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 №4

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

студент группы ИВТ-11 Дунайкин Д.М.

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

доцент, к.т.н. Артамонов Ю.Н.

Котельники – 2020

***Содержание***

|  |  |  |
| --- | --- | --- |
| 1. | Введение……………………………………………………………... | 3 |
| 2. | Часть №1 Разработка численных алгоритмов…………................... | 5 |
| 2.1 | Суммирование рядов и вычисление элементарных функций……………………………………………………….……… | 5 |
| 2.1.1 | Сумма бесконечного ряда…………………………………............... | 5 |
| 2.1.2 | Сумма цепной (бесконечной) дроби………………………...…....... | 6 |
| 2.2 | Приближенные методы нахождения корней уравнения………….. | 14 |
| 2.2.1 | Метод касательных………………………………………….....……. | 14 |
| 2.2.2 | Метод секущих………………………………………………..…..…. | 38 |
| 3. | Часть №2 Разработка игровой программы…………………............ | 62 |
| 4. | Заключение……………………………………………….……....... | 100 |
| 5. | Список литературы…………………………………………….......... | 101 |

1. ***Введение***

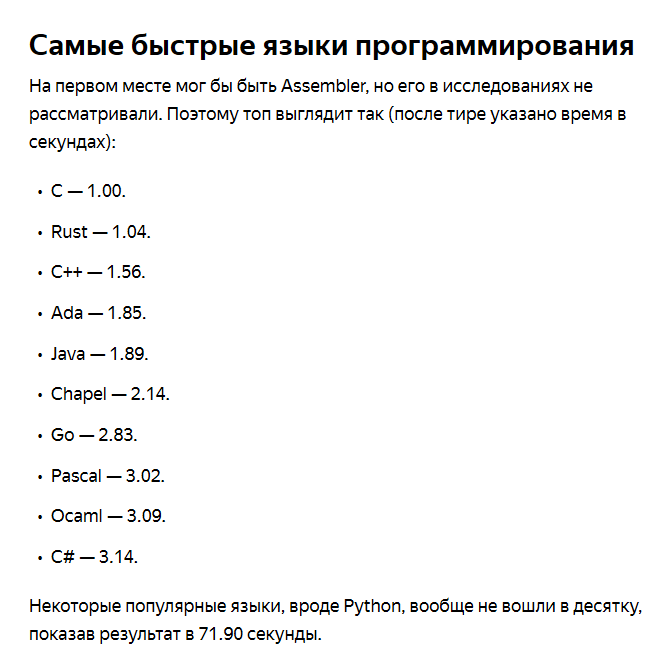
Использованные литературные источники [1,8,9].

Си является языком программирования высокого уровня. Если вы только пришли в этот чудесный мир информационных технологий и у вас есть желание овладеть профессией, которая будет востребована в 21-ом столетии (программист) уж точно и которая при должной сноровке и умении учиться будет приносить вам не только удовольствие от того, чем вы занимаетесь, но также постоянный и высокий доход, тогда начать путь, лично я рекомендую вам с изучения чудесного Си.

Начинающий пользователь спросит меня: “А почему я должен начать учить именно Си, а не какой-либо другой, более функциональный язык программирования?” Ответ с моей стороны полетит к вам незамедлительно.

После перечисления особенностей данного высокоуровневого языка у вас вряд ли останутся сомнения.

* Довольно-таки простая степень изучения, даже если вы полный “чайник” в программировании. Что я имею в виду под “простотой” изучения (без углубленного изучения)? Изучение базовой платформы языка не вызовет у вас трудностей (достаточно знать и главное понимать линейное программирование в Си, ветвистое программирование в Си с использованием конструкции выбора (if/else) или же более сложная конструкция выбора (switch), циклическое программирование (циклы вида for, while, do/while), массивы (одномерные, двумерные…) (статические, динамические).
* При более углубленном изучении языка пользователю будет необходимо изучить структуры данных (статические, динамические), для новичка это будет непростой задачей.
* Достаточное количество литературы, статей, готового кода с подробным описанием для лучшего понимания языка.
* Скорость разработки программного кода + скорость компиляции вашего продукта намного выше других языков. Приведу наглядный пример своих слов:



Комментарии здесь не требуются. Данный топ был взят у ребят из SkillBox (исследование проводилось в 2017 году), в наше время скорость может быть намного выше.

* Для разработки сложных программ стоит рассмотреть структурный подход (реализовывать алгоритм по частям).

Я указал лишь самые поверхностные особенности данного языка, копните чуть глубже, сможете узнать много чего интересного и познавательного об этом языке. Чем больше опыта у вас будет, тем легче вам будет в понимании вообще всех языков программирования, несмотря на их сложность.

**Часть №1. Разработка численных алгоритмов**

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

Использованные литературные источники [2-4,7,10]

**2.1.1 Сумма бесконечного ряда**

Использованные литературный источник [11]

Перед тем как приступить к вычислению суммы бесконечного ряда, следует ввести нового неопытного пользователя в курс дела и объяснить ему, а позже и наглядно показать, что собой представляет сумма ряда. Пожалуй, можно начать с теоретической части.

**Суммой ряда** также называют **бесконечную сумму** — она является одним из центральных понятий глав математического анализа. Если объяснять простым языком “на пальцах”, то сумму ряда можно записать по формуле, приведенной ниже:

Краткая запись будет выглядеть так:

В данной записи переменная может выступать любым членом последовательности: (n-ным или общим членом) первым, вторым … , и данный член может принимать значение как вещественных чисел, так и комплексных.

Для более ясного представления того, что я сейчас вам объяснил, предлагаю взглянуть на **“частичную сумму”**, которая получается, если оборвать бесконечный ряд суммы на определённом члене:

…

…

В случае, когда последовательность данных “частичных сумм” имеет какой-то предел (конечный или бесконечный), то говорят, что сумма бесконечного ряда равна этому пределу.

Мой вариант суммы ряда имеет следующее представление:

,

где – целый член последовательности ряда, – дробный член последовательности ряда.

2). Требуется найти: вычислить значение экспоненты с помощью суммы бесконечного ряда + узнать кол-во членов ряда для достижения заданной точности.

3). Решение: листинг программного кода данного алгоритма задачи будет приложен ниже (стр. 7-10) вместе с алгоритмом расчета экспоненты с помощью цепной дроби.

**2.1.2 Сумма цепной (бесконечной дроби)**

Использованный литературный источник [12]

Стоит начать опять же с теории чтобы новый пользователь имел представление что это вообще такое.

Определение:

Цепная (бесконечная) дробь – выражение вида

**,**

где – представляет собой целое число (натуральные числа + нуль + отрицательные числа), – все остальные числа представляют собой натуральные числа (положительные числа). Также хочу обратить ваше внимание на то, что числа являются **неполными частными** или же **элементами цепной дроби**.

Абсолютно любое число вещественного типа можно представить в виде цепной дроби. Число даже можно представить в виде конечной цепной дроби, но при этом оно должно быть рациональным (то есть в виде обыкновенной дроби, где числитель будет являться целым числом, а знаменатель – натуральным).

Одним из плюсов применения цепных дробей является довольно-таки точное приближенное значение вещественных чисел в виде обыкновенных дробей. Перейдем к конкретному примеру. Мой вариант представляет собой цепную дробь, при которой пользователь сможет вычислить приближенное значение математической константы – экспоненты.

1). Дано: представление экспоненты в виде цепной дроби;

2). Найти: количество членов дроби, необходимое для достижения заданной точности.

3). Решение: прилагаю листинг программного кода всего алгоритма.

#include<stdio.h> //стандартная библиотека ввода-вывода данных.

#include<math.h> //математическая библиотека в языке С.

#include<stdlib.h> //стандартная библиотека в С, содержащая в себе функции, занимающиеся выделением памяти, контролем процесса выполнения программы, преобразованием типов и другое.

long double riad(long int); //прототип фун-ии, вычисляющей приближенное значение экспоненты с помощью суммы ряда.

long double riad(long int toch) //сама фун-ия.

{

long int l=1, t=2, n=1; // "l" отвечает за значение факториала, "t" участвует в нахождении следующего факториала, "n" кол-во членов ряда.

long double e1=2.0; //значение первого члена ряда.

printf("exp = %.20Lf | n = %ld\n",e1,n); //данный вывод на экран делаю для наиболее понятного вычисления экспоненты.

system("sleep 0.5"); //таймер сна в 0.5 секунды.

//с каждой новой итерацией цикла разность fabs(e1-exp(1)) все меньше и меньше => мы приближаемся к заданной точности, как только разность будет < заданной точности, цикл будет прерван.

while(fabs(e1-exp(1))>pow(10,-toch)) //цикл, отвечающий за вычисление экспоненты с заданной точностью.

{

l\*=t; //вычисление факториала.

e1+=1.0/l; //находим следующий член последовательности.

t++; //участвует в нахождении следующего факториала.

n++; //увеличиваем кол-во членов ряда.

printf("exp = %.20Lf | n = %ld\n",e1,n); //данный вывод на экран делаю для наиболее понятного вычисления экспоненты (чтобы пользователь видел как изменяется значение переменной "e1" при новой итерации цикла).

system("sleep 0.5"); //таймер сна в 0.5 секунды.

}

printf("Кол-во членов ряда = %ld\n",n); //вывод на экран значения переменной "n".

return e1; //возвращаем приближенное значение экспоненты.

}

long double drob(long int); //прототип фун-ии, вычисляющей приближенное значение экспоненты с помощью цепной дроби.

long double drob(long int toch) //сама фун-ия.

{

long double eps,x; //переменная "x" отвечает за дробную часть приближенного вычисления экспоненты.

long int i,n; //переменная "i" служит для условий в цикле for, "n" - отвечает за кол-во членов дроби.

eps=pow(10,-toch); //переменная определяет значение точности.

n=1; //кол-во элементов дроби.

x=n; //присваиваем переменной "x" значение переменной "n".

while(fabs(exp(1)-x)>eps) /\*с каждой новой итерацией разность будет все меньше и меньше,

когда разность станет меньше значения переменной eps, цикл закончится.\*/

{

x=n; /\*делаем присваивание, т.к после for делаем n++ => "х" должен присвоиться этому "n".

Иначе после for "х" так и будет равен n =1.\*/

for(i=n;i>1;i--) /\*с каждой новой итерацией значение i уменьшается на единицу.

Если i>0,то когда доходим до i=1, по алгоритму нужно сделать 2 + ...

Но этого уже не надо делать, мы дошли до конца и нужно х присвоить 1+(1/x)\*/

{//вычисление приближенного значения экспоненты

if(i%2 != 0) x=2+(1/x); //с помощью данной конструкции if/else контролируется знак члена дроби + или -

else x=i-1-(1/x); //все зависит от того является ли i чётным или нечётным.

}

x=1+(1/x); //присваивание переменной "x" значения 1+(1/x). => икс в знаменателе вычисляется в for.

printf("exp = %.20Lf | n = %ld\n",x,n); //печатаем с каждой новой итерацией цикла приближенное значение экспоненты, значение переменной n.

system("sleep 0.5"); //таймер сна в 0.5 секунды.

n++; //увеличиваем кол-во членов дроби.

}

printf("Кол-во членов дроби = %ld\n",n-1); /\*вывод на экран значения переменной "n".

после цикла for идет n++, то есть она увеличивается ДО того как произойдет проверка условия в цикле while,

прошли, например, 7 членов дроби, увеличили "n" на 1, потом проверяем условие, оно не проходит, но "n" уже 8.

Потому и при выводе делаем n-1;\*/

return x; //возвращаем приближенное значение экспоненты.

}

int main() //главная функция.

{

long int toch; //переменная "toch" отвечает за кол-во знаков после запятой.

printf("Введите кол-во знаков после запятой: "); //вывод сообщения на экран.

scanf("%ld",&toch); //вводим значение переменной "toch" с клавиатуры.

printf("Вычисление математической экспоненты с помощью суммы бесконечного ряда:\n"); //вывод сообщения на экран.

//Вызов фун-ии, вычисляющей экспоненту с помощью РЯДА, вывод ее результата + вывод значения переменной "toch".

printf("Приближенное значение экспоненты равно: %.20Lf\nТочность вычислений %ld знак(а)(ов) после запятой\n",riad(toch),toch);

printf("----------------------------------------------\n"); //перенос чтобы различать начало и конец методов решений.

printf("Вычисление математической экспоненты с помощью бесконечной (цепной) дроби:\n"); //вывод сообщения на экран.

//Вызов фун-ии, вычисляющей экспоненту с помощью ЦЕПНОЙ ДРОБИ, вывод ее результата + вывод значения переменной "toch".

