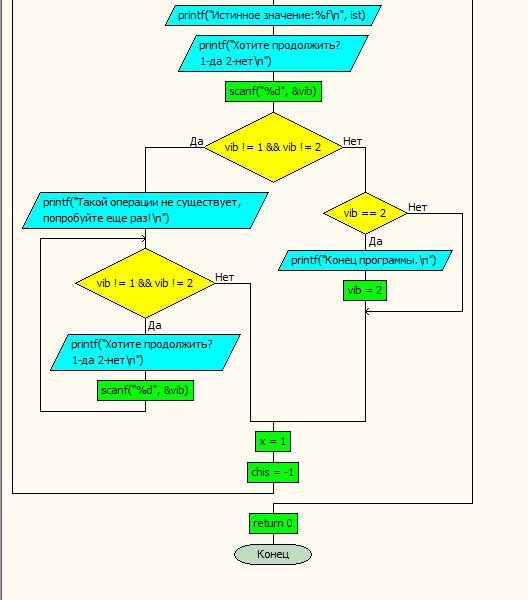


**Блок-схема 1.1.1**

|  |  |  |  |
| --- | --- | --- | --- |
| **Переменная** | **Тип** | **Область изменения** | **Начальное значение** |
| **i** | **int** | **func, main** | **0** |
| **toch** | **double** | **func** | **Присваивается в main** |
| **n** | **int** | **func** | **0** |
| **t** | **int** | **func** | **1** |
| **schet** | **double** | **func** | **0** |

Блок-схема функции **main:**

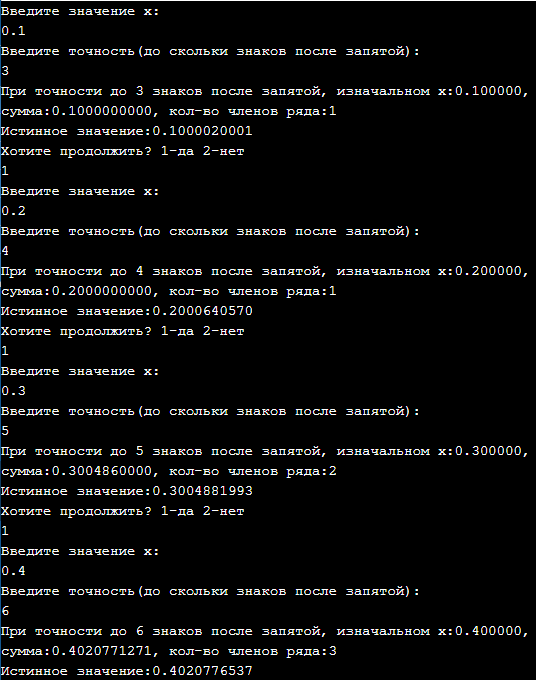
****

****

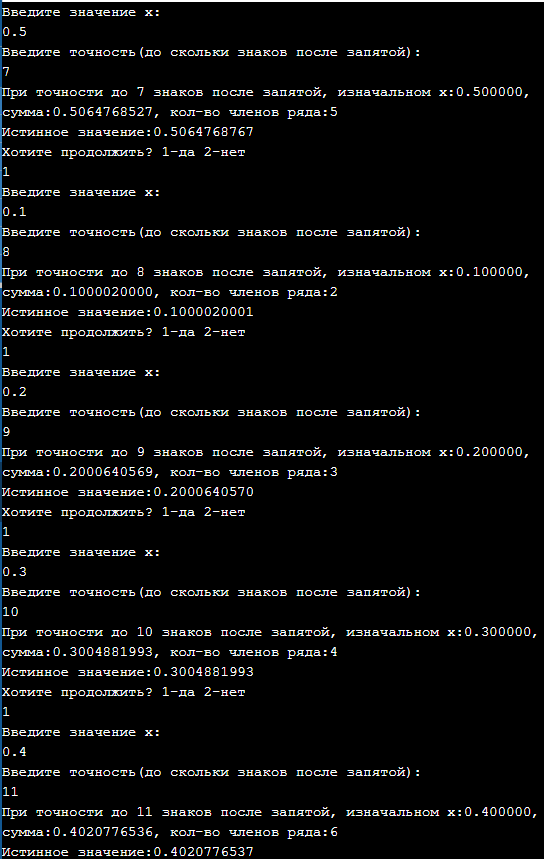
**Блок-схема 1.1.2**

|  |  |  |  |
| --- | --- | --- | --- |
| **Переменная** | **Тип** | **Область изменения** | **Начальное значение** |
| **x** | **double** | **main** | **1** |
| **toch** | **double** | **main** | **Задается далее** |
| **otv** | **double** | **main** | **Задается далее** |
| **ist** | **double** | **main** | **Задается далее** |
| **vib** | **int** | **main** | **Вводится с клавиатуры** |
| **chis** | **int** | **main** | **-1** |
| **i** | **int** | **func, main** | **0** |

Проверка работоспособности программы при различных значениях **х** и точности:



**Скриншот 1.1.1**

****

**Скриншот 1.1.2**

|  |  |  |  |
| --- | --- | --- | --- |
| **Точность (до N знаков после запятой)** | **Значение**  **х** | **Сумма ряда** | **Кол-во членов ряда** |
| **3** | **0.1** | **0,1000000000** | **1** |
| **4** | **0.2** | **0,2000000000** | **1** |
| **5** | **0.3** | **0,3004860000** | **2** |
| **6** | **0.4** | **0,4020771271** | **3** |
| **7** | **0.5** | **0,5064768527** | **5** |
| **8** | **0.1** | **0,1000020000** | **2** |
| **9** | **0.2** | **0,2000640569** | **3** |
| **10** | **0.3** | **0,3004881993** | **4** |
| **11** | **0.4** | **0,4020776536** | **6** |

Вывод: как можно заметить, программа успешно выполняет поиск суммы бесконечного ряда до заданной точности. Чем больше членов ряда будет просуммировано, тем более точное значение мы получим. Однако всегда существует вероятность погрешности, связанной с особенностью вычисления компьютером не целых чисел, что всегда стоит учитывать.

**1.2 Приближенные методы нахождения корней уравнения**

1.2.1 Метод касательных

Так же с помощью хорошо написанного кода можно ускорить поиск корней уравнений вида f(x)=0. Существует много алгоритмов, позволяющих это делать, но для начала мы рассмотрим алгоритм под названием «Метод касательных»[1]. Для большей наглядности см. Рисунок 1.2.1

1) Случайно выбираем первую точку xn на оси абсцисс.

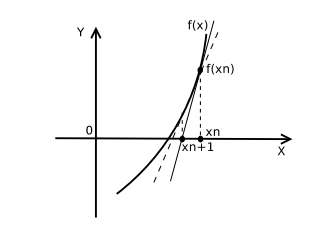
2) После, мы находим значение функции в этой точке, которое мы обозначим как f(xn).

3) Мысленно проводим касательную к функции f(x) через точку f(xn).

4) Точку, где касательная пересечет ось абсцисс, мы назовем xn+1.

5) Точка xn+1 становится следующей, от которой мы будем отталкиваться

Далее пункты 2-5 повторяются до тех пор, пока разность между xn и xn+1 по модулю не станет меньше заданной точности, до которой нам нужен корень.



**Рисунок 1.2.1**

Метод кажется достаточно простым, однако есть одно. Как же искать точку xn+1 без построения графика? В данном вопросе нам поможет простая математика. Известно, что производная в точке, где касательная пересекает ось абсцисс, равна тангенсу угла наклона касательной к функции. В таком случае достаточно просто вывести формулу:

Так же, для корректного вычисления, нужно проверять знаки на концах отрезка. Для этого достаточно отслеживать, разные ли они и уменьшать точность, если это оказалось так.

1.2.2 Метод секущих

«Метод секущих»[2] достаточно похож на «метод касательных», однако в его алгоритме не используется производная. Его алгоритм так же прост для понимания, но все же в нем есть свои нюансы. Для большей наглядности см. Рисунок 1.2.2

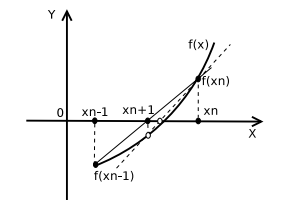
1) Случайно выбираем на оси абсцисс две точки, обозначим их как xn-1 и xn

2)Находим значения функции в этих точках, обозначаем их как f(xn-1) и f(xn)

3) Мысленно проводим через f(xn-1) и f(xn) прямую и находим точку, где эта прямая пересечется с осью абсцисс и обозначаем ее как xn+1

4) Теперь вместо отрезка [xn-1;xn] мы имеем новый отрезок с большим приближением к корню [xn+1;xn]

Далее мы должны повторять пункты 2-4 до тех пор, пока разница между правым и левым краем отрезка по модулю не станет больше точности, до которой нам нужен корень.



**Рисунок 1.2.2**

Для того, чтобы найти xn+1 нам потребуется формула вида:

Однако у данного способа есть свои недостатки. Несмотря на то, что каждое последующее приближение хоть и будет все ближе к корню, оно будет приближаться бесконечно, так как уменьшается с каждым новым найденным xn+1. Это стоит учитывать при решении уравнений с помощью данного способа.

Рассмотрим реализацию решения уравнений с помощью «метода касательных» и «метода секущих»:

Дано: несколько уравнений вида f(x)=0

1) sin(c\*x)-d=0

2)e^(c\*x)-d=0

3)x^5+c\*x^2-d=0

4)x^c-x^(c+1)-d=0

5)(x^2-d)-(x^c-6)=0

6) (c\*xn^3-d)/(x+d)=0

c, d и х вводятся с клавиатуры, так же с клавиатуры задается точность

Найти: реализовать нахождение корня уравнения с помощью метода касательных и метода секущих при различных значениях с, d и x.

Решение: далее будет представлен листинг программного кода с комментариями для данной задачи