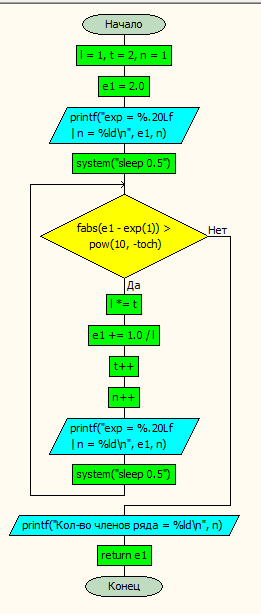
printf("Приближенное значение экспоненты равно: %.20Lf\nТочность вычислений %ld знак(а)(ов) после запятой\n",drob(toch),toch);

return 0; //возвращаем нуль.

}

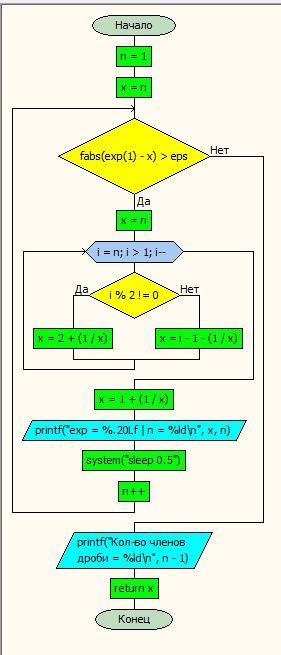
Блок-схемы всех трех функций.

1). Блок-схема функции riad.



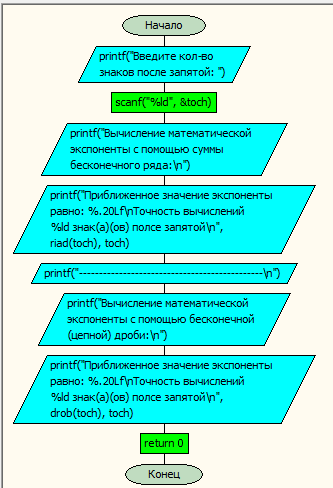
|  |  |
| --- | --- |
| Переменная | Изначальные значения |
| l | 1 |
| t | 2 |
| n | 1 |
| E1 | 2.0 |

2). Блок-схема функции drob.



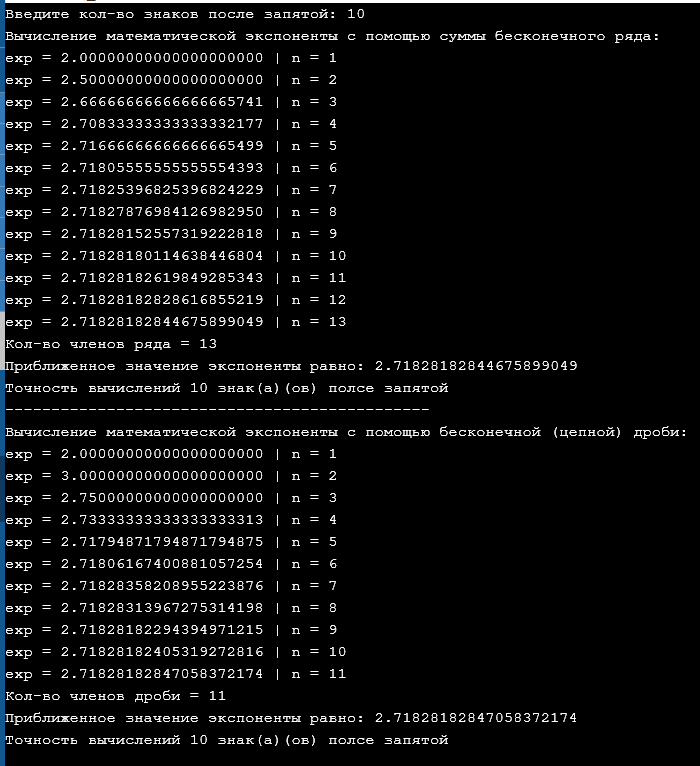
|  |  |
| --- | --- |
| Переменная | Изначальное значение |
| n | 1 |
| x | n |
| i | n |

3). Блок-схема главной функции (main).



|  |  |  |
| --- | --- | --- |
| Точность (кол-во знаков после запятой) | Кол-во членов РЯДА | Кол-во членов ДРОБИ |
| Toch=5 | N=8 | N=7 |
| Toch=8 | N=11 | N=9 |
| Toch=11 | N=14 | N=12 |
| Toch=13 | N=15 | N=13 |
| Toch=15 | N=17 | N=15 |
| Toch=3 | N=6 | N=5 |
| Toch=4 | N=7 | N=7 |

Также прилагаю скриншот работоспособности данной программы.



Вывод: цепной дроби необходимо меньше членов по сравнению с рядом, при одной и той же точности вычислений (редко, когда может произойти ситуация, что количество членов будет одним и тем же в обоих случаях при одной и той же точности). Из этого можно сделать логичный вывод, что дробь быстрее сходится, нежели ряд, к тому же дробь очень часто дает более точное значение математической константы – экспоненты на конце.

**2.2 Приближённые методы нахождения корней уравнения**

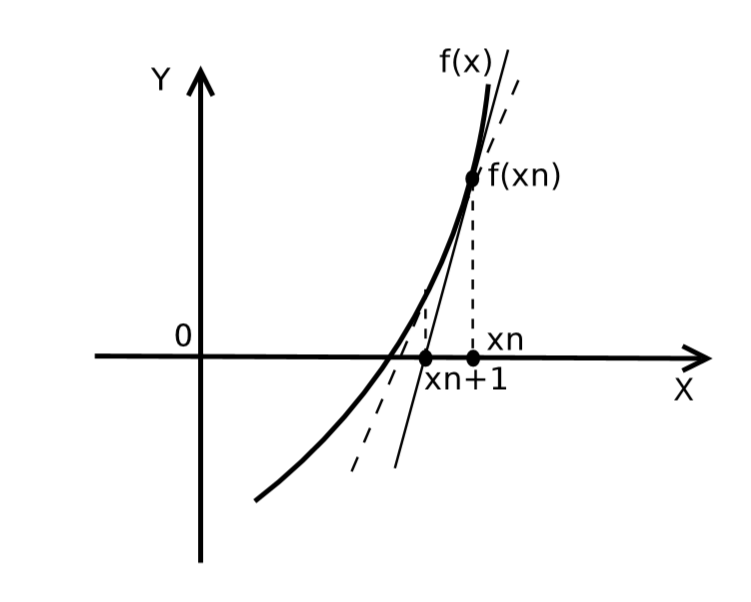
Использованный литературный источник [6]

Теоретическая часть. Бывают такие ситуации, когда дифференциальное уравнение нельзя свести к квадратному или это можно сделать, но время, потраченное на это действие будет нецелесообразным. Именно поэтому существует множество способов решения дифференциальных уравнений путем нахождения приближенных к действительности корней. Но опять же, пользователю необходимо точно быть уверенным насчет того, что уравнение имеет корень потому как, если корней не будет, можно очень долго сидеть и разбирать уравнение, а должного результата не будет.

Давайте перейдем от моих рассуждений к конкретике. Существует три метода решения дифференциальных уравнений путем нахождения приближенных корней. В данной курсовой работе представлены такие методы: метод касательных (он же метод Ньютона) + “модификация” метода касательных – метод секущих (метод хорд).

**2.2.1 Метод касательных**

Что из себя вообще представляет этот метод и в чем его суть? Сейчас я вам все объясню. Стоит начать с того, кто этот метод вообще придумал. Из названия становится сразу ясно, кто этот человек – метод был предложен Исааком Ньютоном – английским физиком- математиком. Кратко о сути решения этим методом – пользователь должен строить последовательные приближения, которые с каждой новой итерацией будут приближаться к истинному корню заданной функции.



**Рис. 2.2.1.1**

Объяснение рисунка.

1. Пользователь выбирает произвольную точку .
2. Проводит касательную к функции в этой точке.
3. Касательная пересечет ось (ось абсцисс) в точке .
4. Теперь пользователю необходимо провести касательную к функции в этой точке.

Данные действия пользователю необходимо проделывать, пока он не достигнет заданной точности, как только это произойдёт, приближенный корень конкретной функции будет найден и задача будет выполнена.

1). Дано:

2). Найти: необходимо вычислить приближенное значение корня у каждой функции, сравнить результат программы с результатом “Wolframalpha”, сравнить количество итераций данного метода с методом секущих при одной и той же точности вычислений.

3). Решение: прилагаю листинг программного кода данного алгоритма.

#include<stdio.h> //стандартная библиотека ввода-вывода данных.

#include<math.h> //математическая библиотека в языке С.

#include<stdlib.h> //стандартная библотека в С, содержащая в себе функции, занимающиеся выделением памяти, контролем процесса выполнения программы, преобразованием типов и другое.

#include<time.h> //заголовочный файл стандартной библиотеки языка программирования C, содержащий типы и функции для работы с датой и временем.

#include<stdbool.h> //стандартная библиотека языка С, поддерживающая работу с типом данных \_Bool.

float zadau\_function(float,float,float,int); //прототип фун-ии, отвечающей за значение фун-ии f(x) в точке xn.

float zadau\_function(float c,float d,float xn,int nf) //реализация этой самой фун-ии.

{

float fx; //переменная fx - отвечает за хранение значения изначальной функции.

if(nf==1) fx=sin(c\*xn)-d; //если выбрана 1-ая фун-ия.

if(nf==2) fx=pow(exp(1),c\*xn)-d; //если выбрана 2-ая фун-ия.

if(nf==3) fx=pow(xn,3)+c\*xn-d; //если выбрана 2-ая фун-ия.

if(nf==4) fx=log2(c\*xn)-d; //если выбрана 4-ая фун-ия.

if(nf==5) fx=sqrt(c\*xn)-d; //если выбрана 5-ая фун-ия.

if(nf==6) fx=(c\*pow(xn,2)-d)/(xn+d); //если выбрана 6-ая фун-ия.

return fx; //возвращаем значение фун-ии в точке x\_n.

}

float zadau\_proisvodnuy(float,float,float,int); //прототип фун-ии, отвечающей за значение производной фун-ии f(x) в точке xn.

float zadau\_proisvodnuy(float c,float d,float xn,int nf) //реализация этой самой фун-ии.

{

float px; //переменная px - отвечает за хранение производной изначальной функции.

if(nf==1) px=c\*cos(c\*xn); //если выбрана 1-ая фун-ия.

if(nf==2) px=c\*pow(exp(1),c\*xn); //если выбрана 2-ая фун-ия.

if(nf==3) px=3\*pow(xn,2)+c; //если выбрана 3-ая фун-ия.

if(nf==4) px=c/(c\*xn\*log(2)); //если выбрана 4-ая фун-ия.

if(nf==5) px=c/(2\*sqrt(c\*xn)); //если выбрана 5-ая фун-ия.

if(nf==6) px=(2\*c\*xn\*(xn+d)-(c\*pow(xn,2)-d))/(pow(xn+d,2)); //если выбрана 6-ая фун-ия.

return px; //возвращаем значение производной в точке x\_n.

}

int znaki(float,float,float,float,int,int); //прототип фун-ии на проверку знаков на концах отрезка.

int znaki(float c,float d,float xn,float xn\_1,int toch,int nf) //реализации этой самой фун-ии.

{

int leftside,rightside; //переменные отвечают за хранение значений фун-ии на концах отрезка.

leftside=zadau\_function(c,d,xn\_1-pow(10,-toch),nf); /\*узнаем какой знак фун-ия имеет на левом конце(плюс или минус).

То есть вычисляем значение функции в точке xn\_1-pow(10, -toch) (см. методичку).\*/

rightside=zadau\_function(c,d,xn\_1+pow(10,-toch),nf); /\*узнаем какой знак фун-ия имеет на правом конце(плюс или минус).

То есть вычисляем значение функции в точке xn\_1+pow(10, -toch) (см. методичку).\*/

if((leftside>0 && rightside>0)||(leftside<0 && rightside<0)) //если знаки равны на концах отрезков, возвращаем 1-правда, иначе возвращаем 0-ложь.

return 1;

else

return 0;

}

float poisk\_korni(float,float,float,int,int); //прототип фун-ии, отвечающий за поиск самого корня фун-ии f(x).

float poisk\_korni(float c,float d,float xn,int toch,int nf) //реализация этой самой функции.

{

int i=0; // i - переменная отвечает за итерации.

float xn\_1; //точка, в которой касательная пересекает ось абцисс.

printf("------------------------------------------------------------------------------------------------------------\n"); //вывод сообщения на экран.

while(true) //цикл работает пока все что внутри правда, цикл остановится как только условие про итерации будет нарушено, то есть итераций будет больше 200-т.

{

if(i>200) //условие на проверку кол-ва итераций.

{

printf("Кол-во итераций превышено, ошибка (-1)\n"); //вывод сообщения на экран.

break; //прерывание цикла, если условие нарушено.

}

xn\_1=xn-(zadau\_function(c,d,xn,nf)/zadau\_proisvodnuy(c,d,xn,nf)); //вычисление точки, в которой касательная к фун-ии f(x) пересекает ось абсцисс.

//для наглядного представления работы этого метода делаю вывод после каждой итерации того, как пользователь приближается к корню.

printf("xn = %-15f | xn\_1 = %-15f | fabs(xn-xn\_1) = %-15f | i = %d\n",xn,xn\_1,fabs(xn-xn\_1),i); //печатаем изначальное значение точки xn, точки xn\_1, разность по модулю между этими точками (чем она меньше, тем ближе мы подбираемся к значению точности), кол-во итераций цикла.

if(fabs(xn-xn\_1)<pow(10,-toch)) break; //с каждой новой касательной разность между xn и xn\_1 становится все меньше и меньше, когда эта разность станет меньше точности, программа закончит проверку на точность и цикл будет прерван.

xn=xn\_1; //теперь точка xn\_1 является начальной точкой и при следующей итерации цикла касательная к фун-ии f(x) будет проходить в этой точке (см. методичку).

i++; //увеличение кол-во итераций на единицу.

}

printf("------------------------------------------------------------------------------------------------------------\n"); //вывод сообщения на экран.

if(znaki(c,d,xn,xn\_1,toch,nf) && i<200) //если функция znaki возвращает 1 и кол-во итераций меньше 200, то следует продолжить вычисление приближений, взяв меньшую точность, то есть + один знак после запятой.

{

printf("Корень из f(x) : %f ",poisk\_korni(c,d,xn,toch++,nf)); //вызов результата фун-ии поиска корня.

}

else //если кол-во итераций превышено и знаки разные, возвращаем последнее значение корня.

return xn\_1;

}

int main() //главная фун-ия.

{

int nf,toch; //переменная "nf" отвечает за номер выбранной фун-ии, "toch" за кол-во знаков после запятой.

float c,d,xn; //переменная "c" - вводим сами, "d" - вводим сами, "xn" - мы ищем значение фун-ии f(x) в этой точке + значение производной в этой же точке.

printf("Введите номер функции: 1 , 2 , 3 , 4 , 5 или 6: "); //вывод сообщения на экран.

scanf("%d",&nf); //вводим с клавиатуры значение переменной "nf".

while(nf>6 || nf<1) //проверка, правильно ли введен номер фун-ии.

{

printf("Попробуйте еще раз: "); //вывод сообщения на экран.

scanf("%d",&nf); //вводим значение переменной "nf" повторно.

}

printf("Введите значение параметра c: "); //вывод сообщения на экран.

scanf("%f",&c); //вводим с клавиатуры значение переменной "c".

if(nf==4)

{ //если выбрали фун-ию с логарифмом.

while(c<=0)

{ //аргумент логарифма должен быть больше нуля.

printf("Попробуйте ввести еще раз , коэффициент 'c' должен быть > нуля\n"); //вывод сообщения на экран.

scanf("%f",&c); //повторно вводим значение переменной "c".

}

}

if(nf==5)

{ //если выбрали фун-ию с корнем.

while(c<0)

{ //аргумент корня должен быть >= 0.

printf("Попробуйте ввести еще раз , коэффициент 'c' должен быть >= 0\n"); //вывод сообщения на экран.

scanf("%f",&c); //повторно вводим значение переменной "c".

}

}

printf("Введите значение параметра d: "); //вывод сообщения на экран.

scanf("%f",&d); //вводим с клавиатуры значение переменной "d".

printf("Введите начальное значение икса: "); //вывод сообщения на экран.

scanf("%f",&xn); //вводим с клавиатуры значение переменной "xn".

if(nf==4)

{ //если выбрали фун-ию с логарифмом.

while(xn<=0)

{ //аргумент логарифма должен быть больше нуля.

printf("Попробуйте ввести еще раз , коэффициент 'xn' должен быть > нуля\n"); //вывод сообщения на экран.

scanf("%f",&xn); //повторно вводим значение переменной "xn".

}

}

if(nf==5)

{ //если выбрали фун-ию с корнем.

while(xn<0)

{ //аргумент корня должен быть >= 0.

printf("Попробуйте ввести еще раз , коэффициент 'xn' должен быть >= 0\n"); //вывод сообщения на экран.

scanf("%f",&xn); //повторно вводим значение переменной "xn".

}

}

if(nf==6)

{ //если выбрана фун-ия с дробью.

while(d+xn==0) //проверка на то, что знаменатель равен нулю.

{

printf("Происходит деление на нуль! Введите другое значение переменной 'd'\n"); //вывод сообщения на экран.

scanf("%f",&d); //вводим значение переменной "d" повторно (можно и не только "d" повторно ввести, но и "xn", или ввести только "xn" повторно, главное чтобы сумма этих двух переменных была больше или меньше нуля).

}

}

printf("Введите точность расчетов (Кол-во знаков после запятой): "); //вывод сообщения на экран.

scanf("%d",&toch); //вводим с клавиатуры значение переменной "toch".

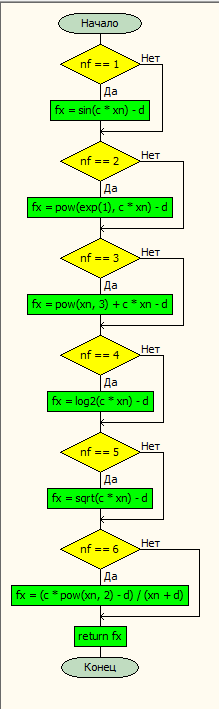
printf("Корень из f(x) : %f ",poisk\_korni(c,d,xn,toch,nf)); //выводим значение корня фун-ии.

return 0; //возвращаем нуль => конец главной фун-ии (программы).

}

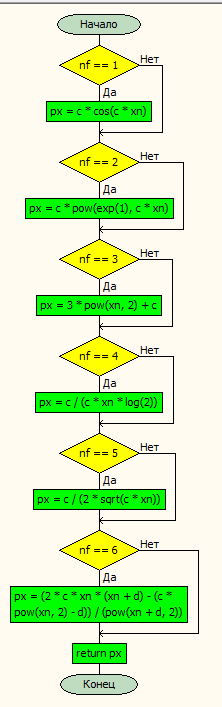
Блок-схемы данного алгоритма:

1. (Блок-схема функции zadau\_function);



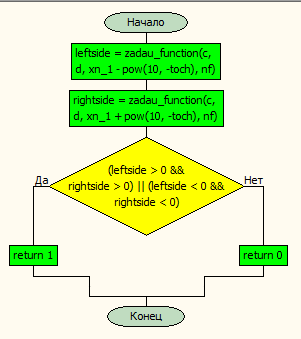
В данной функции пользователь просто находит значение функции в точке , значение которой вводит самостоятельно с клавиатуры.

1. (Блок схема функции zadau\_proisvodnuy);



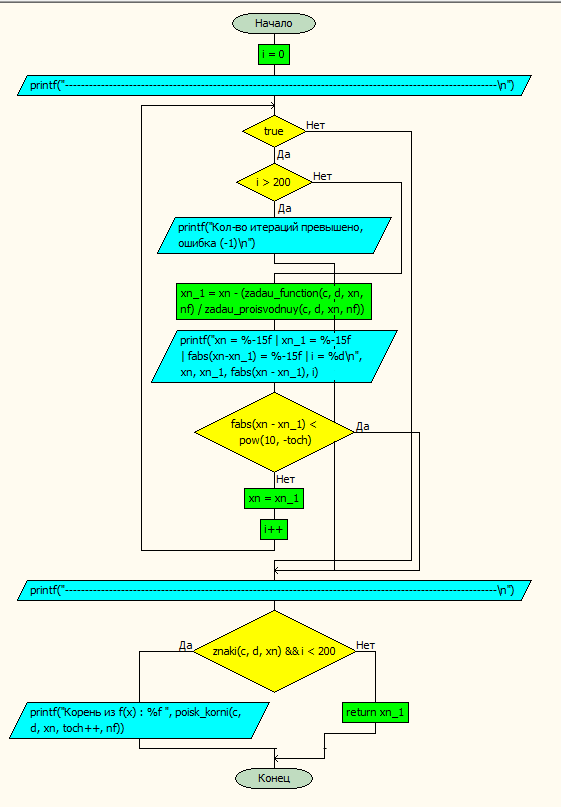
В данной функции пользователь находит значение производной функции в точке , значение которой вводит опять же самостоятельно с клавиатуры.

1. (Блок-схема функции znaki);



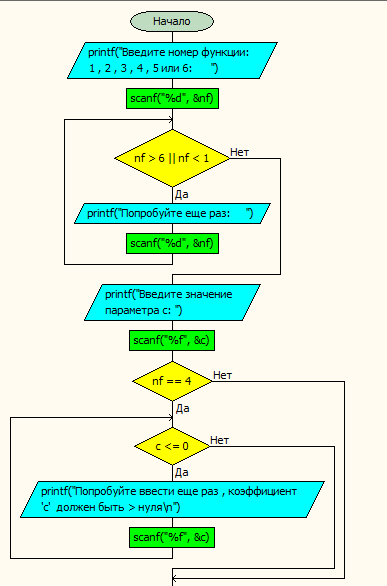
В данной функции пользователь должен узнать, какие знаки функция имеет на концах своего промежутка (на левом и правом, плюс или минус). Если условие будет “истина”, функция возвращает единицу, если условие “ложь”, функция возвращает нуль. Полученный результат будет использован следующей в блок-схеме, где пользователь собственно и будет искать корень функции .

1. (Блок-схема функции poisk\_korni);

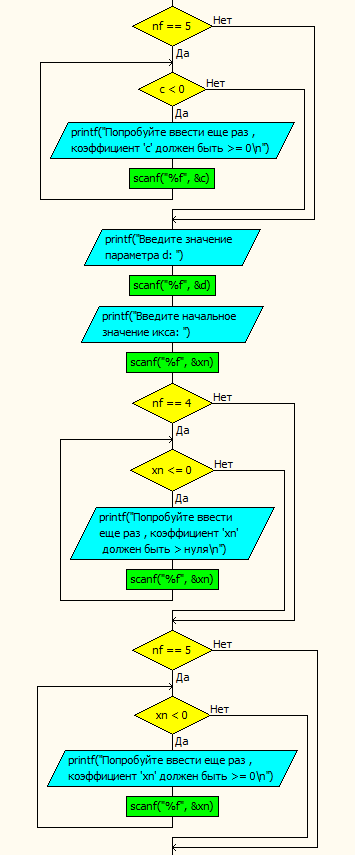


В данной функции пользователь находит приближенный корень функции . Изначально счетчик итераций равен нулю. При каждой новой итерации цикла сначала будет проверяться условие на итерации, если количество этих самых итераций превысит допустимый предел, на экран будет выведено соответствующее сообщение об ошибке и произойдет выход из цикла. Но, пока это условие не будет нарушено, пользователь идет по циклу. Когда мы все-таки выйдем из цикла, пойдет еще одна проверка, вот здесь-то и пригодится результат функции znaki, если эта функция вернула нам единицу, то пользователю необходимо продолжить вычисление корня, при этом взяв меньшую точность вычислений, в ином случае, если функция znaki вернула нуль, функция poisk\_korni возвращает последний известный корень.

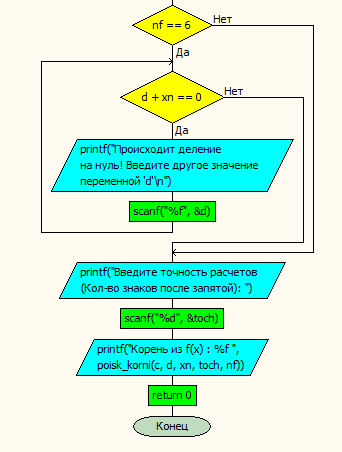
1. (Блок-схема главной функции main).