|  |
| --- |
| #include<stdio.h>  #include<math.h>  #include<stdlib.h>  #include<time.h>  double PROIZVOD(double,int, double,double);  //функция для нахождения производной, принимает значения переменных х, с и //переменной vib, отвечающий за выбранное пользователем уравнение  double PROIZVOD(double xn, int vib, double c, double d)  {  if(vib==1) return cos(c\*xn);//если переменная vib равна 1, то функция //возвращает производную уравнения 1 от переменной xn  else if(vib==2) return exp(c\*xn);//если переменная vib равна 2, то функция //возвращает производную уравнения 2 от переменной xn  else if(vib==3) return (5\*pow(xn,4))+(c\*2\*xn);//если переменная vib равна 3, то //функция возвращает производную уравнения 3 от переменной xn  else if(vib==4) return c\*pow(xn, c-1)-(c+1)\*pow(xn,c);//если переменная vib //равна 4, то функция возвращает производную уравнения 4 от переменной xn  else if(vib==5) return 2\*xn-c\*pow(xn,c-1);//если переменная vib равна 5, то //функция возвращает производную уравнения 5 от переменной xn  else if(vib==6) return ((c\*3\*pow(xn,3))\*(xn+d)-(c\*pow(xn,3)-d))/pow(xn+d,2);//если переменная vib равна 6, то функция возвращает производную //уравнения 6 от переменной xn  }  double FUNC(double,int,double,double);  //функция для нахождения функции, принимает значения переменных х, с, d и //переменной vib,отвечающий за выбранное пользователем уравнение  double FUNC(double xn, int vib, double c, double d)  {  if(vib==1) return sin(c\*xn)-d;//если переменная vib равна 1, то функция //возвращает значение функции от уравнения 1 от переменной xn  else if(vib==2) return exp(c\*xn)-d;//если переменная vib равна 2, то функция //возвращает значение функции от уравнения 2 от переменной xn  else if(vib==3) return pow(xn,5)+(c\*xn\*xn)-d;//если переменная vib равна 3, то //функция возвращает значение функции от уравнения 3 от переменной xn  else if(vib==4) return pow(xn, c)-pow(xn,c+1)+d;//если переменная vib равна 4, //то функция возвращает значение функции от уравнения 4 от переменной xn  else if(vib==5) return (pow(xn,2)-d)-(pow(xn,c)-6);//если переменная vib равна 5, //то функция возвращает значение функции от уравнения 5 от переменной xn  else if(vib==6) return (c\*pow(xn,3)-d)/(xn+d);//если переменная vib равна 6, то //функция возвращает значение функции от уравнения 6 от переменной  }  double KASAT(double, double , double , double, double \*, int);  //функция для вычисления приближенного значения с помощью метода касательных, //принимает начальный х, переменные c, d, toch(отвечающую за точность)  //а так же указатель на переменную step(это понадобиться, чтобы в мейне вывести //точность как количество знаков после запятой)  //и переменную vib, отвечающую за номер выбранного пользователем уравнения  double KASAT(double xn1, double c, double d, double toch, double \*step, int vib)  {  int iter=0;//переменная iter отвечает за количество итераций цикла и изначально //равна нулю  double xn2=xn1, func, proiz;//переменная xn2 отвечает за следующий x, найденный в //ходе программы(изначально равен x введенному с клавиатуры), а переменные func //и proiz соответственно для вычисления функции и производной  do//начало цикла  {  xn1=xn2;//"предыдущему" x присваиваем "следующий" х  func=FUNC(xn1,vib,c,d);//находим функцию от х с помощи функции //FUNC  proiz=PROIZVOD(xn1,vib,c,d);//находим производную от х с помощью функции //PROIZVOD  xn2=xn1-(func/proiz);//"следующему" х присваиваем разность "предыдущего" х //и частного от функции и производной(от "предыдущего" х)  if(((FUNC(xn2,vib,c,d)-toch)>0&&(FUNC(xn2,vib,c,d)+toch)>0)||((FUNC(xn2,vib,c,d)-toch)<0&&(FUNC(xn2,vib,c,d)+toch<0))) //проверка знаков, если разность и сумма //функции от "следующего" х и точности имеет один знак, то...  {  (\*step)=((\*step)+1);//через разименование указателя прибавляем //переменной step единицу, чтобы она изменилась и вне функции(step отвечает за то, //сколько знаков после запятой будет точность)  toch=pow(10.0,(-(\*step)));//увеличиваем toch(отвечающую за //точность) на еще один знак после запятой  }  iter++;//инкрементируем переменную, отвечающую за кол-во итераций  printf("xn=%.10f\txn+1=%.10f\tfabs(xn-xn+1)=%.10f\ti=%d\n",xn1,xn2,fabs(xn1-xn2),iter);//для нагладности изменений //выводим на экран "предыдущий" х, "следующий", их разность по модулю и //количество пройденных итераций  }while(iter<=200&&(fabs(xn1-xn2)>toch));//цикл продолжается, пока количество //итераций меньше или равно 200 и разность между "предыдущим" и "следующим" х //по модулю больше точности  if(iter>=200) xn1=99999999;//если количество итераций превысило или равно 200, //то х присваивается значение, показывающее "ошибку"  return xn1;//возвращаем х  }  double SEKUSH(double, double, double , double , double, double \*, int);  //функция для вычисления приближенного значения с помощью метода секущих, //принимает начальный х, переменные c, d, toch(отвечающую за точность)  //а так же указатель на переменную step(это понадобиться, чтобы в мейне вывести //точность как количество знаков после запятой)  //и переменную vib, отвечающую за номер выбранного пользователем уравнения  double SEKUSH(double xn1, double xn2, double c, double d, double toch, double \*step, int vib)  {  int iter=0;//переменная iter отвечает за количество итераций цикла и изначально //равна нулю  double func1, func2=FUNC(xn2,vib,c,d), xs, xn=xn1, xk=xn2;//переменные func1 и //func2 отвечают за значение функции на краях отрезка(так правый край отрезка //никогда не меняется, его вычисляем при объявлении)  //переменная xs отвечает за следующее найденное приближение левого края отрезка, //а переменные xn и xk отвечают соответственно за левый и правый край отрезка  while(iter<=400&&(fabs(xn2-xn1)>toch))//пока количество итераций не превысит 400 //и разность правого и левого края отрезка по модулю больше точности, цикл //продолжается  {  func1=FUNC(xn1,vib,c,d);//переменной func1 присваиваем значение //функции от левого края отрезка при помощи функции FUNC  xs=xn2-(func2\*((xn2-xn1)/(func2-func1)));//ищем следующее приближение по //формуле и присваиваем его xs  iter++;//инкрементируем переменную, отвечающую за кол-во итераций  printf("x1=%.10f\tx2=%.10f\tfabs(x2-x1)=%.10f\ti=%d\n",xn1,xn2,fabs(xn2-xn1),iter);//для большей наглядности выводим //на экран значение правого и левого края отрезка, их разность и количество //пройденных итераций  xn1=xs;//левому краю присваиваем новое приближение  }  return xn1;//после выхода из цикла возвращяем левый край отрезка, так как он //ближе всего к истинному корню. однако изза специфики метода, цикл поиска коря //будет бесконечным поэтому выводится последнее найденное значение по //прохождению 400 итераций цикла  }  int main()  {  double xn, xk, toch, c, d, step=-1, otv;//объявляем переменные xn и xk для ввода х с //клавиатуры, переменную, отвечающую за точность toch, переменные c и d //вводимые с клавиатуры  //переменную step, отвечающую за количество знаков после запятой и переменную //otv отвечающую за найденное значение  int vib, vib2;//объявляем две переменные выбора, vib отвечает за выбор уравнения, а //переменная vib2 за выбор способа нахождения приближенного значения  while(vib!=0)//пока переменная выбора уравнений не равна нулю, цикл повторяется  {  printf("Выберете уравнение для подсчета:\n");//на экран выводится //предложение выбрать уравнение  printf("sin(cx)-d=0 - 1\ne^(cx)-d=0 - 2\nx^5+cx^2-d=0 - 3\nx^c-x^(c+1)+d=0 - 4\n(x^2-d)-(x^c-6)=0 - 5\n(c\*xn^3-d)/(x+d)=0 - 6\nЕсли хотите завершить программу - 0\n");//на экран выводятся уравнения, которые можно решить, и вариант завершить //программу. у каждого выбора свой номер  scanf("%d",&vib);//с клавиатуры вводится выбор пользователя  if(vib==0)//если пользователь выбрал завершение программы...  {  printf("Программа завершена!\n");//вывод на экран сообщения о завершении //программы  vib=0;//переменная, отвечающая за выбор уравнений обнуляется, чтобы далее //весь цикл закончился  }  else if(vib<0||vib>6) printf("Такой операции не существует, попоробуйте еще раз!\n\n");//если введена неверная операция, выводится сообщение о том, что такой //операции не существует  else//если же номер уравнения введен правильно...  {  while(vib2!=2&&vib2!=1)//цикл будет продолжаться, пока пользователь не //введет верную операцию  {  printf("Выберете метод для решения\nКасательных - 1, Секущих - 2\n");//на //экран выводится предложение выбрать метод для решения  scanf("%d",&vib2);//с клавиатуры считывается выбор метода решения  if(vib2!=2&&vib2!=1) printf("Такой операции не существует, попоробуйте еще раз!\n\n");//если введена неверная операция, выводится сообщение о том, что такой //операции не существует  }  switch(vib2)//конструкция switch принимает значение выбора метода //решения пользователя  {  //если пользователь выбрал пметод касательных, то...  case 1:  printf("Введите x:\n");scanf("%lf",&xn);//ввод х  printf("Введите c:\n");scanf("%lf",&c);//ввод с  printf("Введите d:\n");scanf("%lf",&d);//ввод d  while (step<=-1)//пока количество знаков после запятой будет //меньше или равно -1...  {  printf("Введите точность:\n");//выводится предложение ввести //точность  scanf("%lf", &step);//считывается количество знаков после запятой  if (step<=-1) printf("Попробуйте еще раз\n");//если количество знаков //после запятой не вошло в допустимый диапазон, выводится предложение //попробовать еще раз  else toch=pow(10,-step);//иначе переменной toch(отвечающей за //точность) присваивается приемлемый вид для вычислений в соответствии с //количеством заданных знаков после запятой  }  otv=KASAT(xn,c,d,toch,&step,vib);//переменной otv //присваиваем значение, которое вернет функция KASAT при введенных выше //значениях c,d и хn, step, vib  if(otv==99999999) printf("Произошла ошибка, попробуйте еще раз!\n");//если значение, которое вернет функция будет "ошибочным", то выведется //сообщение об ошибке  else printf("Корень выражения=%f найден с точностью:%f знаков после запятой\n\n",otv,step);//иначе на экран выведется найденный корень уравнения и //точность  break;//прерывание первого case  //если пользователь выбрал пметод касательных, то...  case 2:  printf("Введите начальный х:\n");scanf("%lf",&xn);//на экран //выводится предложение ввести левый край отрезка и значение считывается с //клавиатуры  xk=xn-1;//присваиваем правому краю отрезка число, меньшее чем у //левого, чтобы можно было начать проверку на вверный ввод правого края  while(xk<=xn)//пока правый край меньше или равен левому цикл //будет проверять правильность ввода правого края отрезка  {  printf("Введите конечный х:\n");scanf("%lf",&xk);//ввод //правого края отрезка  if(xk<=xn) printf("Введен недопустимый конечный х, попробуйте еще раз\n");//если правый край оказался меньше или равен левому, то //выводится сообщение об ошибке  }  printf("Введите c:\n");scanf("%lf",&c);//ввод с  printf("Введите d:\n");scanf("%lf",&d);//ввод d  while (step<=-1)//пока количество знаков после запятой будет //меньше или равно -1...  {  printf("Введите точность:\n");//выводится предложение ввести //точность  scanf("%lf", &step);//считывается количество знаков после запятой  if (step<=-1) printf("Попробуйте еще раз\n");//если количество знаков //после запятой не вошло в допустимый диапазон, выводится предложение //попробовать еще раз  else toch=pow(10,-step);//иначе переменной toch(отвечающей за //точность) присваивается приемлемый вид для вычислений в соответствии с //количеством заданных знаков после запятой  }  otv=SEKUSH(xn,xk,c,d,toch,&step,vib);//переменной otv //присваиваем значение, которое вернет функция SEKUSH при введенных выше //значениях c,d и хn, xk, step, vib  printf("Корень выражения=%f наден приближенно с точностью:%f знаков после запятой\n\n",otv,step);//вывод на экран полученного ответа  break;//завершение второго case  }  vib2=3;//после вычисления приближенного значения, нужно вернуть изначальные значения переменных, для корректной работы проверок  vib=10;//возвращаем начальное значение  step=-1;//опять возвращаем начальное значение  }  }  return 0;//завершение функции main  } |

**Листинг 1.2.1**

Данный код описывает нахождение приближенного значения корня уравнения вида f(x)=0 до заданной точности. Пользователь может задавать начальные параметры x, d, c, а так же точности. Поиск корней пользователь может продолжать ровно до того момента, пока сам не решит закончить. Для этого существует выбор выхода из программы. Реализованы две функции SEKUSH и KASAT для поиска корней с помощью метода секущих и метода касательных соответственно. Так же сделаны две функции PROIZVOD и FUNC для поиска значения производной и функции в точке x, где поиск зависит от того, какое уравнение выбрал пользователь для исследования. Т.е. если пользователь выбрал уравнение под номером 1, то поиск значения функции и производной в точке х, будет именно от 1 уравнения. В функции main реализован выбор уравнения из заданных, а так же выбор метода решения. Вследствие этого в функции main существует ряд ограничений на ввод:

1) Ограничение на ввод некорректного выбора уравнения. Пользователь может ввести только порядковые номера уравнений, для их выбора, а так же 0 для выхода из программы. При ошибочном вводе на экран выводится соответствующее сообщение, после будет предложено выбрать уравнение еще раз. Выбор уравнения будет продолжаться до того момента, пока пользователь не введет корректный выбор.

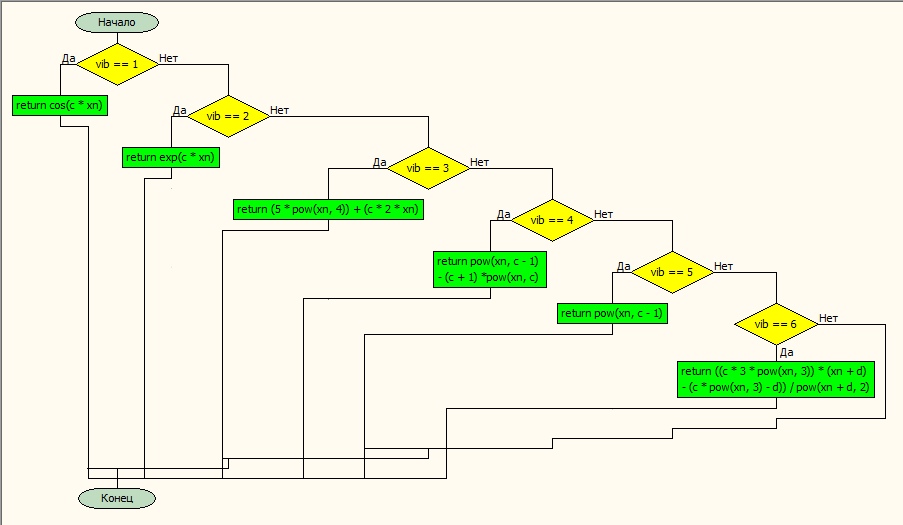
2) Ограничение на ввод некорректного выбора метода решения уравнения. Пользователь может ввести только порядковые номера методов решения уравнения, для их выбора. При ошибочном вводе на экран выводится соответствующее сообщение, после будет предложено выбрать метод еще раз. Выбор метода решения будет продолжаться до того момента, пока пользователь не введет корректный выбор.

3) Ограничение на выбор точности. Так как пользователь вводит точность, как количество знаков после запятой, есть вероятность, что он может ввести отрицательное значение. Поэтому реализовано ограничение на ввод отрицательной точности. Если все же это произошло, пользователю на экран выводится уведомление об этом и предложение ввести точность снова. Выбор точности будет продолжаться до того момента, пока пользователь не введет корректный выбор.

4) Ограничение на выбор значений правой и левой части отрезка в методе секущих. Так как левый край отрезка должен быть строго меньше правого края отрезка, если пользователь введет значение правого края отрезка некорректно, пользователю на экран выводится уведомление об этом и предложение ввести правый край отрезка снова. Выбор правого края будет продолжаться до того момента, пока пользователь не введет корректный выбор.

Более подробно с алгоритмом можно ознакомиться, прочитав комментарии, добавленные к коду (см. **Листинг 1.2.1**). Далее будут представлены блок-схемы данного программного кода.