Продолжение ниже…



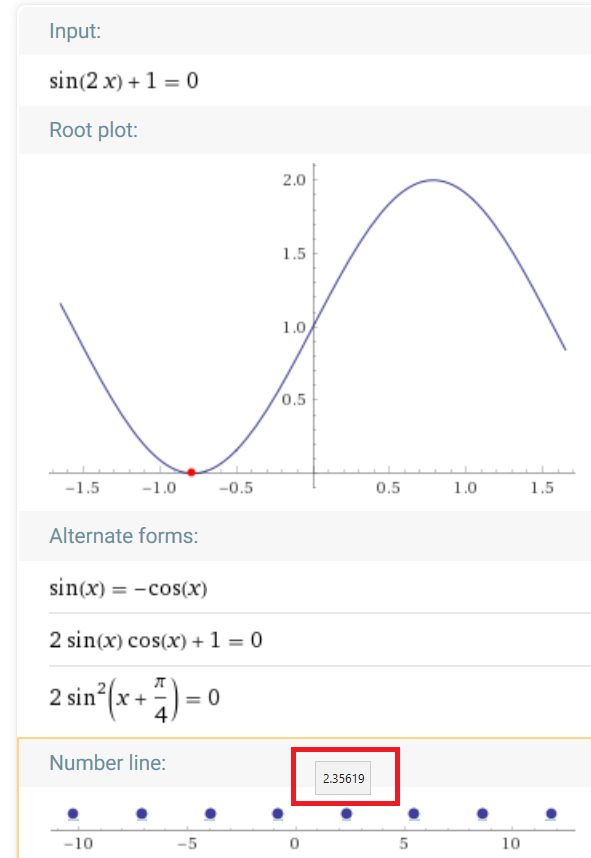
Продолжение ниже…

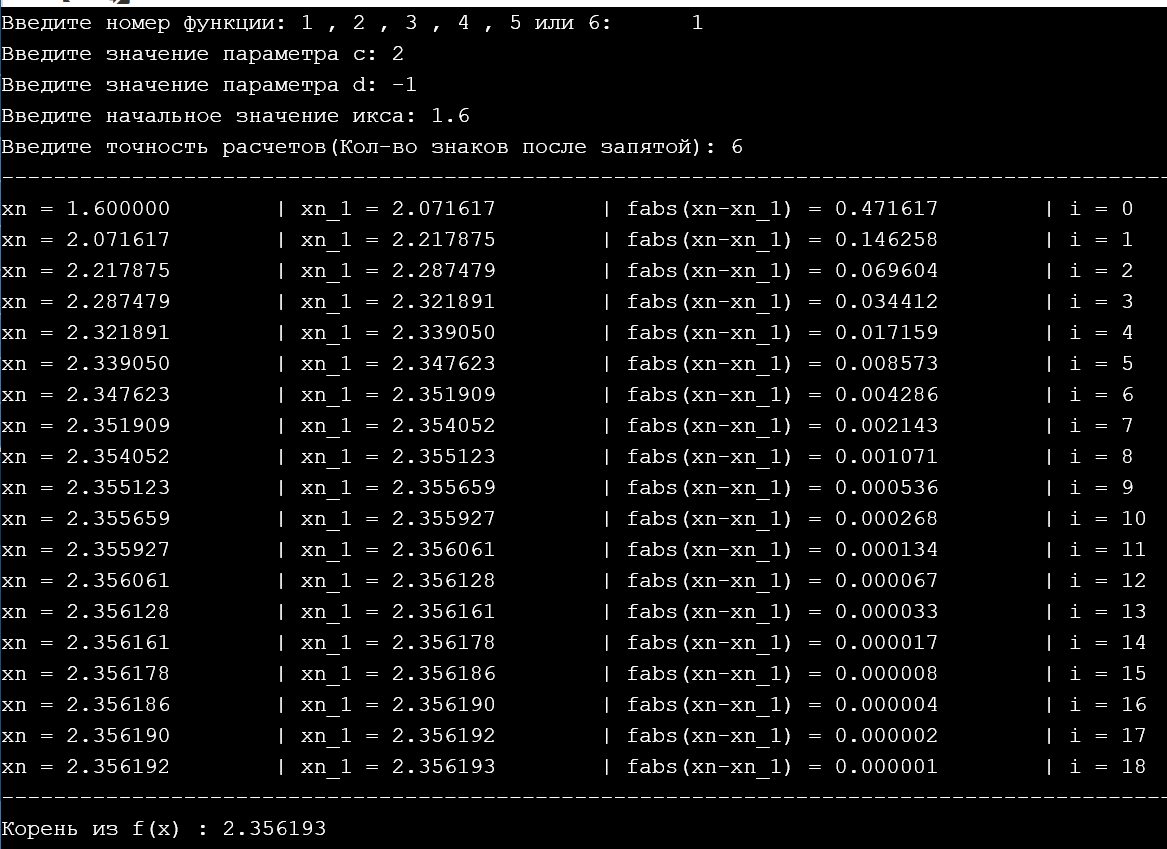


В данной функции пользователь выбирает функцию, корень которой ему нужен. Дальше идут проверки на правильность введенных значений переменных.

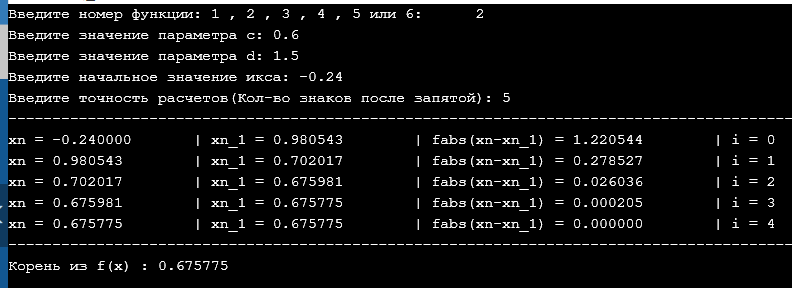
Проверка работоспособности программы:

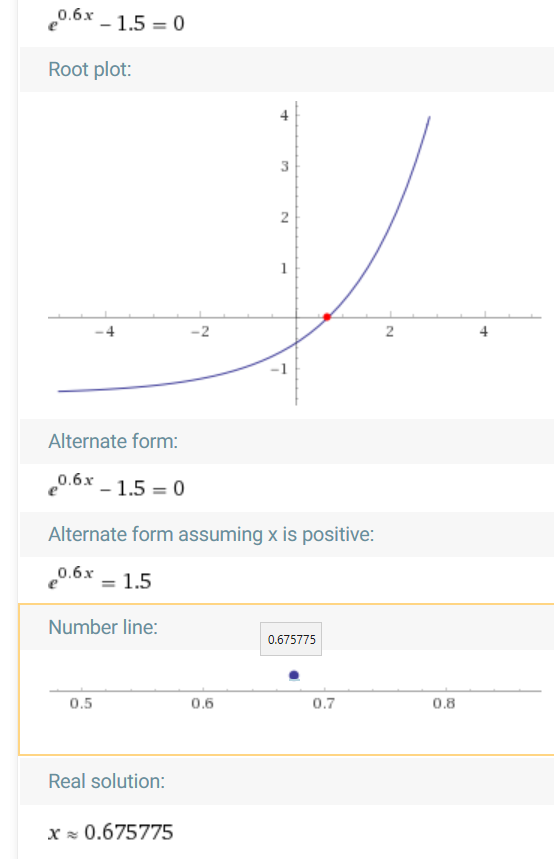
1). Функция с синусом.



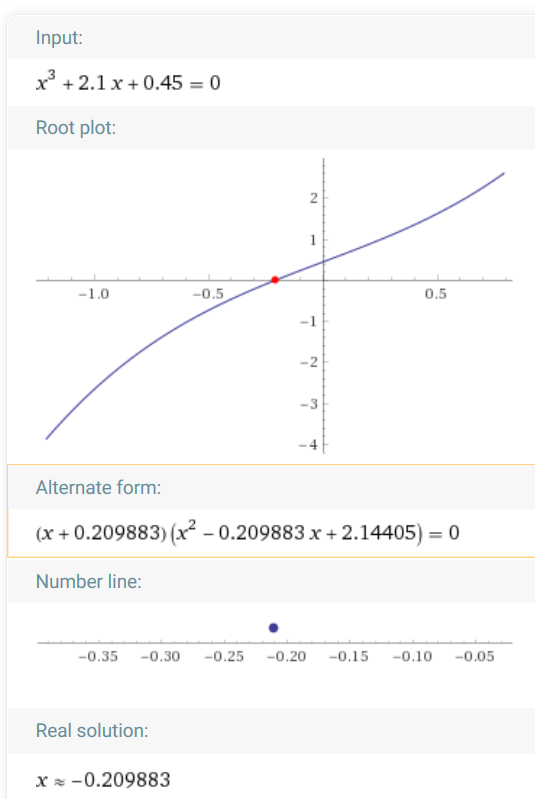


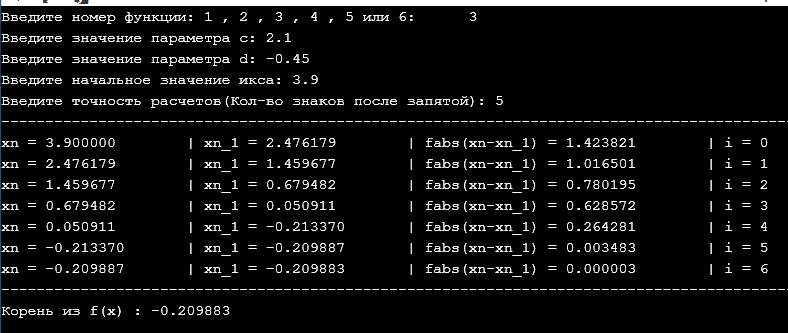
2). Функция с экспонентой.



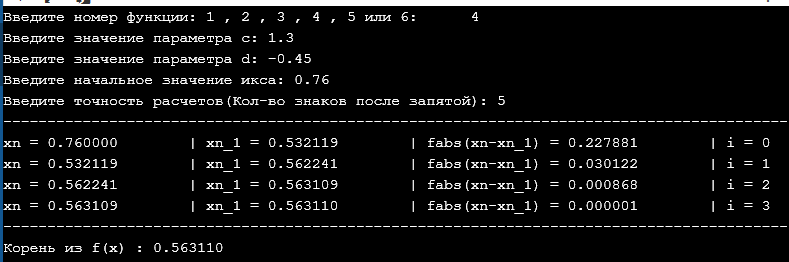


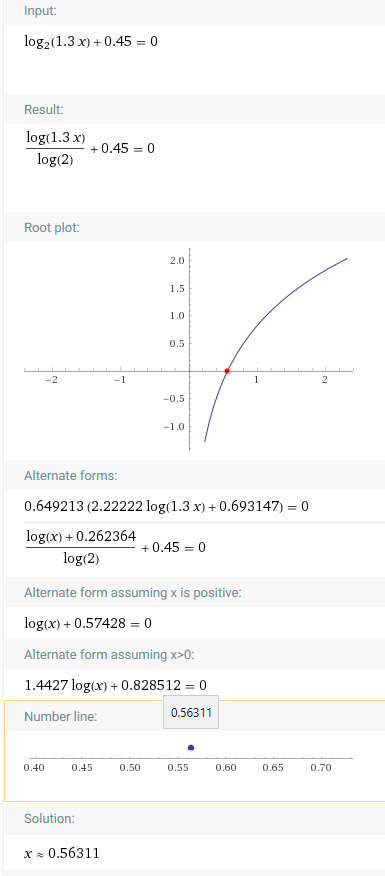
3). Функция с кубом.



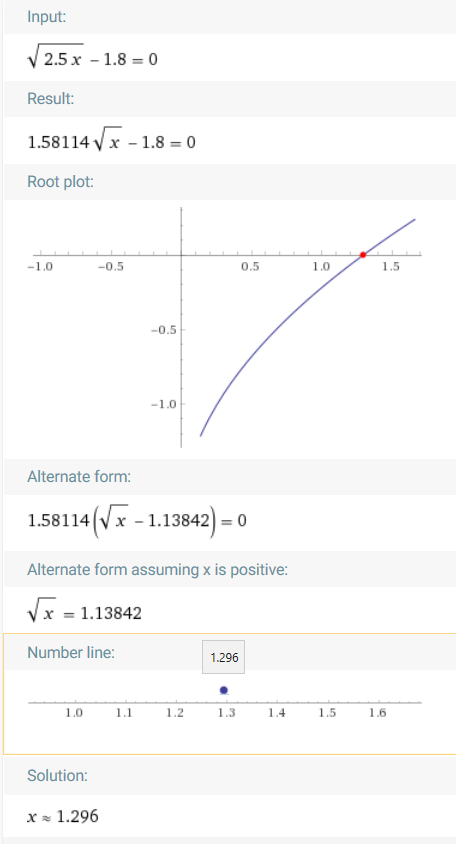


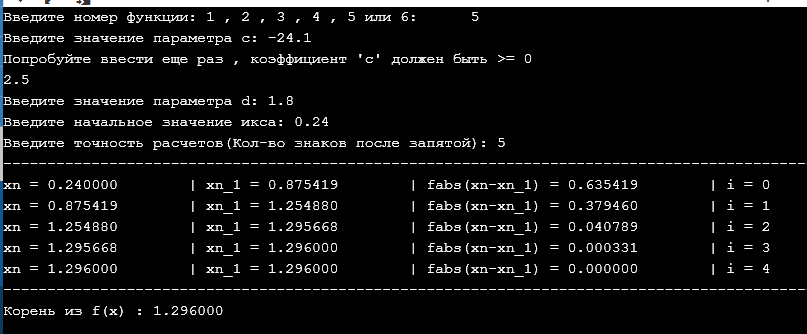
4). Функция с логарифмом.



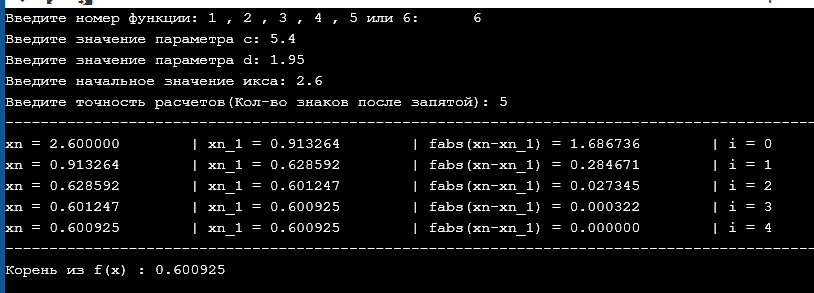


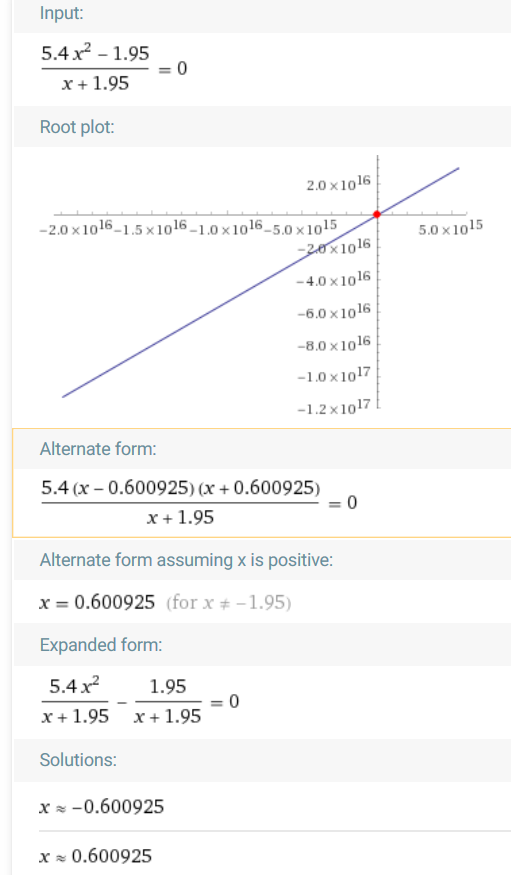
5). Функция с корнем.





6). Функция с дробью.



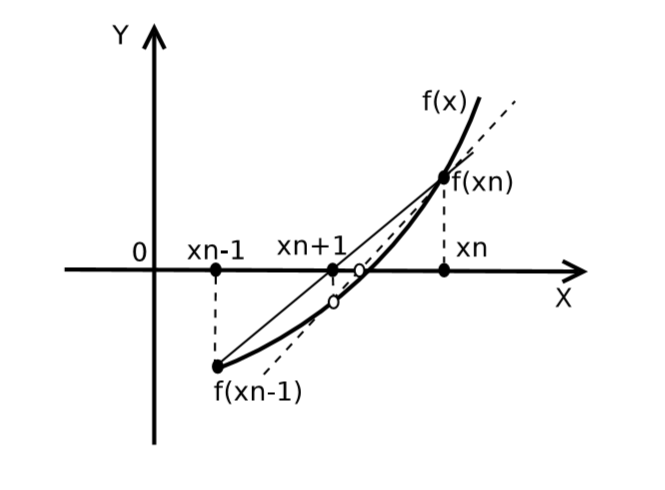


Вывод по данному методу: точность вычислений достаточно высока. Замечен интересный факт, что самая первая функция синуса – тригонометрическая, для нахождения ее корня данному методу необходимо итераций больше, нежели для всех остальных функций.

**2.2.2 Метод секущих (метод хорд)**

Как обычно, пройдемся сначала по теоретической части, дальше на конкретных примерах посмотрим, что этот метод из себя представляет.

Данный способ решения уравнений называют модификацией от метода касательных. Он также основывается на итерациях, с каждой новой итерацией пользователь приближается все ближе и ближе к корню. Отличие в том, что пользователь берет не производную конкретной функции, а ее приближения в виде конечных разностей.



**Рис. 2.2.2.1**

Объяснение рисунка.

1. Пользователь задает две разные точки и , они будут являться приближениями к истинному корню.
2. Далее пользователю необходимо найти значение конкретной функции в каждой из этих точек.
3. Теперь пользователь должен провести прямую через точки с координатами и .
4. Данная прямая пересечет ось – абсцисс в точке .
5. Теперь данная точка будет являться следующим приближением.

Данные действия пользователю необходимо проделывать, пока он не достигнет заданной точности, как только это произойдёт приближенный корень конкретной функции будет найден и ваша задача будет выполнена.

1). Дано:

2). Найти: приближенный корень каждой функции, сравнить полученный результат с результатом “Wolframalpha”, сравнить кол-во итераций этого метода с методом касательных при одной и той же точности;

3). Решение: прилагаю листинг программного кода данного алгоритма.

#include<stdio.h> //стандартная библиотека ввода-вывода данных.

#include<math.h> //математическая библиотека в языке С.

#include<stdlib.h> //стандартная библотека в С, содержащая в себе функции, занимающиеся выделением памяти, контролем процесса выполнения программы, преобразованием типов и другое.

#include<time.h> //заголовочный файл стандартной библиотеки языка программирования C, содержащий типы и функции для работы с датой и временем.

#include<stdbool.h> //стандартная библиотека языка С, поддерживающая работу с типом данных \_Bool.

float zadau\_function\_xn(float,float,float,int); //прототип фун-ии, отвечающей за значение фун-ии f(x) в точке xn.

float zadau\_function\_xn(float c,float d,float xn,int nf) //реализация этой самой фун-ии.

{

float fx; //переменная fx - отвечает за хранение значения изначальной функции в точке xn.

if(nf==1) fx=sin(c\*xn)-d; //если выбрана 1-ая фун-ия.

if(nf==2) fx=pow(exp(1),c\*xn)-d; //если выбрана 2-ая фун-ия.

if(nf==3) fx=pow(xn,3)+c\*xn-d; //если выбрана 2-ая фун-ия.

if(nf==4) fx=log2(c\*xn)-d; //если выбрана 4-ая фун-ия.

if(nf==5) fx=sqrt(c\*xn)-d; //если выбрана 5-ая фун-ия.

if(nf==6) fx=(c\*pow(xn,2)-d)/(xn+d); //если выбрана 6-ая фун-ия.

return fx; //возвращаем значение фун-ии в точке xn.

}

float zadau\_function\_xn1(float,float,float,int); //прототип фун-ии, отвечающей за значение фун-ии f(x) в точке xn1.

float zadau\_function\_xn1(float c,float d,float xn1,int nf) //реализация этой самой фун-ии.

{

float fx1; //переменная fx1 - отвечает за хранение значения изначальной функции в точке xn1.

if(nf==1) fx1=sin(c\*xn1)-d; //если выбрана 1-ая фун-ия.

if(nf==2) fx1=pow(exp(1),c\*xn1)-d; //если выбрана 2-ая фун-ия.

if(nf==3) fx1=pow(xn1,3)+c\*xn1-d; //если выбрана 2-ая фун-ия.

if(nf==4) fx1=log2(c\*xn1)-d; //если выбрана 4-ая фун-ия.

if(nf==5) fx1=sqrt(c\*xn1)-d; //если выбрана 5-ая фун-ия.

if(nf==6) fx1=(c\*pow(xn1,2)-d)/(xn1+d); //если выбрана 6-ая фун-ия.

return fx1; //возвращаем значение фун-ии в точке xn1.

}

float poisk\_korni(float,float,float,float,int,int); //прототип фун-ии, отвечающей за поиск корня f(x).

float poisk\_korni(float c,float d,float xn,float xn1,int toch, int nf) //реализация этой самой фун-ии.

{

int i=0; //i - переменная отвечает за итерации.

float xn\_1; //точка, в которой прямая, проведенная через точки с координатами (xn1,f(xn1)) и (xn,f(xn)) пересечет ось абцисс (Ox);

printf("-----------------------------------------------------------\n"); //перенос строки.

while(true) //цикл работает пока все что внутри правда, цикл остановится как только условие про итерации будет нарушено, то есть итераций будет больше 200-т.

{

if(i>200) //в методичке про это ничего не сказано для данного метода (только для касательных), но как по мне, 200-т итераций вполне должно хватить для поиска приближённого корня.

{

printf("Ну коли 200-т итераций не хватило, мои полномочия усё :)\n"); //печать сообщения на экран.

break; //если всё-таки условие про 200 итераций нарушится, цикл прервется и фун-ия вернёт приближённый корень, полученный при последней итерации.

}

xn\_1 = xn - zadau\_function\_xn(c,d,xn,nf)\*((xn - xn1)/(zadau\_function\_xn(c,d,xn,nf) - zadau\_function\_xn1(c,d,xn1,nf))); //вычисление приближённого корня по формуле (см. методичку).

printf("xn = %-15f | xn1= %-15f | xn\_1 = %-15f | i = %d\n",xn,xn1,xn\_1,i); //данное действие я делаю для наглядного видения того, как идёт приближённое вычисление корня.

if(fabs(zadau\_function\_xn1(c,d,xn1,nf))<pow(10, - toch)) break; //Поскольку xn\_1 все ближе к корню, значит значение функции в этой точке все ближе к 0. т.е. рано или поздно значение функции должно стать меньше заданной точности.

xn1=xn\_1; //Изначально мы сами задаем диапазон: от xn до xn1 (предполагаем, что значения функции в них имеют разные знаки). Затем мы нашли xn\_1. Эта точка находится между xn и xn1. Теперь нам надо снова взять диапазон, но уже меньший, т.е. xn1 должна сместиться ближе к xn и, соответственно, к корню функции.

i++; //увеличиваю кол-во итераций на единицу.

}

printf("-----------------------------------------------------------\n"); //перенос строки.

return xn\_1; //возвращение приближённого корня, полученного при самой последней итерации цикла.

}

int main() //главная фун-ия.

{

int nf,toch; //переменная "nf" отвечает за номер выбранной фун-ии, "toch" за кол-во знаков после запятой.

float c,d,xn,xn1; //переменная "c" - константа, "d" - константа, "xn" в этой точке вычисляется значение конкретной фун-ии, "xn1" в этой точке вычисляется значение конкретной фун-ии;

printf("Введите номер функции: 1 , 2 , 3 , 4 , 5 или 6: "); //вывод сообщения на экран.

scanf("%d",&nf); //вводим с клавиатуры номер фун-ии, приближённый корень для которой нам нужен.

while(nf>6 || nf<1) //проверка, правильно ли введен номер фун-ии.

{ //если введено недопустимое число, нужно ввести допустимое число повторно.

printf("Попробуйте еще раз: "); //вывод сообщения на экран.

scanf("%d",&nf); //вводим значение переменной "nf" повторно.

}

printf("Введите значение параметра c: "); //вывод сообщения на экран.

scanf("%f",&c); //вводим значение переменной "c".

if(nf==4)

{ //если выбрали фун-ию с логарифмом.

while(c<=0) //доп. проверки.

{ //аргумент у логарифма всегда должен быть положительным.

printf("Попробуйте ввести еще раз , коэффициент 'c' должен быть > нуля\n"); //вывод сообщения на экран.

scanf("%f",&c); //вводим значение переменной "c" повторно.

}

}

if(nf==5)

{ //если выбрали фун-ию с корнем.

while(c<0)

{ //аргумент корня должен быть >= 0.

printf("Попробуйте ввести еще раз , коэффициент 'c' должен быть >= 0\n"); //вывод сообщения на экран.

scanf("%f",&c); //повторно вводим значение переменной "c".