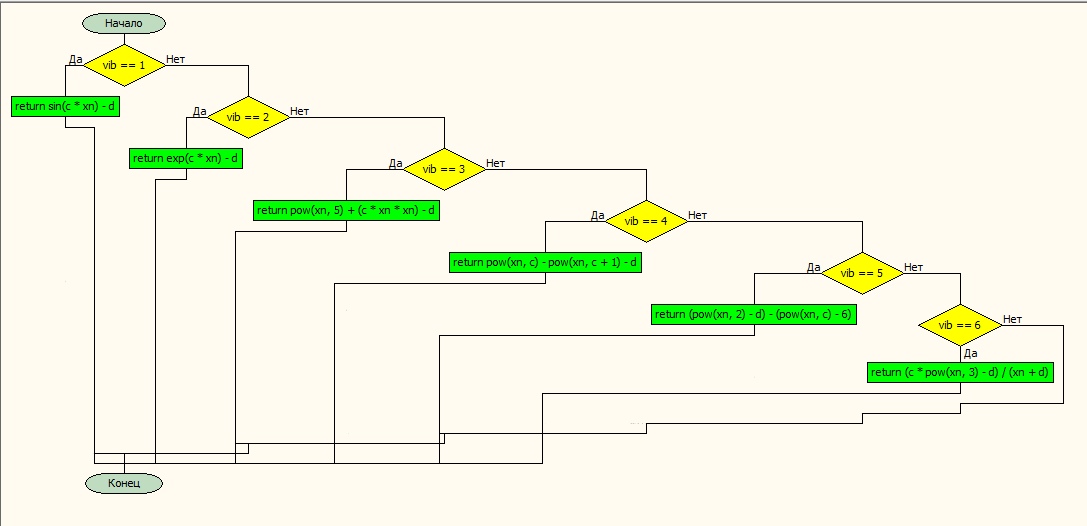
Блок-схема функции PROIZVOD:



**Блок-схема 1.2.1 (см. стр. 25)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Переменная** | **Тип** | **Область изменения** | **Начальное значение** |
| **xn** | **double** | **PROIZVOD** | **задается в main** |
| **c** | **double** | **PROIZVOD** | **задается в main** |
| **d** | **double** | **PROIZVOD** | **задается в main** |
| **vib** | **int** | **PROIZVOD** | **задается в main** |

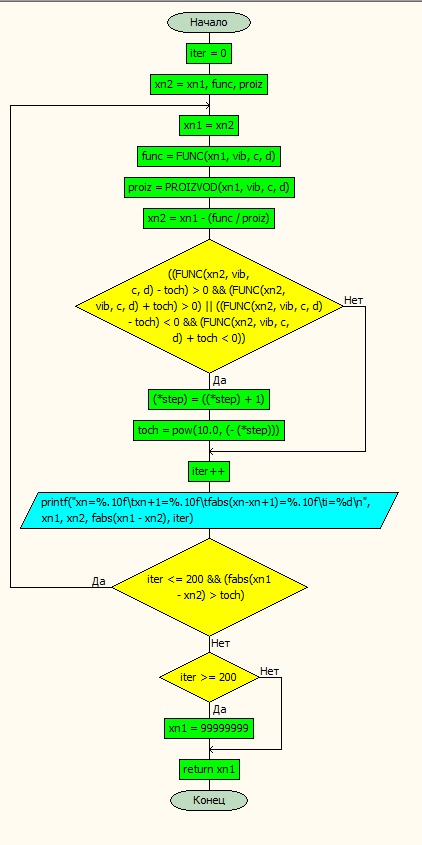
Блок-схема функции FUNC:



**Блок-схема 1.2.2**

|  |  |  |  |
| --- | --- | --- | --- |
| **Переменная** | **Тип** | **Область изменения** | **Начальное значение** |
| **xn** | **double** | **FUNC** | **задается в main** |
| **c** | **double** | **FUNC** | **задается в main** |
| **d** | **double** | **FUNC** | **задается в main** |
| **vib** | **int** | **FUNC** | **задается в main** |

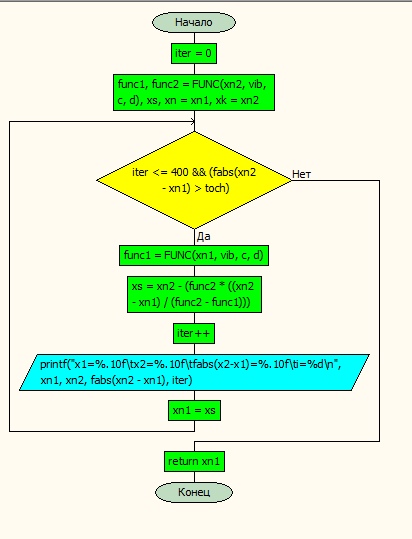
Блок-схема функции KASAT:



**Блок-схема 1.2.3 (см. стр. 27)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Переменная** | **Тип** | **Область изменения** | **Начальное значение** |
| **xn** | **double** | **KASAT** | **задается в main** |
| **c** | **double** | **KASAT** | **задается в main** |
| **d** | **double** | **KASAT** | **задается в main** |
| **vib** | **int** | **KASAT** | **задается в main** |
| **xn2** | **double** | **KASAT** | **xn1** |
| **func** | **double** | **KASAT** | **задается в ходе цикла** |
| **prois** | **double** | **KASAT** | **задается в ходе цикла** |
| **iter** | **int** | **KASAT** | **0** |
| **toch** | **double** | **KASAT** | **задается в main** |
| **step** | **double** | **KASAT, main** | **задается в main** |

Блок-схема функции SEKUSH:



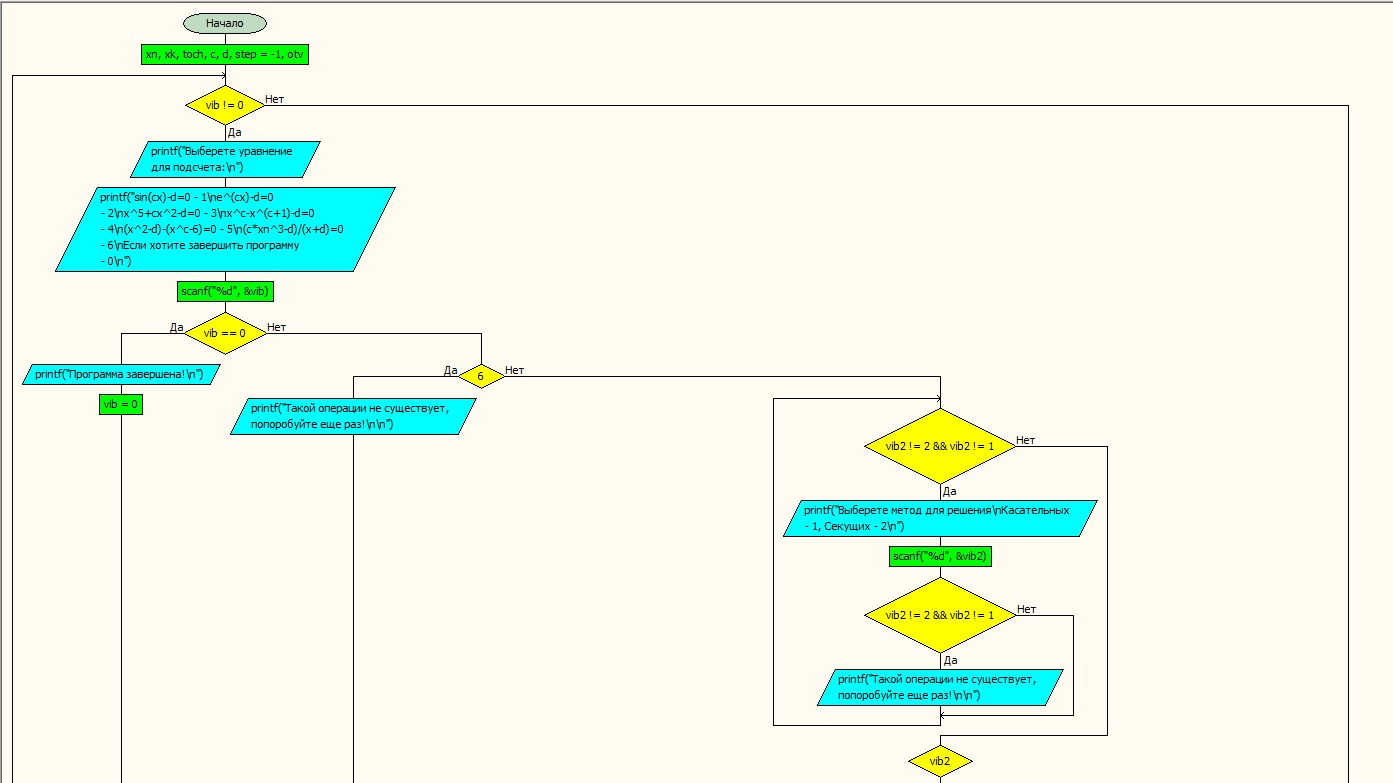
**Блок-схема 1.2.4**

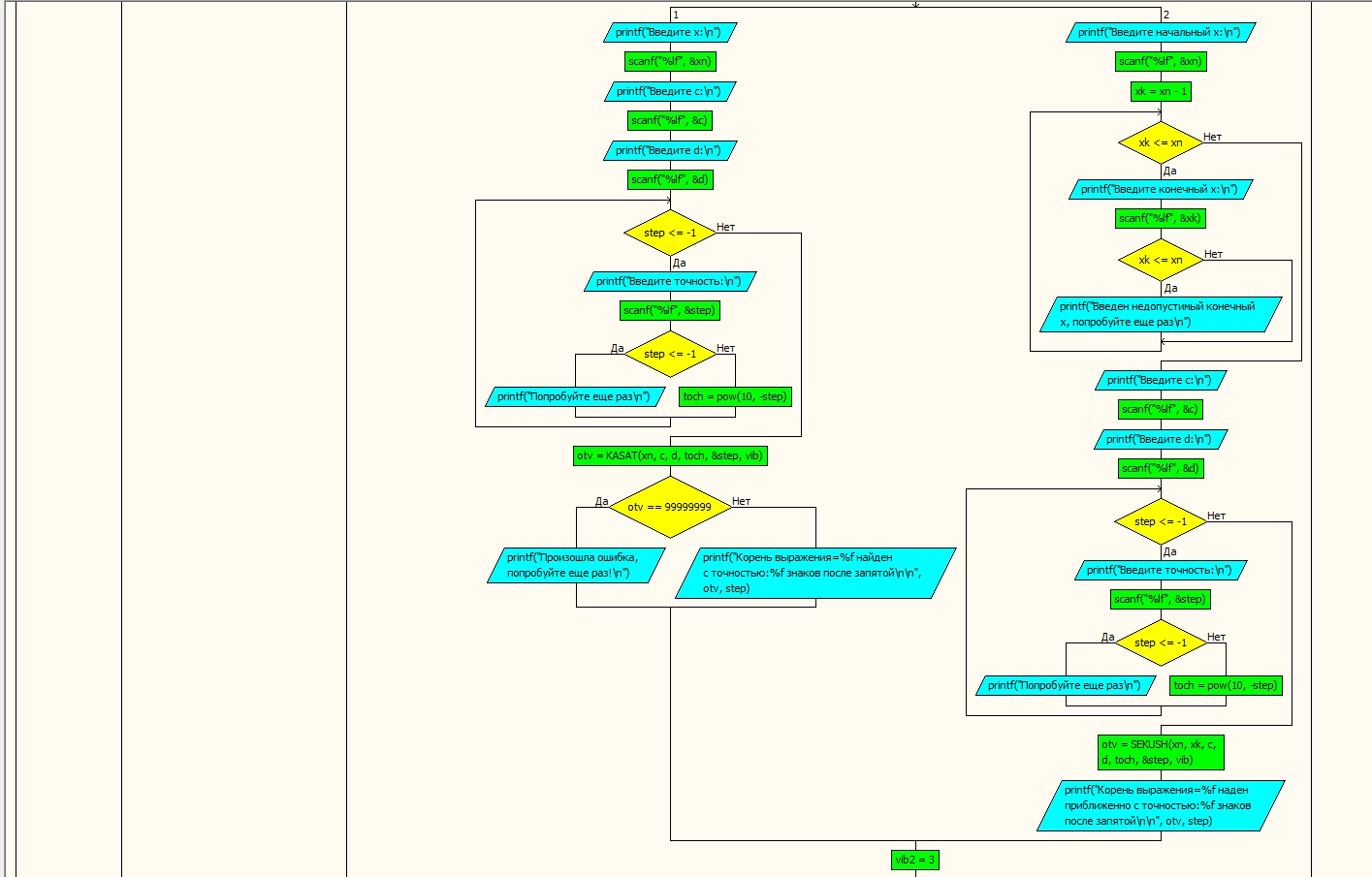
|  |  |  |  |
| --- | --- | --- | --- |
| **Переменная** | **Тип** | **Область изменения** | **Начальное значение** |
| **xn1** | **double** | **SEKUSH** | **задается в main** |
| **c** | **double** | **SEKUSH** | **задается в main** |
| **d** | **double** | **SEKUSH** | **задается в main** |
| **vib** | **int** | **SEKUSH** | **задается в main** |
| **xn2** | **double** | **SEKUSH** | **задается в main** |
| **func1** | **double** | **SEKUSH** | **задается в ходе цикла** |
| **func2** | **double** | **SEKUSH** | **FUNC(xn2,vib,c,d)** |
| **iter** | **int** | **SEKUSH** | **0** |
| **toch** | **double** | **SEKUSH** | **задается в main** |
| **xn** | **double** | **SEKUSH** | **xn1** |
| **xk** | **double** | **SEKUSH** | **xn2** |
| **xs** | **double** | **SEKUSH** | **задается в ходе цикла** |