}

}

printf("Введите значение параметра d: "); //вывод сообщения на экран.

scanf("%f",&d); //вводим с клавиатуры значение переменной "d".

printf("Введите значение xn: "); //вывод сообщения на экран.

scanf("%f",&xn); //вводим с клавиатуры значение переменной "xn".

if(nf==4)

{ //если выбрали фун-ию с логарифмом.

while(xn<=0) //доп. проверки.

{ //аргумент у логарифма всегда должен быть положительным.

printf("Попробуйте ввести еще раз , коэффициент 'xn' должен быть > нуля\n"); //вывод сообщения на экран.

scanf("%f",&xn); //вводим значение переменной "xn" повторно.

}

}

if(nf==5)

{ //если выбрали фун-ию с корнем.

while(xn<0)

{ //аргумент корня должен быть >= 0.

printf("Попробуйте ввести еще раз , коэффициент 'xn' должен быть >= 0\n"); //вывод сообщения на экран.

scanf("%f",&xn); //повторно вводим значение переменной "xn".

}

}

if(nf==6)

{ //если выбрали фун-ию с дробью.

while(d+xn==0) //проверка на то, что знаменатель равен нулю.

{

printf("Происходит деление на нуль! Введите другое значение переменной 'xn'\n"); //вывод сообщения на экран.

scanf("%f",&xn); //вводим значение переменной "xn" повторно (можно и не только "xn" повторно ввести, но и "d", или ввести только "d" повторно, главное чтобы сумма этих двух переменных была больше или меньше нуля).

}

}

printf("Введите значение xn1: "); //вывод сообщения на экран.

scanf("%f",&xn1); //вводим с клавиатуры значение переменной "xn1".

if(nf==4)

{ //если выбрали фун-ию с логарифмом.

while(xn1<=0) //доп. проверки.

{ //аргумент у логарифма всегда должен быть положительным.

printf("Попробуйте ввести еще раз , коэффициент 'xn1' должен быть > нуля\n"); //вывод сообщения на экран.

scanf("%f",&xn1); //вводим значение переменной "xn1" повторно.

}

}

if(nf==5)

{ //если выбрали фун-ию с корнем.

while(xn1<0)

{ //аргумент корня должен быть >= 0.

printf("Попробуйте ввести еще раз , коэффициент 'xn1' должен быть >= 0\n"); //вывод сообщения на экран.

scanf("%f",&xn1); //повторно вводим значение переменной "xn".

}

}

if(nf==6)

{ //если выбрали фун-ию с дробью.

while(d+xn1==0) //проверка на то, что знаменатель равен нулю или нет.

{

printf("Происходит деление на нуль! Введите другое значение переменной 'xn1'\n"); //вывод сообщения на экран.

scanf("%f",&xn1); //вводим значение переменной "xn1" повторно (можно и не только "xn1" повторно ввести, но и "d", или ввести только "d" повторно, главное чтобы сумма этих двух переменных была больше или меньше нуля).

}

}

printf("Введите точность расчетов (Кол-во знаков после запятой): "); //вывод сообщения на экран.

scanf("%d",&toch); //вводим с клавиатуры значение переменной "toch".

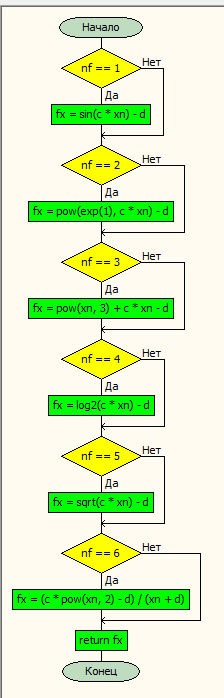
printf("Корень из f(x) : %f ",poisk\_korni(c,d,xn,xn1,toch,nf)); //выводим значение корня фун-ии.

return 0; //возвращаем нуль => конец главной фун-ии (программы).

}

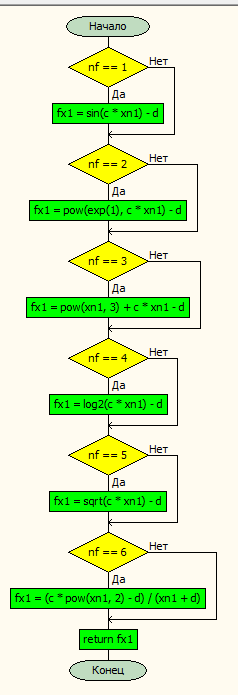
Также предлагаю вам взглянуть на блок-схемы данной программы.

1. (Блок-схема функции zadau\_function\_xn);



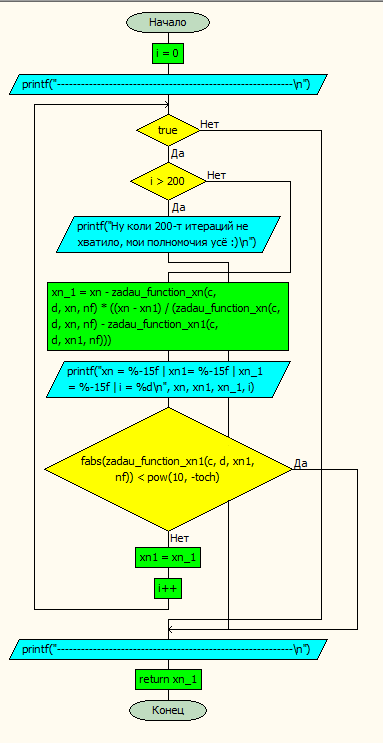
В данной функции высчитывается значение выбранной функции в точке , значение которой пользователь вводит самостоятельно с клавиатуры.

1. (Блок-схема функции zadau\_function\_xn1);



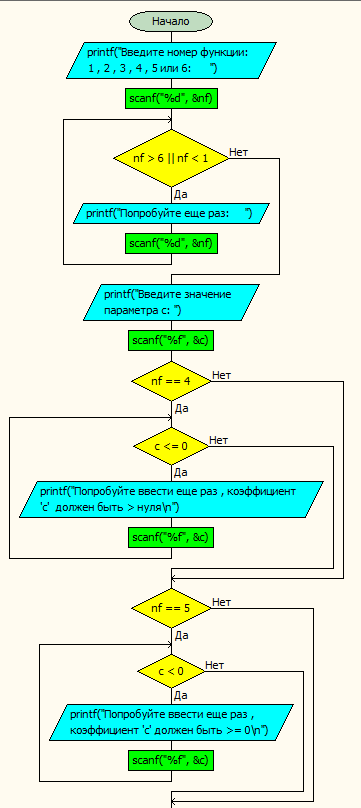
В данной функции не происходит ничего нового по сравнению с предыдущей, различие в том, что теперь пользователь высчитывает значение функции в точке , значение которой опять же вводит на свое усмотрение с клавиатуры.

1. (Блок-схема функции poisk\_korni):

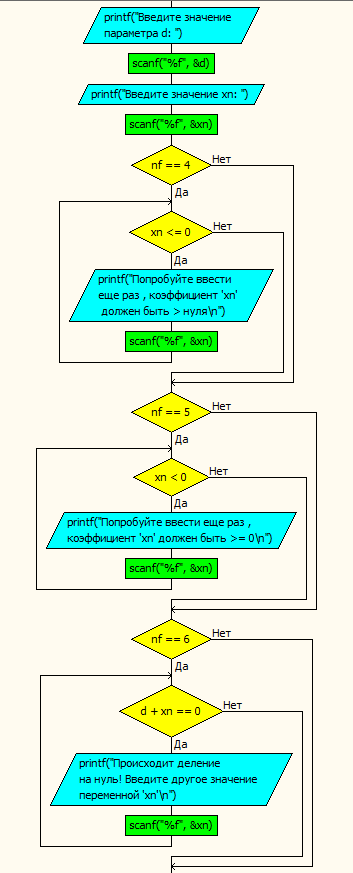


В данной функции идет вычисление приближенного корня конкретной функции . Здесь также, как и в методе касательных, присутствует предел по итерациям, если условие будет ложным, произойдет выход из цикла, в ином случае, если будет истина, начнется вычисление корня согласно алгоритму данного метода.

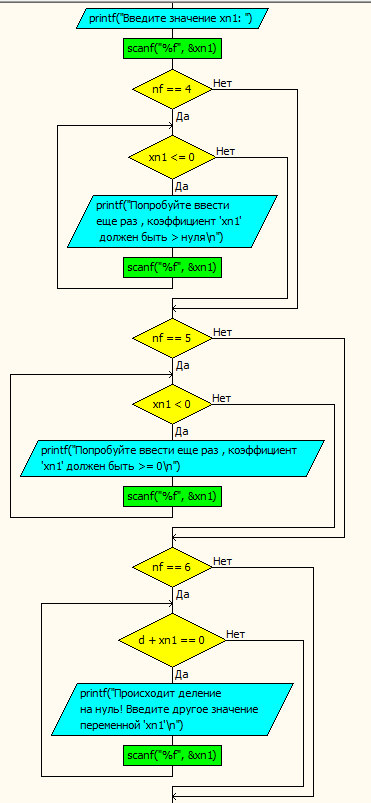
1. (Блок-схема главной функции main).



Продолжение ниже…



Продолжение ниже…



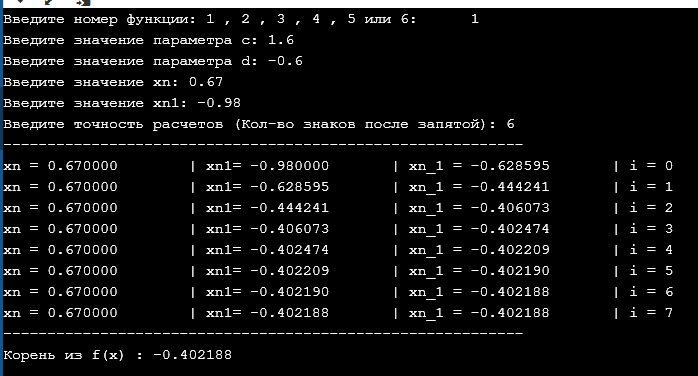
Продолжение ниже…

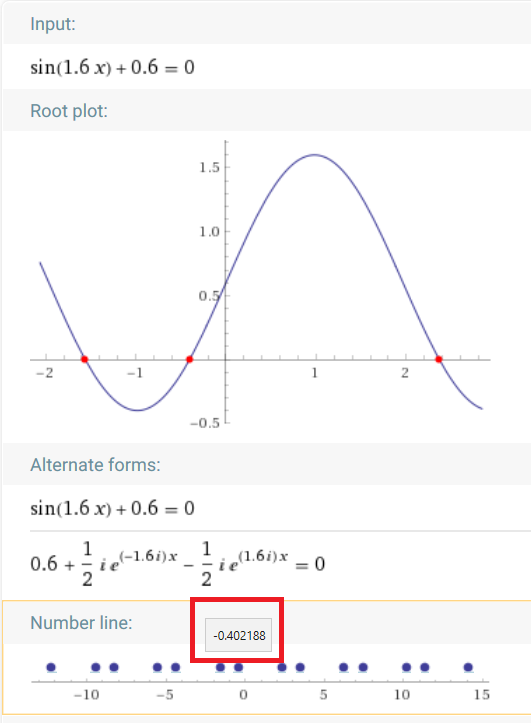


В данной функции пользователь выбирает функцию, приближенный корень которой хочет узнать, вводит соответствующие параметры и смотрит на результат. К нему мы сейчас и перейдем.

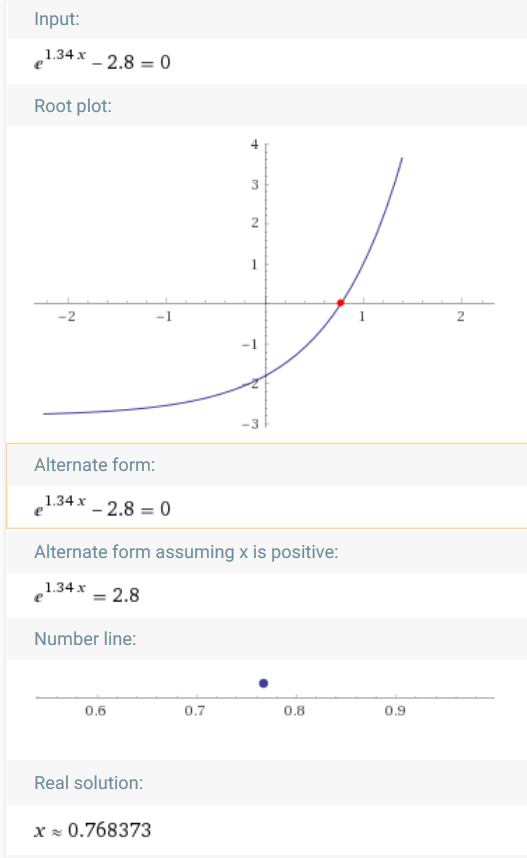
Проверка работоспособности данной программы.