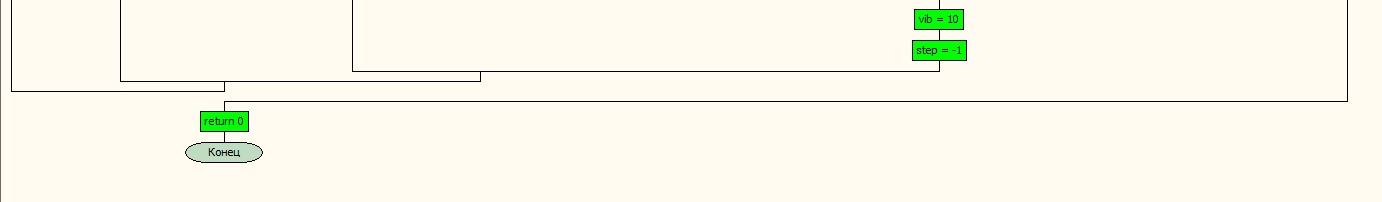
Блок-схема функции main:

|  |  |  |  |
| --- | --- | --- | --- |
| **Переменная** | **Тип** | **Область изменения** | **Начальное значение** |
| **xn** | **double** | **main** | **задается далее** |
| **c** | **double** | **main** | **задается далее** |
| **d** | **double** | **main** | **задается далее** |
| **vib** | **int** | **main** | **задается далее** |
| **xk** | **double** | **main** | **задается далее** |
| **otv** | **double** | **main** | **задается далее** |
| **step** | **double** | **main** | **-1** |
| **vib2** | **int** | **main** | **задается далее** |
| **toch** | **double** | **main** | **задается далее** |

Блок-схема расположена на стр. 31-33

****

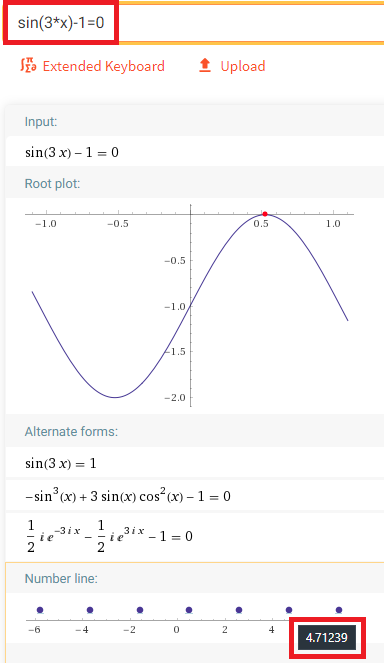
****

****

**Блок-схема 1.2.5**

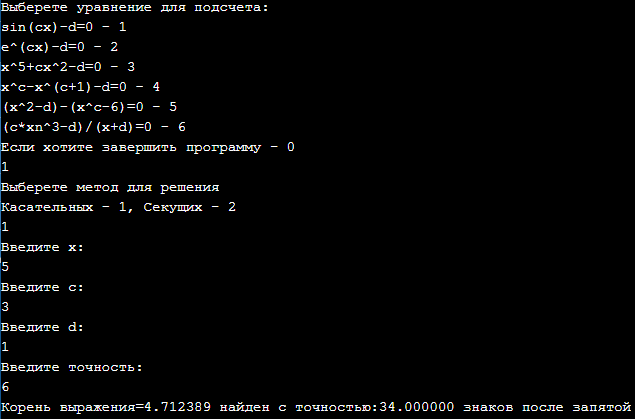
Проверка работоспособности программы:

1) Уравнение sin(c\*x)-d=0 (см. Скриншот 1.2.1, 1.2.2, 1.2.3)



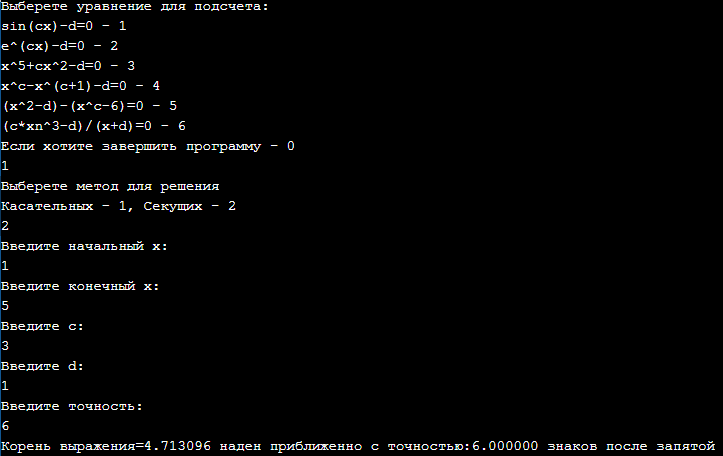
**Скриншот 1.2.1**

1.1)Решение с помощью метода касательных (см. Скриншот 1.2.2)



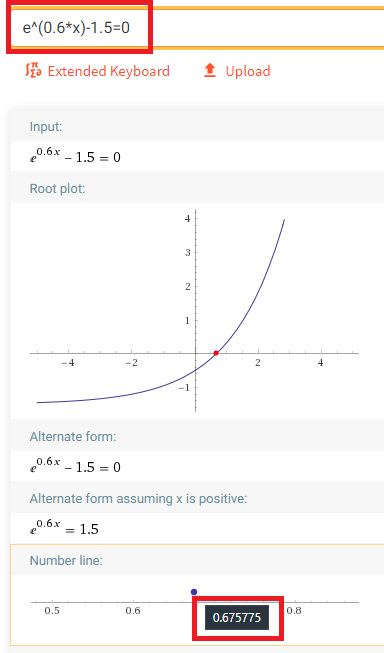
**Скриншот 1.2.2**

1.2)Решение с помощью метода секущих (см. Скриншот 1.2.3)



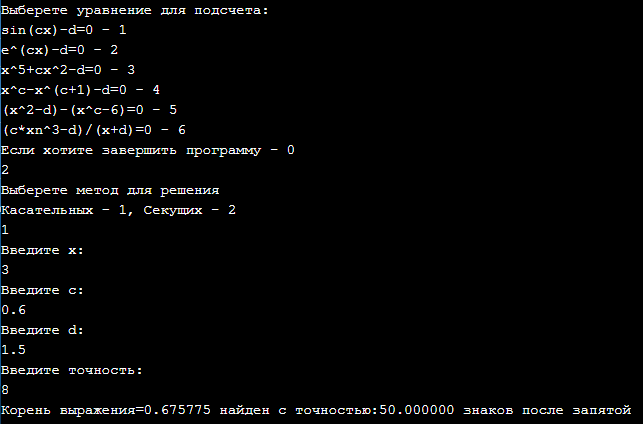
**Скриншот 1.2.3 (см. стр. 35)**

2) Уравнение: e^(c\*x)-d=0 (см. Скриншот 1.2.4, 1.2.5, 1.2.6)



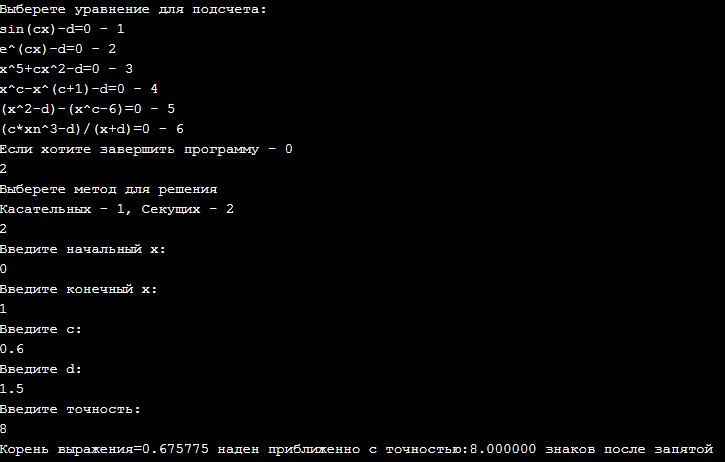
**Скриншот 1.2.4**

2.1)Решение с помощью метода касательных (см. Скриншот 1.2.5)



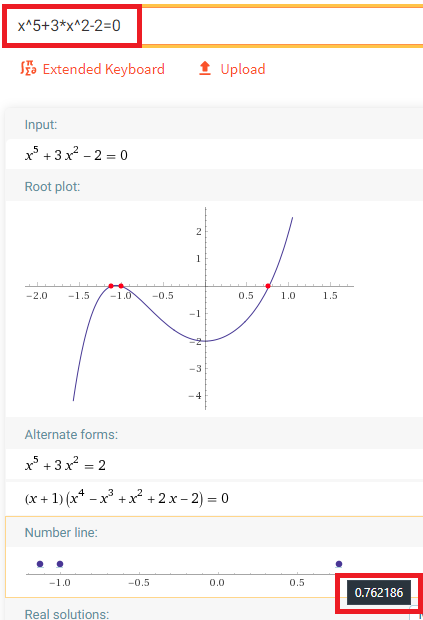
**Скриншот 1.2.5**

2.2)Решение с помощью метода секущих (см Скриншот 1.2.6)



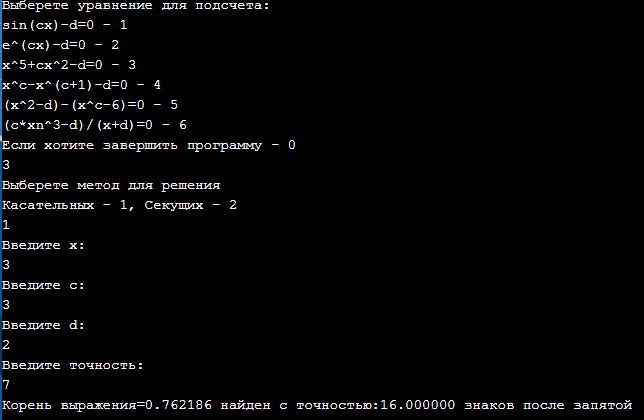
**Скриншот 1.2.6**

3) Уравнение: x^5+c\*x^2-d=0 (см Скриншот 1.2.7, 1.2.8, 1.2.9)



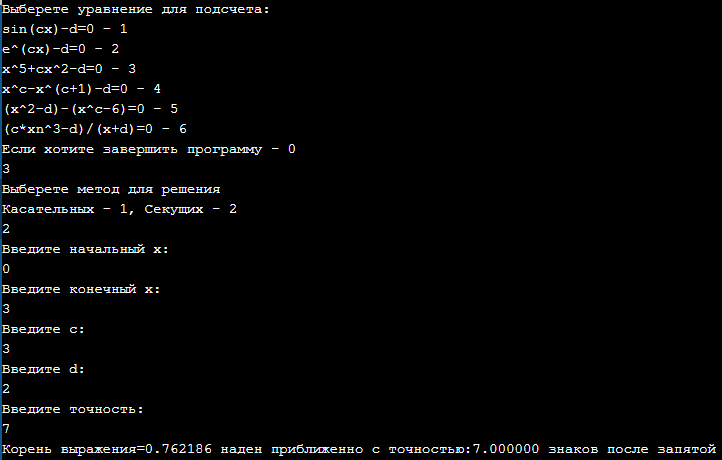
**Скриншот 1.2.7**

3.1) )Решение с помощью метода касательных (см. Скриншот 1.2.8, стр. 39)



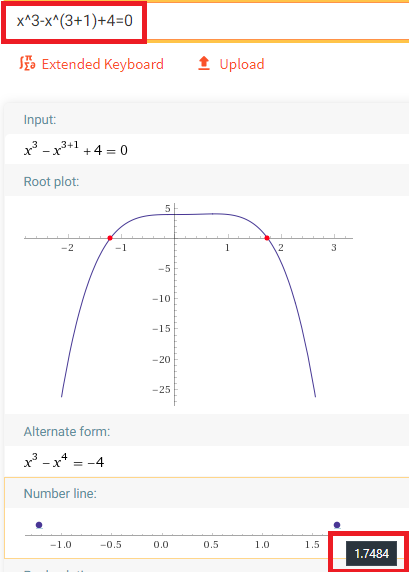
**Скриншот 1.2.8**

3.2)Решение с помощью метода секущих (см Скриншот 1.2.9)

****

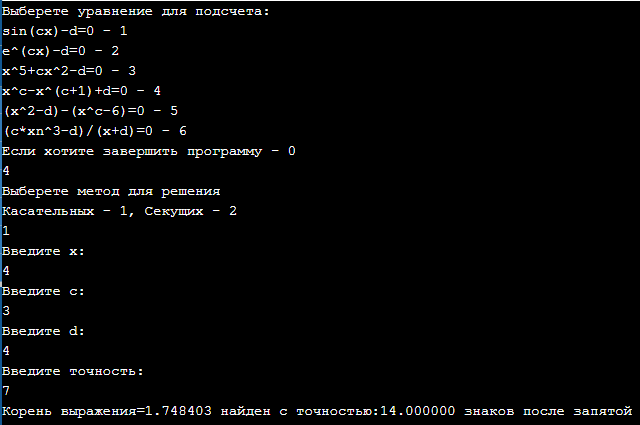
**Скриншот 1.2.9**

4) Уравнение: x^c-x^(c+1)+d=0 (см. Скриншот 1.2.10, 1.2.11, 1.2.12)