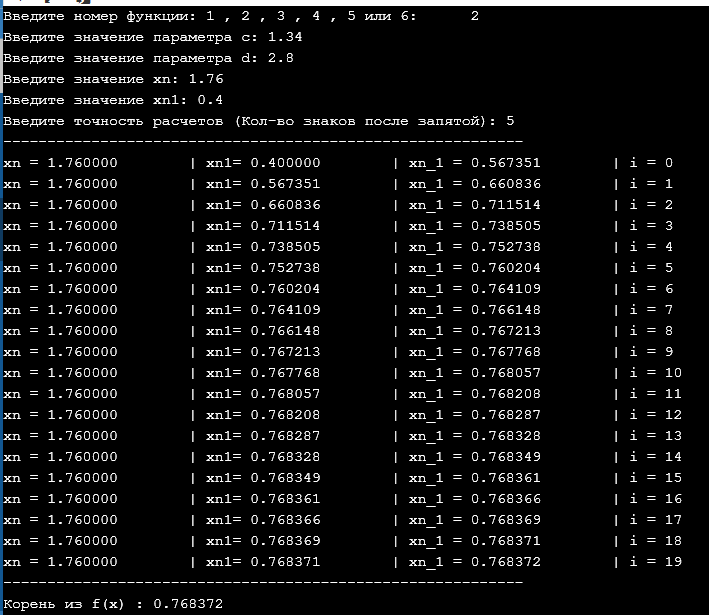
1). Функция синуса.



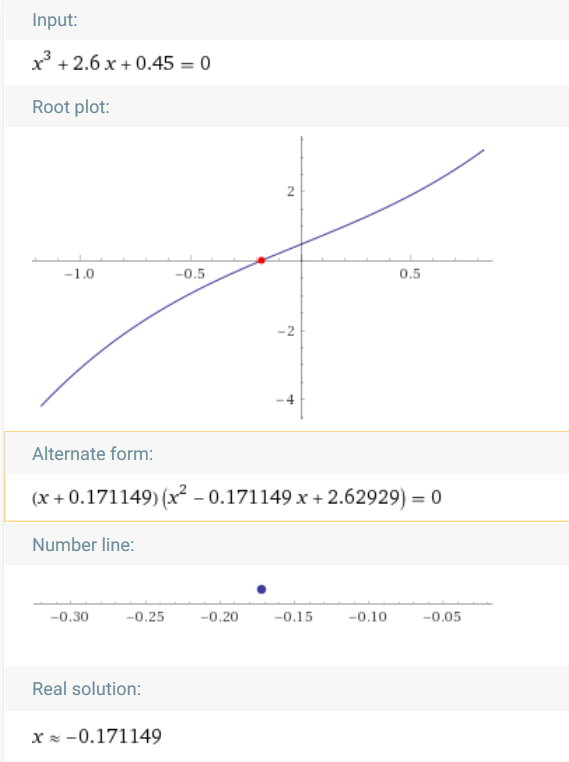


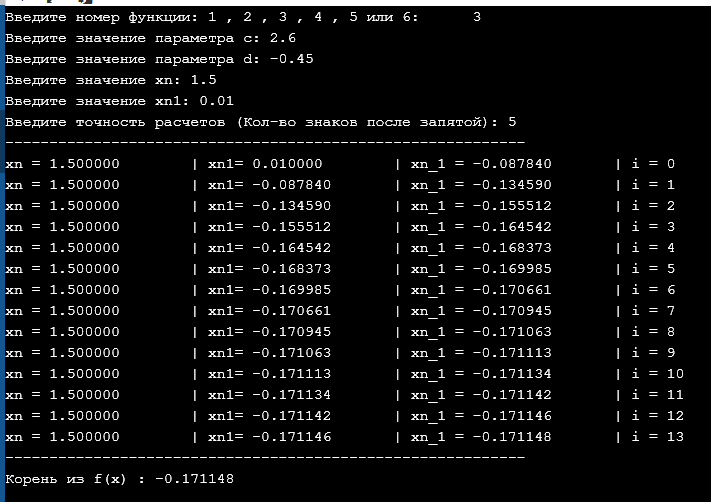
2). Функция с экспонентой.



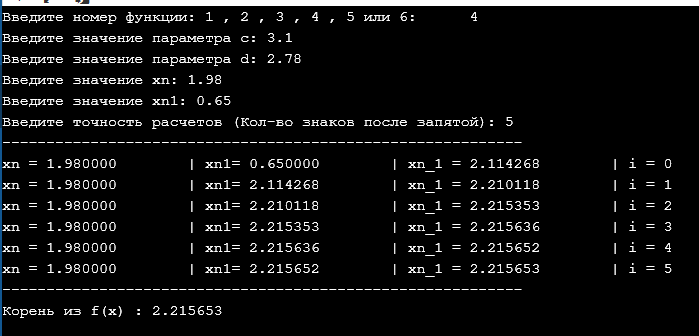


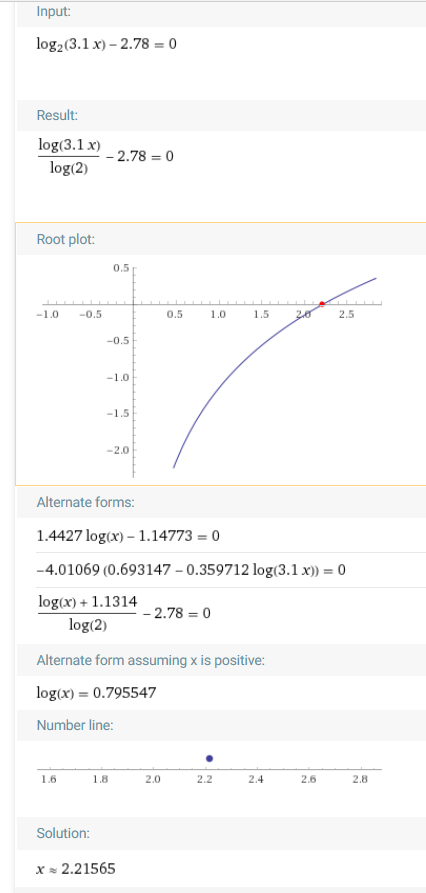
3). Функция с кубом.



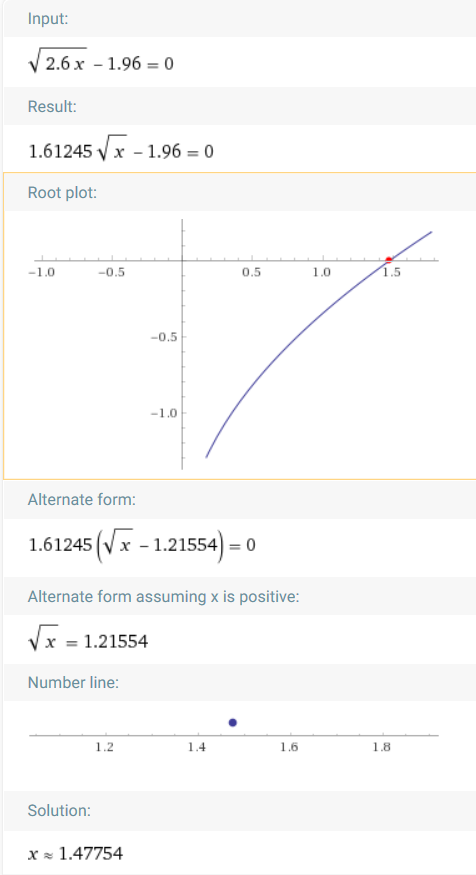


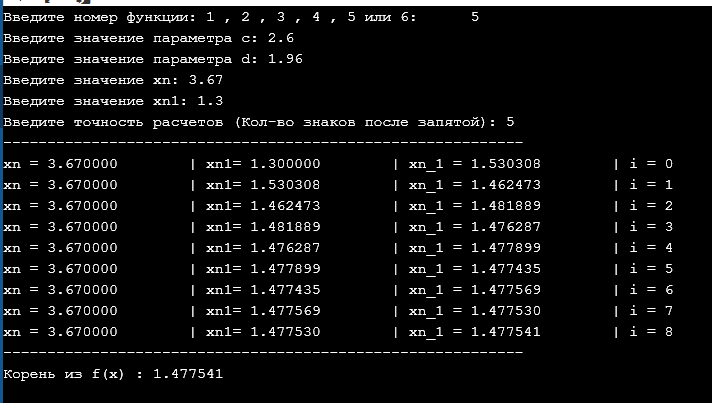
4). Функция логарифма.



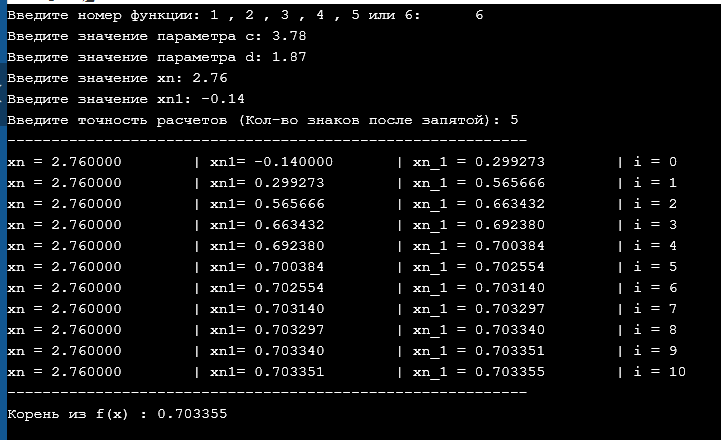


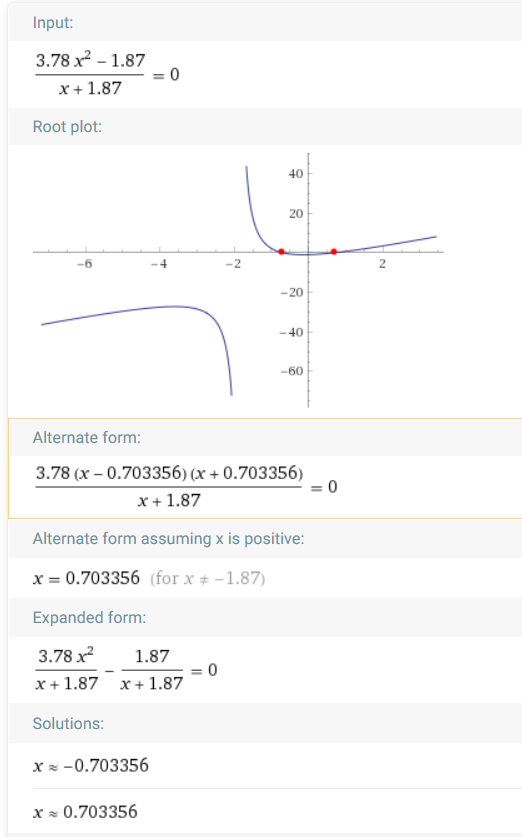
5). Функция с корнем.





6). Функция с дробью.





Вывод по данному методу: пользователь увидел метод секущих в действии. Ответы, в некоторых случаях, расходятся с истиной на одну или две десятые. Так как мы ищем приближенный корень, то такая погрешность допустима. Можно заметить, что в данном методе тригонометрической функции синуса не потребовалось так много итераций для вычисления приближенного корня, нежели как в методе касательных. Какой из этих двух продемонстрированных методов вам покажется привлекательнее, удобнее и проще, тем и пользуйтесь.

**Часть №2 Разработка игровой программы**

Использованный литературный источник [5]

В данном разделе пользователь имеет возможность протестировать игру на языке С. А в дальнейшем, если он захочет, может создать ее сам.