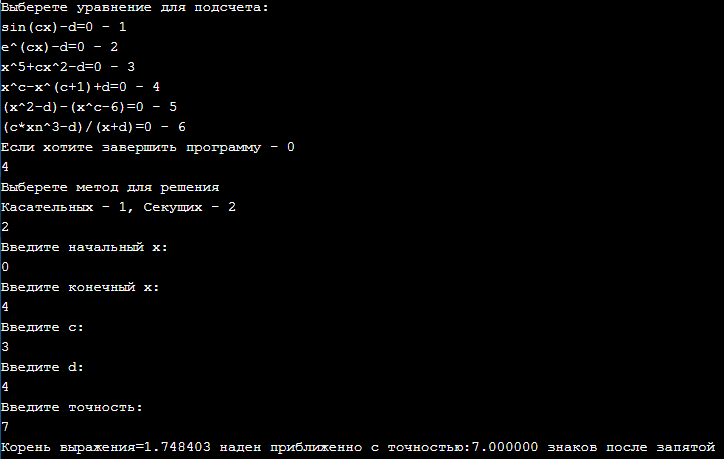
**Скриншот 1.2.10**

4.1) Решение с помощью метода касательных (см. Скриншот 1.2.11, стр. 41)



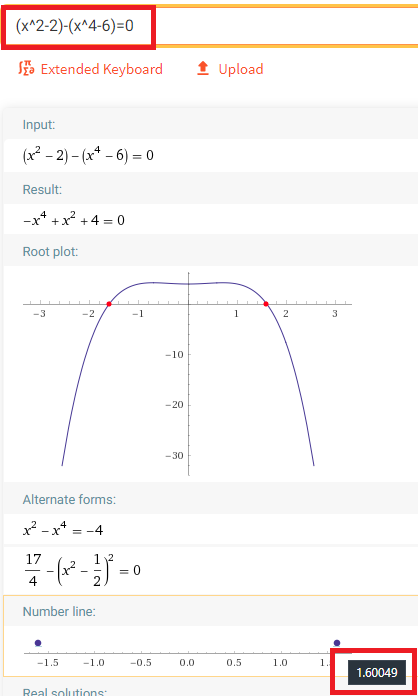
**Скриншот 1.2.11**

4.2)Решение с помощью метода секущих (см. Скриншот 1.2.12)



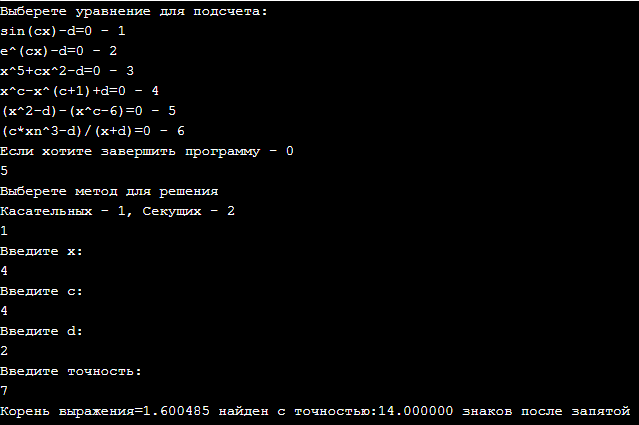
**Скриншот 1.2.12**

5) Уравнение: (x^2-d)-(x^c-6)=0 (см. Скриншот 1.2.13, 1.2.14, 1.2.15)



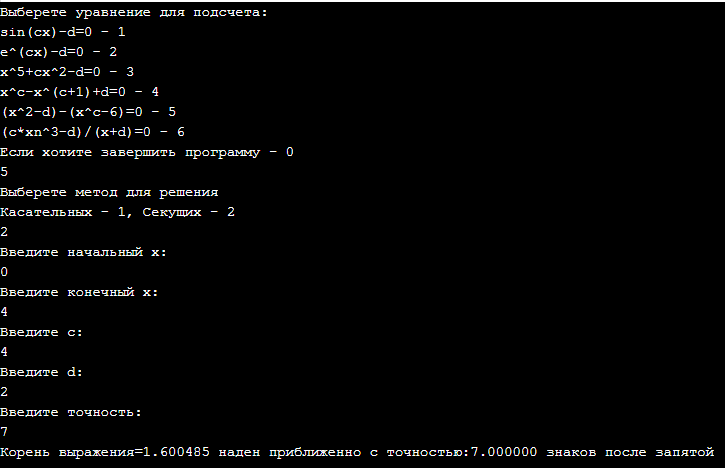
**Скриншот 1.2.13**

5.1) Решение с помощью метода касательных (см. Скриншот 1.2.14, стр. 43)



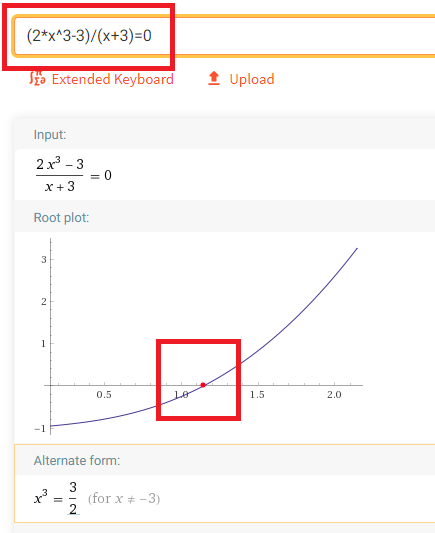
**Скриншот 1.2.14**

5.2)Решение с помощью метода секущих (см. Скриншот 1.2.15)



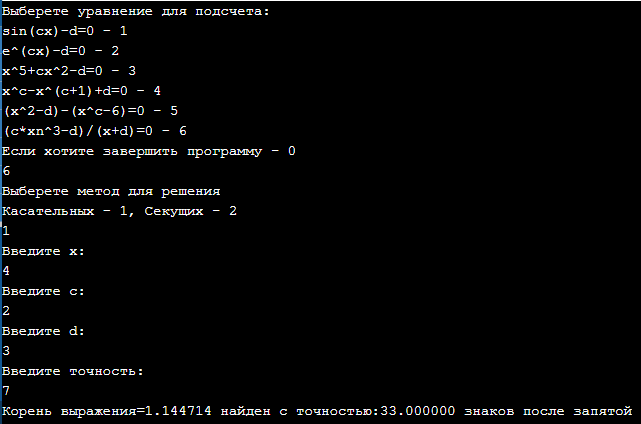
**Скриншот 1.2.15**

6) Уравнение: (c\*xn^3-d)/(x+d)=0 (см. Скриншот 1.2.16, 1.2.17, 1.2.18)



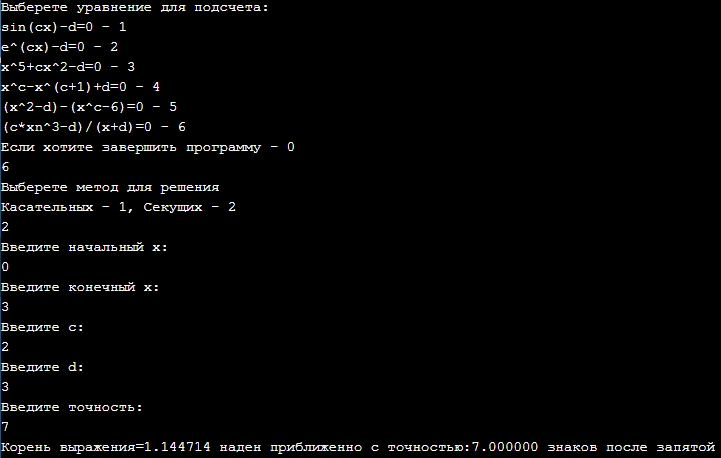
**Скриншот 1.2.16**

6.1) Решение с помощью метода касательных (см. Скриншот 1.2.17, стр. 45)



**Скриншот 1.2.17**

5.2)Решение с помощью метода секущих (см. Скриншот 1.2.18)



**Скриншот 1.2.18**

Вывод: как можно заметить, поиск приближенных корней уравнения с помощью методов секущих и касательных вполне реален. Вычисление происходит с некоторыми погрешностями, что в данном случае допустимо, так как методы реализуют именно поиск приближенного значения. Так же, стоит заметить, что методу секущих иногда требуется больше итераций для поиска приближенного значения. Так как метод секущих может искать приближенное значение почти бесконечно, стоит всегда ограничивать его по количеству итераций. Методы просты в понимании и в реализации, однако в методе касательных нужно вычислять производную, а в методе секущих нет. Это стоит учитвоать, используя их для вычисления приближенных значений.

**ЧАСТЬ №2.РАЗРАБОТКА ИГРОВОЙ ПРОГРАММЫ**

Разработка игр – сложный и кропотливый процесс[9]. В данной сфере программирования много вниманию уделяется не только алгоритмам, но и психологии людей. Ведь популярность игр заключается именно в том, что они затрагивают эмоции человека. Мало написать программу стрельбы по кружочкам, нужно дать человеку цель, для чего он будет это делать. Возможно, эти кружочки – вражеские захватчики по сюжету игры, а может, за каждое попадание по кружочкам человек будет получать какое-то поощрение, что пробудит в нем дух соревнования. Даже то, что игра имеет конец, до которого нужно дойти, уже пробуждает в человеке желание в нее поиграть.

В данной теме мы разберем особенности разработки игры «Устный счет».

Дано: основная суть игры заключается в том, что программа генерирует случайное арифметическое выражение, в котором могут присутствовать скобки, после вычисляет его значение и предлагает пользователю решить его. В выражении могут генерироваться сложение, вычитание, деление, умножение и возведение степень. Помимо основной части, игра должна иметь 3 дополнения к своему функционалу.

Как можно заметить, в целом программа состоит из трех блоков – случайной генерации выражения, последующего его вычисления и предоставления возможности пользователю решить его. Обозначим блоки как

2.1 Случайная генерация выражения

2.2 Вычисление и вывод выражения

2.3 Проверка ответа пользователя и игровая часть

Решение: далее в Листинге 2.1(см. стр. 46-59) будет представлена моя реализация кода игры, с некоторыми сносками, разбор кода будет происходить по блокам, которые были обозначены ранее.

|  |
| --- |
| #include<locale.h>  #include<stdio.h>  #include<math.h>  #include<stdlib.h>  #include<time.h>  void ZAPZN(char\*, int, int);  //функция заполенения массива знаков  void ZAPZN(char\* ZNAK, int kol, int skob)  {  int sk = skob;  for (int i = ((kol - 2) - skob); i >= 1; i -= 2)//от (количества символов-количество скобок) //заполняем через каждые две ячейки, так как помимо знаков одновременно вставляются еще //и скобки  {  int c = 1 + rand() % 5;//генерируется случайное число от 1 до 5, а потом в зависимости от //числа в ячейку массива вставляется один из 5 знаков и скобка в конец цепочки и в перед //знаком  if (c == 1) { ZNAK[i] = '+'; ZNAK[(kol - 2) - sk + 1] = ')'; sk--; ZNAK[i - 1] = '('; }  else if (c == 2) { ZNAK[i] = '-'; ZNAK[(kol - 2) - sk + 1] = ')'; sk--; ZNAK[i - 1] = '('; }  else if (c == 3) { ZNAK[i] = '\*'; ZNAK[(kol - 2) - sk + 1] = ')'; sk--; ZNAK[i - 1] = '('; }  else if (c == 4) { ZNAK[i] = '/'; ZNAK[(kol - 2) - sk + 1] = ')'; sk--; ZNAK[i - 1] = '('; }  else if (c == 5) { ZNAK[i] = '^'; ZNAK[(kol - 2) - sk + 1] = ')'; sk--; ZNAK[i - 1] = '('; }  }  }  void ZAPCH(int\*, int, int, int);  //функция заполнения массива чисел. массив чисел нужен для более простой генерации //чисел, если пользователь захочет задать диапазон генерации сам  void ZAPCH(int\* CHIS, int kol, int a, int b)  {  int ot = a, doo = b - a + 1;//так как функция универсальна и для стандартного диапазона, и //для введенного пользователем, в самой функции по формуле приходится вычислять начало //и конец диапазона генерации из-за специфики генерации чисел с помощью rand  for (int i = 0; i < kol; i++)  {  CHIS[i] = ot + rand() % doo;  /\*printf("%d ", CHIS[i]);\*/  }  /\*printf("\n");\*/  }  void PREOBRAZ(char\*, int, int\*, int\*);  //функция-преобразователь массива знаков. в зависимости от приоритета убираются лишние //скобки, переменная отвечающая за скобки передана как указатель, чтобы менять ее и в //мейне  void PREOBRAZ(char\* VIR, int kol, int\* skob, int\* nuli)  {  int i;  for (i = ((kol - 2) - (\*skob)); i >= 3; i -= 2)//начиная с самого последнего знака(не считая //скобки) цикл бежит по знакам до 2 с начала  {  //в зависимости от приоритетности каждый знак сравнивается с предыдущим, чтобы понять, //нужно убирать скобки или нет,если скобки не нужны, они обнуляются, как открывающаяся //так и закрывающаяся  if ((VIR[i] == '+') && VIR[i - 2] != '^' && VIR[i - 2] != '\*' && VIR[i - 2] != '/' && VIR[i - 2] != '-') { VIR[(kol - 2) - (\*skob) + 1] = '0'; VIR[i - 1] = '0'; (\*skob)--; }  else if ((VIR[i] == '-') && VIR[i - 2] != '^' && VIR[i - 2] != '\*' && VIR[i - 2] != '/' && VIR[i - 2] != '-') { VIR[(kol - 2) - (\*skob) + 1] = '0'; VIR[i - 1] = '0'; (\*skob)--; }  else if ((VIR[i] == '\*') && VIR[i - 2] != '^' && VIR[i - 2] != '/') { VIR[(kol - 2) - (\*skob) + 1] = '0'; VIR[i - 1] = '0'; (\*skob)--; }  else if ((VIR[i] == '/') && VIR[i - 2] != '^' && VIR[i - 2] != '\*' && VIR[i - 2] != '/') { VIR[(kol - 2) - (\*skob) + 1] = '0'; VIR[i - 1] = '0'; (\*skob)--; }  }  if (i == 1 && VIR[i - 1] == '(') { VIR[i - 1] = '0'; VIR[kol - 2] = '0'; (\*skob)--; }//перед самым //первым знаком скобка не обязательна, поэтому она убирается  int d = kol - 2;  while (d != 0)//данный цикл сдвигает все нули в конец, чтобы потом уменьшить //размерность массива, чтобы он не занимал память  {  for (i = 1; i <= (kol - 2); i++)  {  if (VIR[i - 1] == '0')  {  VIR[i - 1] = VIR[i];  VIR[i] = '0';  }  }  d--;  }  for (i = 1; i <= (kol - 2); i++) { if (VIR[i] == '0') (\*nuli)++; }  //данный цикл считает количество нулей, чтобы после можно было уменьшить размерность //массива на правильное кол-во знаков  }  void VIVOD(char\*, int\*, int);  //данная функция, это функция вывода. единственная ее функция - вывести на экран цельное //получившееся выражение, которое на самом деле состоит из двух массивов - знаков и чисел  void VIVOD(char\* VIR, int\* CHIS, int kol)  {  int j = 0, i;  for (i = 0; i < kol - 1; i++)//данный масиив, в опредленном порядке выводит элементы //массива знаков и массива чисел  {  //отличительная особенность данного вывода в том, что если считываемое число оказалось //отрицательным, оно берется в скобки, чтобы было понятно, что минус относится конкретно //к числу  if (VIR[i] != '(' && VIR[i] != ')' && VIR[i + 1] != ')' && CHIS[j] > 0) { printf("%d%c", CHIS[j], VIR[i]); j++; }  else if (VIR[i] != '(' && VIR[i] != ')' && VIR[i + 1] != ')' && CHIS[j] < 0) { printf("(%d)%c", CHIS[j], VIR[i]); j++; }  else if (VIR[i] != '(' && VIR[i] != ')' && VIR[i + 1] == ')' && CHIS[j] > 0 && CHIS[j + 1] > 0) { printf("%d%c%d%c", CHIS[j], VIR[i], CHIS[j + 1], VIR[i + 1]); i++; j += 2; }  else if (VIR[i] != '(' && VIR[i] != ')' && VIR[i + 1] == ')' && CHIS[j] < 0 && CHIS[j + 1]>0) { printf("(%d)%c%d%c", CHIS[j], VIR[i], CHIS[j + 1], VIR[i + 1]); i++; j += 2; }  else if (VIR[i] != '(' && VIR[i] != ')' && VIR[i + 1] == ')' && CHIS[j] > 0 && CHIS[j + 1] < 0) { printf("%d%c(%d)%c", CHIS[j], VIR[i], CHIS[j + 1], VIR[i + 1]); i++; j += 2; }  else if (VIR[i] != '(' && VIR[i] != ')' && VIR[i + 1] == ')' && CHIS[j] < 0 && CHIS[j + 1] < 0) { printf("(%d)%c(%d)%c", CHIS[j], VIR[i], CHIS[j + 1], VIR[i + 1]); i++; j += 2; }  else if (VIR[i] == '(') printf("%c", VIR[i]);  else if (VIR[i] == ')' && VIR[i - 1] == ')') printf("%c", VIR[i]);  if (i == kol - 2 && VIR[i] != '(' && VIR[i] != ')' && CHIS[j] > 0) printf("%d", CHIS[j]);  else if (i == kol - 2 && VIR[i] != '(' && VIR[i] != ')' && CHIS[j] < 0) printf("(%d)", CHIS[j]);  }  printf("\n");  }  float SCHET(char\*, int\*, int, int, int);  //данная функция отвечает за вычисление всего выражения  float SCHET(char\* VIR, int\* CHIS, int kol, int skob, int chis)  {  int i = kol, j = chis - 2;  float schet = (float)CHIS[chis - 1];  for (i = kol; i >= 0; i--)//влгоритм вычисления достаточно простой, просто проверка знаков //и соответствующие им действия  {  if (VIR[i] == '^')  {  schet = pow((float)CHIS[j], schet); j--;  }  else if (VIR[i] == '\*')  {  schet = (float)CHIS[j] \* schet; j--;  }  else if (VIR[i] == '/')  {  schet = (float)CHIS[j] / schet; j--;  }  else if (VIR[i] == '-')  {  schet = CHIS[j] - schet; j--;  }  else if (VIR[i] == '+')  {  schet = CHIS[j] + schet; j--;  }  }  return schet;  }  int main()  {  setlocale(LC\_ALL, "ru");//данная функция нужна для русификации текста и особой роли не играет  int i, j = 0, chisl, skob = 0, jach = 0, nuli = 0, a, b, ball = 0, proverka, itog2;  int vib = 10;  float itog;  srand(time(NULL));  while (vib != 0 && ball <= 1000)//данный цикл будет работать, пока пользователь не //захочет закончить игру, либо пока не наберет 1000 быллов  {  if (ball > 0) printf("\tКоличество ваших баллов: %d\n", ball);//после каждого решенного //уравнения пользователь будет видеть количество набранных баллов  else printf("\tПока у вас нет баллов :(\n");  printf("Чтобы прочитать правила игры нажмите :3\nЕсли хотите задать количество чисел вручную нажмите :1\nЕсли хотите, чтобы количество чисел было задано случайно, нажмите :2\nЕсли хотите завершить программу, нажмите :0\n");  scanf("%d", &vib);//для начала узнается то, какое количество чисел будет задано - //вручную или рандомное количество, либо же пользователь завершит программу  switch (vib)  {  case 1://данный кейс будет работать, если пользователь задает количество чисел сам  printf("Если хотите задать диапазон чисел вручную, нажмите :1\nЕсли хотите выбрать стандартный диапазон(от 1 до 10 включительно) нажмите :2\n");  scanf("%d", &vib);//а здесь считывается то, какой будет использован диапазон значений  if (vib == 1)//это условие отвечает за введенное вручную кол-во чисел и введенный //вручную диапазон  {  printf("Введите диапазон от:"); scanf("%d", &a);  printf("До:"); scanf("%d", &b);  if (a > b) { printf("Задан неверный дипазон!\n\n"); }//проверка на ошибку и вывод //сообщения о ней, если она все же произошла  else  {  printf("Введите количество чисел:"); scanf("%d", &chisl);  if (chisl <= 1) printf("Введено недопустимое количество чисел!\n\n");  else  {  printf("\n");  int\* CHIS = (int\*)malloc(chisl \* sizeof(int));//выделяется память под массив //чисел  ZAPCH(CHIS, chisl, a, b);//массив чисел заполняется в соответствии с //даипазоном  skob = chisl - 1;//назначается максимально возможное кол-во закрытых скобок //для выражения  jach = (chisl - 1) \* 3 + 1;//определяется количество ячеек в соответствии с //количеством чисел  char\* VIR = (char\*)malloc(jach \* sizeof(char));//выделяется память под массив //знаков  ZAPZN(VIR, jach, skob);//массив знаков заполняется  PREOBRAZ(VIR, jach, &skob, &nuli);//и преобразовывается  jach = (chisl - 1) \* 3 + 1 - nuli;//количество ячеек меняется в зависимости от //того, сколько образовалось нулей после преобразования  nuli = 0;//количество нулей обнуляется  VIR = (char\*)realloc(VIR, jach \* sizeof(char));//уменьшение длинны массива до //нового количества ячеек  VIR[jach - 1] = '\0';//из массива символов делаем строку  VIVOD(VIR, CHIS, jach);//выводим на экран выражение  itog = SCHET(VIR, CHIS, jach - 2 - skob, skob, chisl);//считаем наше выражение  free(CHIS);//освобождаем память выделенную под массив чисел  free(VIR);//освобождаем память выделенную под массив знаков  itog2 = (int)itog;//округляем ответ до целого  if (itog < 10000 && itog>-10000)//если ответ получился достаточно считаемым //для челоека, то...  {  printf("=%f %d\n\n", itog, (int)itog);//данная строка нужна для проверки //правильности вычисления, однако ее можно закомментировать, дабы пользователь не видел //ответ  printf("Введите свой ответ(округлите до целого, если получилось не целое число):\n"); scanf("%d", &proverka);  //пользователю предлагается ввести ответ, если //он угадал, ему начисляются баллы, если нет - отнимаются  if (proverka == itog2)  {  printf("Правильно! Вам начислено 100 баллов!\n\n");  ball += 100;  }  else if (proverka != itog2)  {  printf("Увы, Вы ошиблись :( С Вас снимаются 50 баллов\n\n");  ball -= 50;  }  }  else if (itog > 10000)printf("Значение выражения вышло слишком большим, попробуйте еще раз!\n\n");  else if (itog < 10000)printf("Значение выражения вышло слишком маленьким, попробуйте еще раз!\n\n");  //если получислось так, что ответ оказался слишком большим или слишком //маленьким(трудным для вычисления)пользователю выводится уведомление об этом и //баллы не снимаются  }  }    }  else if (vib == 2)//это условие отвечает за введенное вручную кол-во чисел и //выбранный стандартный диапазон  {  a = 1; b = 10;//переменным отвечающим за диапазон присваиваем стандартные //значения, далее алгоритм не отличается от vib==1  printf("Введите количество чисел:"); scanf("%d", &chisl);  printf("\n");  if (chisl <= 1) printf("Введено недопустимое количество чисел!\n\n");  else  {  int\* CHIS = (int\*)malloc(chisl \* sizeof(int));  ZAPCH(CHIS, chisl, a, b);  skob = chisl - 1;  jach = (chisl - 1) \* 3 + 1;  char\* VIR = (char\*)malloc(jach \* sizeof(char));  ZAPZN(VIR, jach, skob);  PREOBRAZ(VIR, jach, &skob, &nuli);  jach = (chisl - 1) \* 3 + 1 - nuli;  nuli = 0;  VIR = (char\*)realloc(VIR, jach \* sizeof(char));  VIR[jach - 1] = '\0';  VIVOD(VIR, CHIS, jach);  itog = SCHET(VIR, CHIS, jach - 2 - skob, skob, chisl);  free(CHIS);  free(VIR);  itog2 = (int)itog;  if (itog < 10000 && itog>-10000)  {  printf("=%f %d\n\n", itog, (int)itog);  printf("Введите свой ответ(округлите до целого, если получилось не целое число):\n"); scanf("%d", &proverka);  if (proverka == itog2)  {  printf("Правильно! Вам начислено 150 баллов!\n\n");  ball += 150;  }  else if (proverka != itog2)  {  printf("Увы, Вы ошиблись :( С Вас снимаются 50 баллов\n\n");  ball -= 50;  }  }  else if (itog > 10000)printf("Значение выражения вышло слишком большим, попробуйте еще раз!\n\n");  else if (itog < 10000)printf("Значение выражения вышло слишком маленьким, попробуйте еще раз!\n\n");  }  }  else if (vib != 1 || vib != 2) printf("Такой операции не существует, попробуйте еще раз!\n\n");//если пользователь ввел несуществующую операцию, на экран выведется //сообщение об этом  break;  case 2://данный кейс будет работать, если пользователь выберет случайное кол-во чисел,  //алгоритм вычисления выражения в нем почти не отличается от 1 кейса, однако есть пару //нюансов которые далее будут закомментированны  printf("Если хотите задать диапазон чисел вручную, нажмите :1\nЕсли хотите выбрать стандартный диапазон(от 1 до 10 включеительно) нажмите :2\n");  scanf("%d", &vib);  if (vib == 1)  {  printf("Введите диавпазон от:"); scanf("%d", &a);  printf("До:"); scanf("%d", &b); printf("\n");  if (a > b) { printf("Задан неверный дипазон!\n\n"); }  else  {  chisl = 2 + rand() % 9;//данная строка отвечает за генерацию количества чисел, //могут сгенерироваться от 2 до 10 чисел, от 2 лишь потому,что генерация от меньшего //значения была бы бессмысленной  int\* CHIS = (int\*)malloc(chisl \* sizeof(int));  ZAPCH(CHIS, chisl, a, b);  skob = chisl - 1;  jach = (chisl - 1) \* 3 + 1;  char\* VIR = (char\*)malloc(jach \* sizeof(char));  ZAPZN(VIR, jach, skob);  PREOBRAZ(VIR, jach, &skob, &nuli);  jach = (chisl - 1) \* 3 + 1 - nuli;  nuli = 0;  VIR = (char\*)realloc(VIR, jach \* sizeof(char));  VIR[jach - 1] = '\0';  VIVOD(VIR, CHIS, jach);  itog = SCHET(VIR, CHIS, jach - 2 - skob, skob, chisl);  free(CHIS);  free(VIR);  itog2 = (int)itog;  if (itog < 10000 && itog>-10000)  {  printf("Введите свой ответ(округлите до целого, если получилось не целое число):\n"); scanf("%d", &proverka);  if (proverka == itog2)  {  printf("Правильно! Вам начислено 150 баллов!\n\n");  ball += 150;  }  else if (proverka != itog2)  {  printf("Увы, Вы ошиблись :( С Вас снимаются 50 баллов\n\n");  ball -= 50;  }  }  else if (itog > 10000)printf("Значение выражения вышло слишком большим, попробуйте еще раз!\n\n");  else if (itog < 10000)printf("Значение выражения вышло слишком маленьким, попробуйте еще раз!\n\n");  }  }  else if (vib == 2)  {  a = 1; b = 10;  chisl = 2 + rand() % 9;  int\* CHIS = (int\*)malloc(chisl \* sizeof(int));  ZAPCH(CHIS, chisl, a, b);  skob = chisl - 1;  jach = (chisl - 1) \* 3 + 1;  char\* VIR = (char\*)malloc(jach \* sizeof(char));  ZAPZN(VIR, jach, skob);  PREOBRAZ(VIR, jach, &skob, &nuli);  jach = (chisl - 1) \* 3 + 1 - nuli;  nuli = 0;  VIR = (char\*)realloc(VIR, jach \* sizeof(char));  VIR[jach - 1] = '\0';  VIVOD(VIR, CHIS, jach);  itog = SCHET(VIR, CHIS, jach - 2 - skob, skob, chisl);  free(CHIS);  free(VIR);  itog2 = (int)itog;  if (itog < 10000 && itog>-10000)  {  printf("=%f %d\n\n", itog, (int)itog);  printf("Введите свой ответ(округлите до целого, если получилось не целое число):\n"); scanf("%d", &proverka);  if (proverka == itog2)  {  printf("Правильно! Вам начислено 200 баллов!\n\n");  ball += 200;  }  else if (proverka != itog2)  {  printf("Увы, Вы ошиблись :( С Вас снимаются 50 баллов\n\n");  ball -= 50;  }  }  else if (itog > 10000)printf("Значение выражения вышло слишком большим, попробуйте еще раз!\n\n");  else if (itog < 10000)printf("Значение выражения вышло слишком маленьким, попробуйте еще раз!\n\n");  }  else if (vib != 1 || vib != 2) printf("Такой операции не существует, попробуйте еще раз!\n\n");  break;  case 3://данный кейс отвечает за вывод правил на экран  printf("\n");  printf("Добро пожаловать в игру Устный счет!\n");  printf("Суть игры заключается в том, что программа генерирует случайное математическое выражение,\nкоторое Вам предстаит решить.\n");  printf("За каждое решенное выражение Вы получите определенное количество баллов.\n");  printf("Колличество баллов зависит от того, какую операцию Вы Выберете.\n");  printf("Если Вы решите задать количество чисел вручную,\nа так же зададите диапазон значений сами, выйгрыш составит 100 баллов.\n\n");  printf("Если Вы решите задать количество чисел вручную,\nа так же выберете стандартный диапазон значений, выйгрыш составит 150 баллов.\n\n");  printf("Если Вы решите выбрать случайное количество чисел,\nа так же выберете стандартный диапазон значений, выйгрыш составит 200 баллов.\n\n");  printf("Если Вы решите выбрать случайное количество чисел,\nа так же зададите диапазон значений сами, выйгрыш составит 150 баллов.\n\n");  printf("За каждый пройгрыш с Вас будут сниматься 50 баллов\n");  printf("Когда количество ваших баллов составит 1000, Вы победите!\n");  printf("Если вышло так, что корень сгенерированного выражения оказался слишком большой/маленький, не волнуйтесь,\n");  printf("Ваши накопленные баллы никуда не исчезнут и вам представится еще одна попытка :)\n");  printf("Если так получилось, что корень выражения оказался с дробной частью, округлите его до целого в меньшую сторону (21.888=21)\n");  printf("Если Вы хотите закончить игру, просто выберете операцию 0\n");  printf("Удачи :)\n\n");  break;  case 0://данный кейс отвечает за завершение игры  printf("Накопленные баллы:%d Игра закончена!", ball);//на экран выводится //количество набранных баллов и сообщение об окончании игры  vib = 0;//переменная обнуляется, дабы цикл закончился  break;  default://если пользователь ввел любую иную операцию...  printf("\nТакой операции не существует, попробуйте еще раз!\n\n");//если //пользователь ввел несуществующую операцию, на экран выведется сообщение об этом  break;  }  if (ball >= 1000)//если после вычислений и ввода ответа пользователь набрал 1000 //баллов игра завершается  {  printf("Поздравляем, Вы набрали %d баллов и победили!\n", ball);  break;  }  }  return 0;  } |

**Листинг 2.1(см. стр. 46-59)**

2.1 Случайная генерация выражения

Как известно, генерация псевдослучайных чисел в языке Си представлена функцией rand() прототип которой лежит в заголовочном файле <stdlib.h>.[3] Однако данная функция генерирует числа из одной последовательности, поэтому при каждой компиляции они повторяются. Чтобы такого не происходило, в качестве аргумента функции srand() прототип которой так же лежит в <stdlib>, используется функция time() (которая объявлена в заголовочном файле <time.h>). В свою очередь аргументом функции time() должна быть константа NULL. Тогда данная функция вернет количество секунд, прошедших с 1 января 1970 года. Функция srand() принимает это количество и меняет последовательность, из которой генерируются псевдослучайные числа. Зная это, можно сгенерировать числа в заданных диапазонах. Для этого достаточно использовать данную формулу:

где А-число от которого нам нужно сгенерировать случайное число, а В – до которого.

Это работает потому, что функция rand() возвращает случайные числа из отрезка [0;RAND\_MAX], поэтому беря остаток от деления от rand() и числа с помощью % мы ограничиваем диапазон слева, а прибавляя число к rand() мы сдвигаем диапазон справа на это число. Однако ограничение с помощью остатка от деления всегда на 1 меньше, чем само число, на которое мы делим, поэтому мы прибавляем единицу, чтобы число тоже входило в отрезок. Так же из-за сдвига левой границы, нужно вычесть значение ее сдвига, когда двигаем правую границу.

Зная данную информацию, уже можно приступать к генерации выражения. Так как я решила, что моим первым дополнением к игре будет возможность задавать диапазон генерируемых значений вручную, в функцию генерации чисел ZAPCH (см. Листинг 2.1) я передаю не только количество самих чисел, но и переменные a и b, отвечающие за левую и правую границу диапазона значений соответственно. Внутри самой функции, по формуле, заданной выше, я вычисляю сдвиги для функции rand(). В функции main() пользователь может, как ввести границы самостоятельно, так и использовать стандартный диапазон от 1 до 10 включительно. Если был выбран стандартный диапазон, то функции ZAPCH передается именно он и ввода с клавиатуры пользователю не предоставляется.

Так же свойства функции rand() были использованы и в функции заполнения знаков ZAPZN. rand() возвращает целые числа, поэтому каждый знак я обозначила цифрой от 1 до 5.

В цикле генерировалось случайное число от 1 до 5, и в зависимости от того, какое оно было, в массив знаков вписывался нужный символ. Так же в конец массива знаков и перед сгенерированной математической операцией вставлялись скобки, закрытая и открытая соответственно. Визуализировать функцию ZAPZN можно так:

1. (\*)

2. (/(\*))

3.(+(/(\*)))

4.(-(+(/(\*))))

При генерации приоритеты математических операций не учитывались, поэтому для того, чтобы избавиться от лишних скобок я создала функцию PREOBRAZ, которая их удаляла, учитывая приоритеты операций. На месте лишней открывающейся скобки ставится ноль, соответствующая ей закрывающаяся скобка тоже обнуляется. В итоге, после преобразования массив знаков выглядит примерно так:

0-(+0/(\*)0)0

Далее, все в той же функции PREOBRAZ, все получившиеся нули сдвигаются в конец массива знаков:

-(+/(\*))0000

В функции main() массив знаков был задан не привычным способом A[], а как указатель, что позволяет нам с помощью функции realloc() перераспределить память, убрав из массива ненужные нули.

Вторым дополнением в игре является возможность сгенерировать количество чисел случайно, либо задать количество чисел с клавиатуры. Генерация количества чисел организована в функции main(), если пользователь выбрал случайную генерацию. Так как функции генерации, преобразования и вычисления выражения принимают в качестве аргумента количество чисел в выражении, они достаточно универсальны и могут быть использованы при любом количестве чисел.

2.2 Вычисление и вывод выражения

Перейдем к функциям вычисления и вывода на экран. Функцию для вычисления выражения я обозначила как SCHET. Данная функция работает достаточно просто, в цикле, с конца считываются математические операции. Они поочередно применяются к числам из массива чисел, которые тоже берутся с конца. То есть счет происходит слева направо. Такое возможно из-за специфики генерации знаков, которые точно так же генерировалось с конца. Полученный результат возвращается функцией в виде числа типа double.

Функция вывода VIVOD тоже не очень сложна. В цикле, начиная с первого элемента массива знаков, считываются символы. Если данный символ является операцией и после него не стоит открытая скобка, то выводится число из массива чисел и знак, который был считан. Если же считанный знак окажется открытой скобкой, то на экран выведется открытая скобка. Если следующий знак после считанного окажется закрывающейся скобкой, то выведется сначала число, потом операция, далее следующее число и закрывающаяся скобка. Ну и последний случай, если считанный знак окажется закрывающейся скобкой, то он просто выведется. Так же учтен случай, если в выражении скобок не оказалось вообще, тогда в таком случае последняя считанная математическая операция выведется как число, операция, последнее число.

Хоть и не рекомендуется использовать вывод на экран в функциях, однако в данном случае это самое оптимальное решение, учитывая что нужно соеденять два разных по типу массива – массив символов и массив чисел.

2.3 Проверка ответа пользователя и игровая часть

Половина дела сделана, осталось организовать саму игру. Для этого я использовала конструкцию switch в цикле. Это позволило добавить третье дополнение – накопление очков за каждое правильно вычисленное пользователем выражение. Для подсчета количества баллов я ввела переменную ball, которая увеличивается на определенное количество при правильно вычисленном выражении. Самое интересное, что сочетание разных выборов пользователя дает разное количество баллов. Так как пользователь может выбрать как случайное количество чисел, так и задать его самостоятельно, этот выбор создает две ветки вычисления выражения, которые я и разделила с помощью конструкции switch. Каждая ветка этой конструкции разделяется еще на две, потому что пользователь так же может выбрать диапазон значений сам, либо же выбрать стандартный. Итого мы получаем 4 конечных ветки.: случайное количество-заданный диапазон, заданное количество-заданный диапазон, случайное количество-стандартный диапазон, заданное количество-стандартный диапазон. Каждая из веток будет отличаться, хоть совсем и незначительно. Как уже говорилось ранее, при стандартном диапазоне сдвиги присваиваются в коде программы, а при заданном – определяются пользователем. Так же происходит и с количеством чисел.

Каждый выбор пользователя проверяется на корректность введения. Если же пользователь ввел некорректное значение или выбор программа сообщает ему об этом.

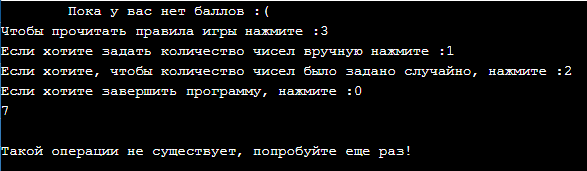
После вычисления значения выражения с помощью функции SCHET, это значение присваивается переменной и округляется до целого в меньшую сторону. Это сделано потому, что человек не сможет вычислить значение до такой же точности как компьютер, поэтому округление в данном случае – лучшее из решений. Если же так получилось, что значение сгенерированного выражения оказалось слишком большим, или слишком маленьким (меньше или больше 10000), то высвечивается уведомление об этом и пользователь может сгенерировать выражение заново, не теряя баллы. Далее ответ пользователя считывается и ему начисляются баллы, если он угадал, и вычитаются, если нет.

Так же, все в той же конструкции switch организован просмотр правил игры. Это особенно полезно, если пользователь забыл, за какие комбинации выборов сколько баллов дается.

Игра будет продолжаться ровно до того момента, пока пользователь не наберет 1000 очков, либо же не захочет закончить игру.

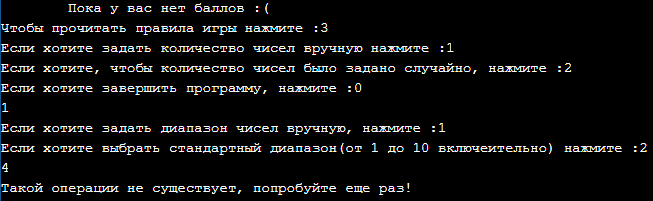
Проверка работоспособности:

1) Проверка некорректного ввода операции выбора из случайного количества чисел, заданного количества чисел, вывода правил и завершения программы



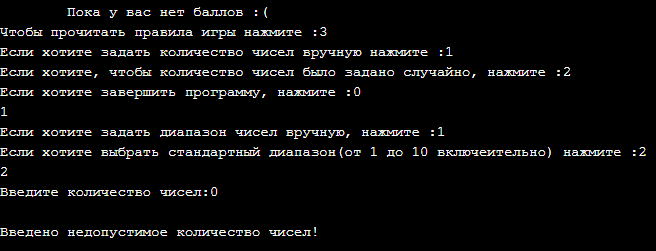
**Скриншот 2.1**

2) Проверка некорректного ввода выбора из заданного диапазона и стандартного (см. Скриншот 2.2 стр. 63)



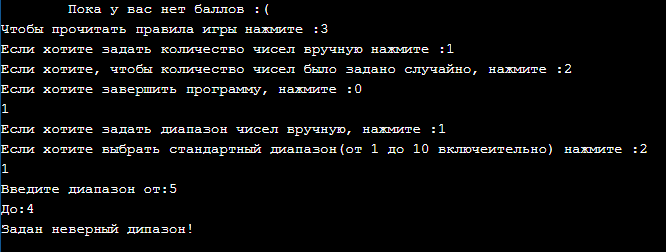
**Скриншот 2.2**

3)Проверка некорректного ввода количества чисел



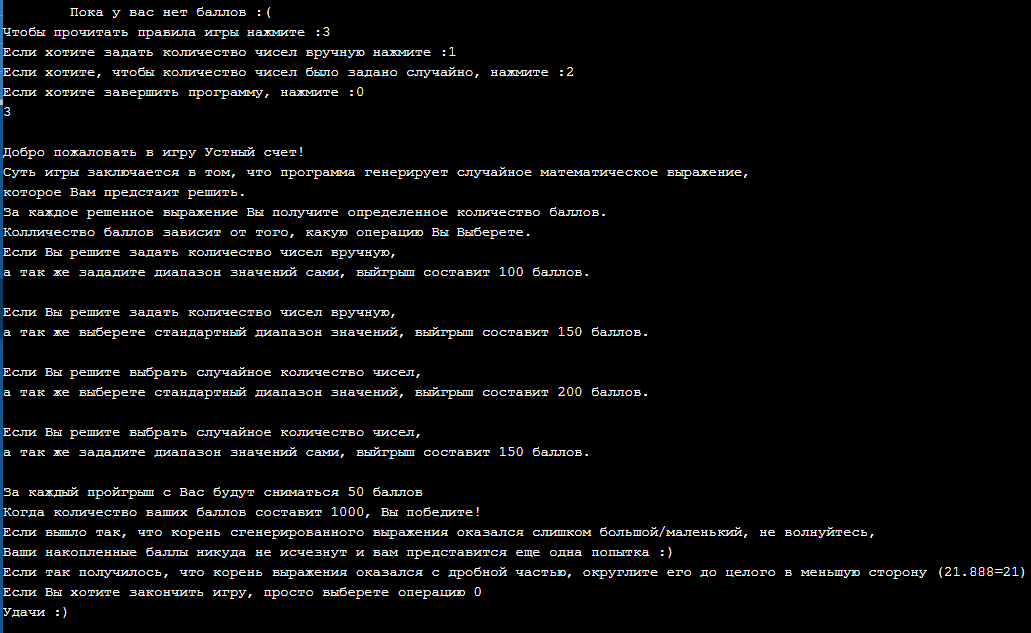
**Скриншот 2.3**

4)Проверка некорректного ввода диапазона значений



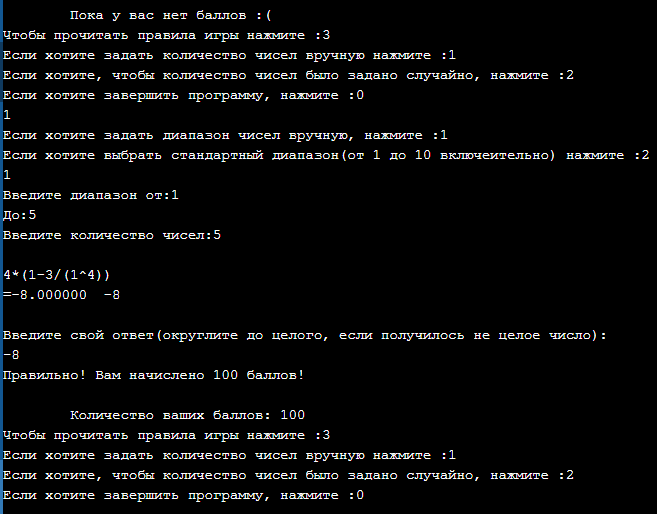
**Скриншот 2.4**

5)Проверка вывода правил(см. Скриншот 2.5 стр. 64)



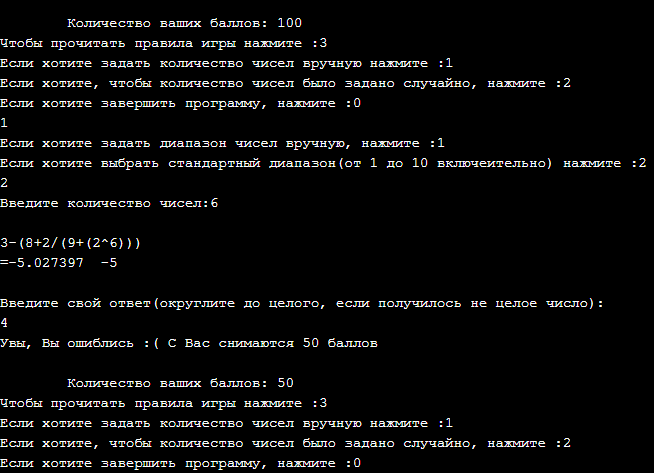
**Скриншот 2.5**

6)Проверка ввода правильного ответа с прибавлением баллов(см. Скриншот 2.6, стр.65)



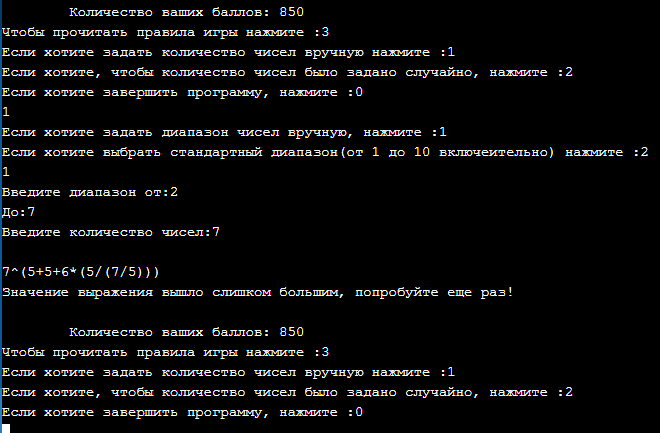
**Скриншот 2.6**

7) Проверка ввода неправильного значения со снятием баллов



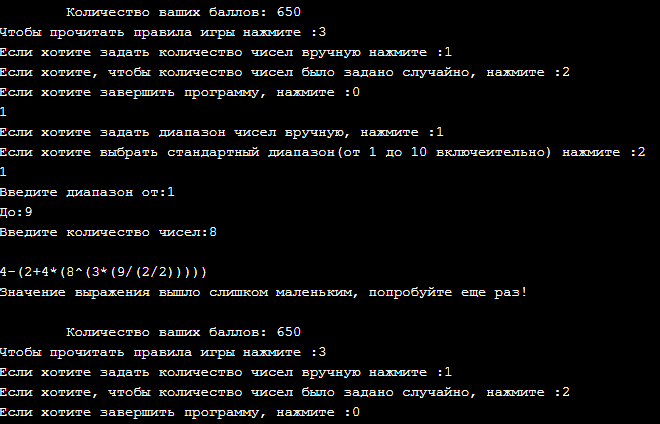
**Скриншот 2.7 (см. стр. 65)**

8) Проверка, если значение выражения получилось слишком большим с сохранением баллов



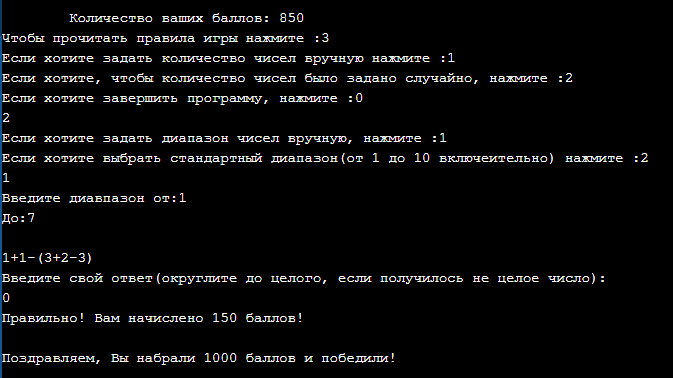
**Скриншот 2.8**

9) Проверка, если значение выражения получилось слишком маленьким с сохранением баллов



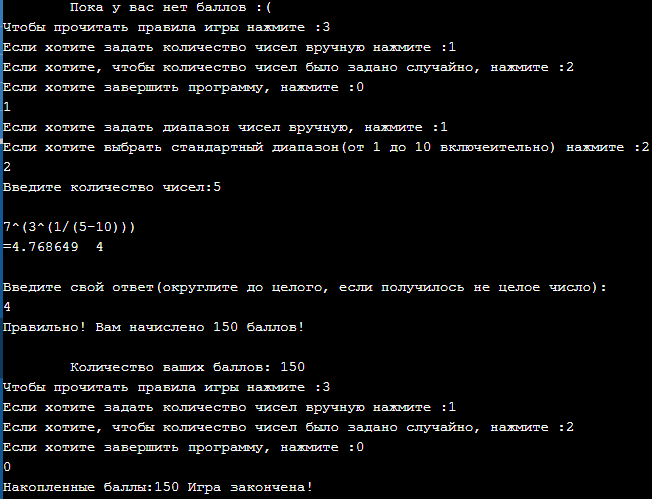
**Скриншот 2.9**

10) Проверка завершение игры по достижении 1000 баллов



**Скриншот 2.10**

11) Проверка завершения игры, если пользователь решил прекратить игру



**Скриншот 2.11**

Вывод: в ходе работы выяснилось, что большую роль в любых игровых программах играет хорошо продуманный интерфейс, предусматривающий неправильный ввод пользователем каких-либо переменных. Ведь любая неточность может привести к полной неработоспособности программы. Так же при разработке стоит учитывать, что у пользователя должна быть хоть какая-то свобода выбора, иначе игра кажется скучной и быстро надоедает. Система поощрений так же должна быть разнообразна, ведь это дает пользователю большую вовлеченность в процесс игры.

**ЗАКЛЮЧЕНИЕ**

Алгоритмы, разобранные в этой курсовой работе, показали, что существует много разных подходов, к которым прибегают программисты для создания приложений. Для разного типа задач используются разные решения, ведь для математических расчетов нужно одно, а для создания игр совсем другое. Мы разобрали малую часть тех задач, которые решают программисты с помощью языков высокого уровня. Трудно не заметить, что при написании кода используются знания из разных областей, затрагиваются даже гуманитарные науки, такие, как психология человека.

Яркая особенность языков программирования высокого уровня заключается именно в том, что для их изучения достаточно желания их изучить, они логичны и понятны. Синтаксис языка Си оказался очень удобен и прост, сложные конструкции языков программирования низкого уровня были бы в разы длиннее. Компактность, простота и надежность всегда будут отличительной чертой этого языка.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

**[1] Кочан. Стефан. Программирование на языке С, 3-е издание.: Пер. с англ. – М.:ООО “И.Д.Вильямс”, 2006. – 496 с. : ил. Парал.тит.англ.(касательных)**

**[2]Шень А. Программирование: теоремы и задачи. – 4-е изд., стереотипн. – М.: МЦНМО, 2011. – 296 с.: ил.(секущих)**

**[3] Генерация случайных чисел в языке Си**

**Режим доступа: youngcoder.ru/lessons/4/sluchainie\_chisla\_na\_c.php**

**[4]Златопольский, Д. М. Сборник задач по программированию. – 3-е изд., перераб. и доп. – СПб.: ВХВ-Петербург, 2011. – 304 с.: ил. – (ИиИКТ)**

**[5]Могилев, А.В. Методы программирования. Компьютерные вычисления /А.В. Могилев, Л.В. Листрова. – СПб.: БХВ-Петербург, 2008. – 320 с.: ил. – (ИиИКТ)**

**[6]Языки программирования высокого уровня. Режим доступа: https://programmera.ru/articls/sovremennye-yazyki-programmirovaniya-vysokogo-urovnya/**

**[7]Машинный код. Материал из национальной библиотеки им. Баумана от  20 апреля 2017 г. Режим доступа:** **https://ru.bmstu.wiki/Машинный\_код**

**[8]Смирнов, В.И. Курс высшей математики. Том I/Пред. Л.Д. Фадеева, пред. и п прим. Е.А. Грининой: 24-е изд. – СПб.:БХВ-Петербург, 2008. – 624 с.: ил. – (Учебная литература для вузов)**

**[9]Роллингз, Эндрю, Моррис, Дэйв. Проектирование и архитектура игр.: Пер. с англ. – М.: Издательский дом “Вильямс”, 2006. – 1040 с.: ил. – Парал. тит. англ.**

**[10]Рейтин стран по числу компьютеров на 1000 жителей. Режим доствупа: https://ru.wikipedia.org/wiki/Рейтинг\_стран\_по\_числу\_компьютеров\_на\_1000\_жителей\_(2004)**