В моем варианте выпала игра “Жизнь”. Сейчас я вам объясню правила данной игры:

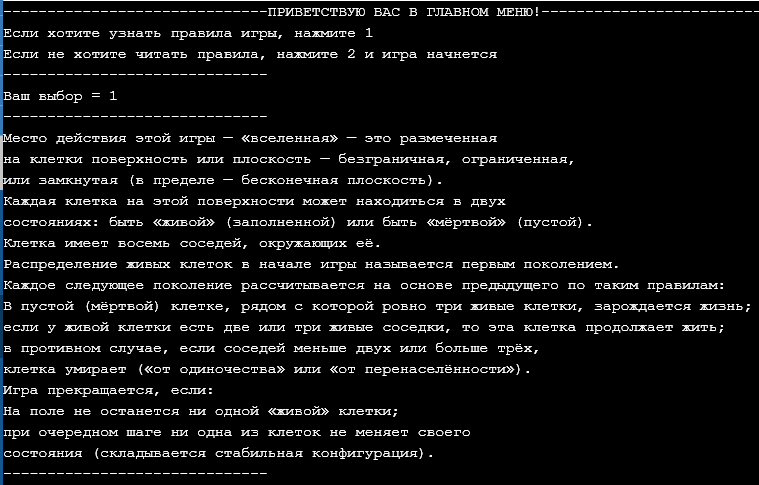
* имеется поле определенной длины и ширины, на поле присутствуют клетки;
* каждая клетка может быть либо живой, либо мертвой;
* распределение живых клеток в самом начале игры называют первым поколением;
* каждое новое поколение создается на основе предыдущего;
* если текущая клетка мертва, но у нее есть три живых клетки соседа, то мертвая клетка оживает;
* если текущая клетка жива и у этой живой клетки есть либо 2, либо 3 живых клетки- соседа, то данная живая клетка продолжает жить и существовать;
* иначе, если данное условие нарушается или недействительно, то живая клетка умирает;
* игра заканчивается, если на поле не осталось живых клеток, либо конфигурация при каждом новом поколении зацикливается – клетки не изменяются.

Думаю, стоит перейти к самой реализации игры.

Начнем с наглядного представления.

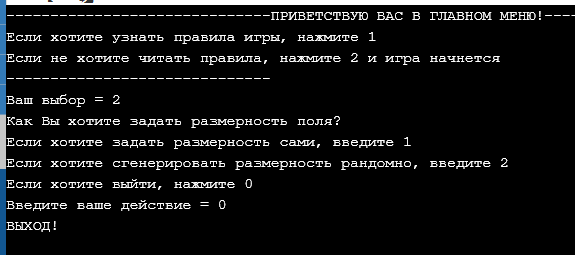
Как только запускается программа, перед глазами пользователя выплывает главное меню игры, в котором игрок может либо почитать правила игры, для чего ему необходимо нажать единицу, либо сразу начать игру, для чего требуется нажать два.

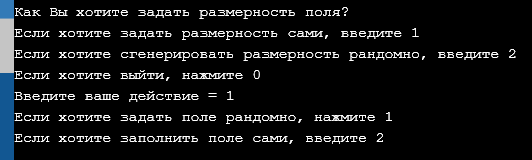
Начнем с правил, по которым происходят все действия во время самой игры.



Если же пользователь нажал двойку, происходит следующий выбор. Игроку предоставляется возможность между выбором размерности поля, то есть он может задать размерность самостоятельно, для чего ему потребуется вновь ввести единицу, или же он может ввести двойку, тогда размерность поля сгенерируется рандомно, еще пользователь может ввести нуль, в данном случае ничего не произойдет, как таковая игра просто закончится.

Демонстрация выхода из игры.





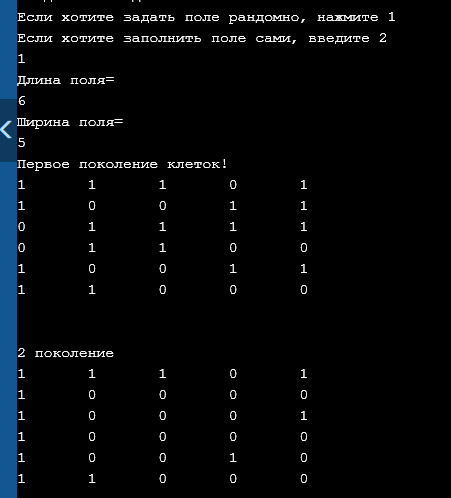
Предположим, пользователь захотел ввести размерность самостоятельно, снова выбор, нажать единицу и задать самое первое поколение клеток рандомно, либо ввести двойку и заполнить поле самостоятельно.

Я продемонстрирую оба случая.

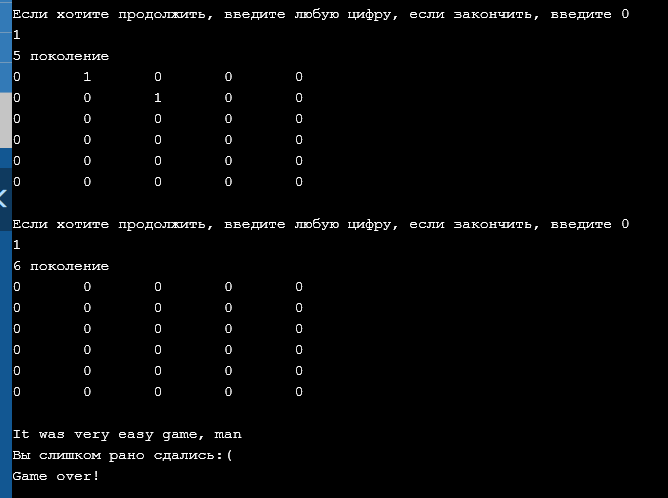
Ну а начал игрок, как мы можем увидеть ниже, с размеров (6x5) и выбрал рандомную генерацию клеток, то есть на их появление он никак уже не повлияет, пользователю остается лишь наблюдать появление новых поколений, пока на поле не останется живых клеток – тогда игра закончится, либо пока поколения не начнут зацикливаться – тогда игра в скором времени тоже закончится.

После генерации самого первого поколения, алгоритм сразу же выводит новое – второе поколение.

Дальше игроку дается выбор, что он хочет сделать: продолжить игру - тогда ему следует нажать любую цифру, кроме нуля или ввести любое число, либо, если он хочет уже закончить игру, пускай нажимает нуль. Также, при удачном окончании игры без зацикливания, будет выведено сообщение, поясняющее, насколько тяжелым было это “выживание”.



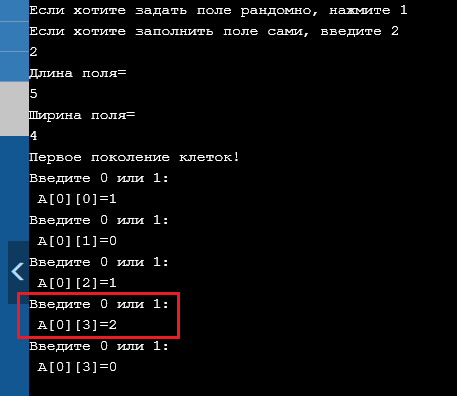
Пользователь нажал любую цифру, кроме нуля, и пошла дальнейшая генерация новых поколений.



Как видно из скриншотов работоспособности программы, 6-ое поколение стало последним. Живых клеток не осталось, и игра закончилась, выведя соответствующее сообщение.

Теперь попробуем другую вариацию данной игры, ситуацию, когда пользователь сам задает значения клеток первого поколения.

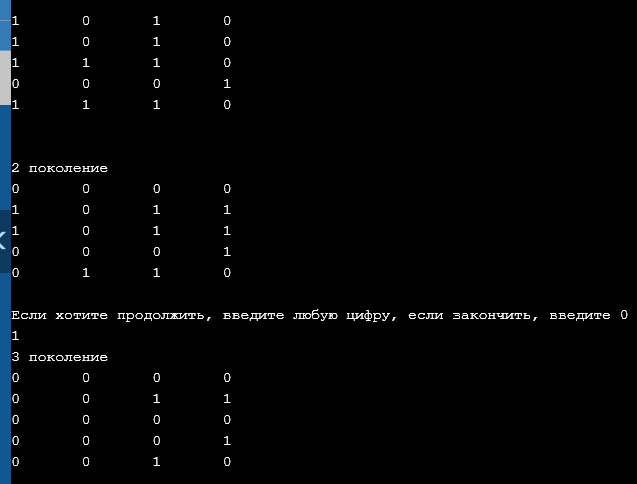
Прилагаю соответствующие скриншоты работы программы.

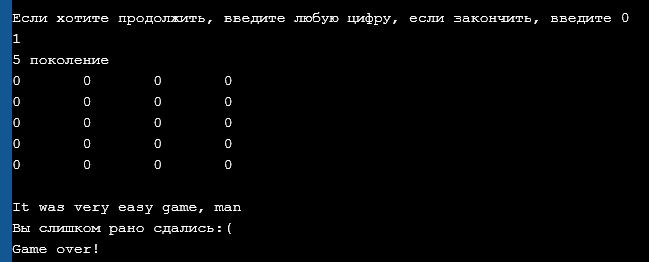


Здесь пользователь уже выбрал размерность поля (5x4). Демонстрирую ситуацию на проверку “дурака” при вводе элемента массива с индексами A[0][3].

После того, как пользователь закончил заполнение поля, сразу же вывелось сформированное первое поколение и далее - второе, сформированное на основе первого. Опять пользователю дается выбор продолжить игру или закончить ее.

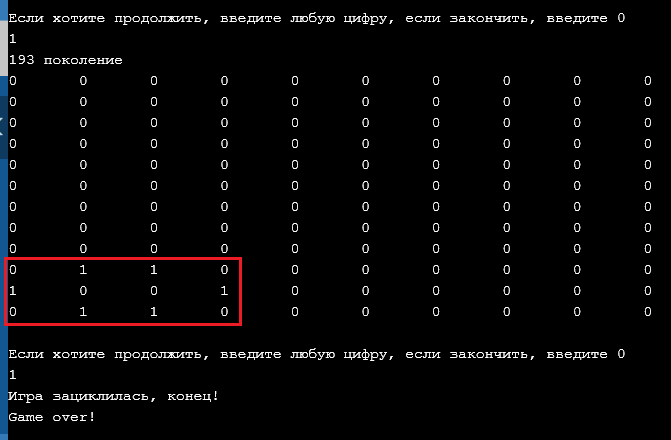
Давайте продолжим…





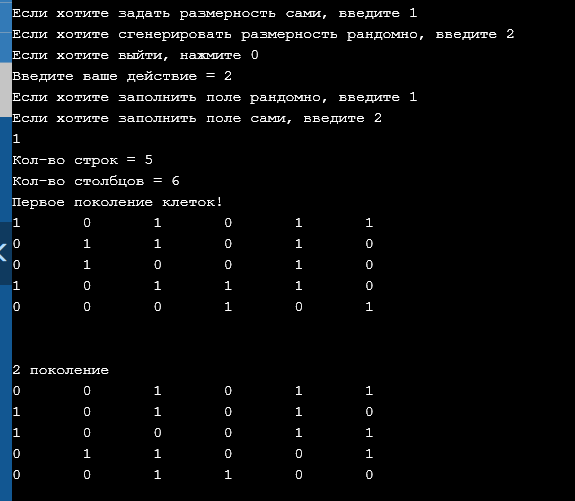
Генерация новых поколений прошла успешно, без каких-либо зацикливаний.

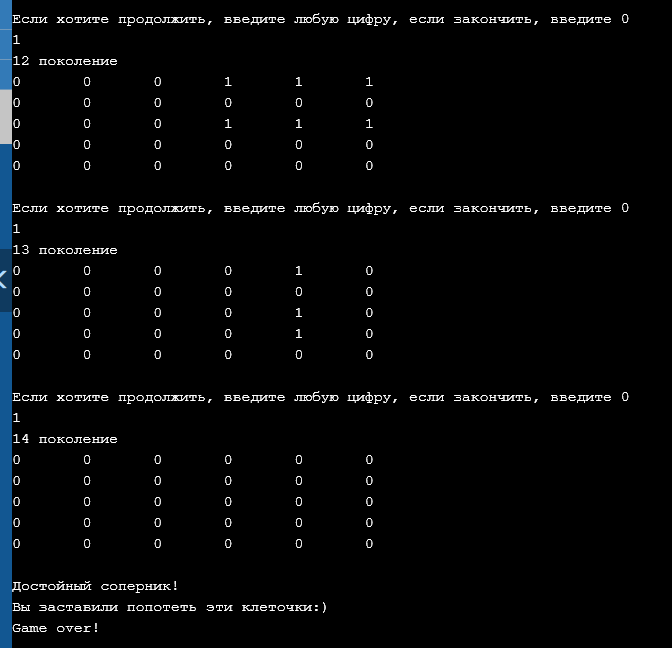
Теперь продемонстрирую ситуации, когда пользователь выбрал размерность поля задать рандомно и генерацию поля тоже выбрал рандомно.



Размерность поля была сгенерирована (12x16). Произошла ситуация как такового зацикливания полей, получилось “устойчивое” поле, называемое – “ульем”.

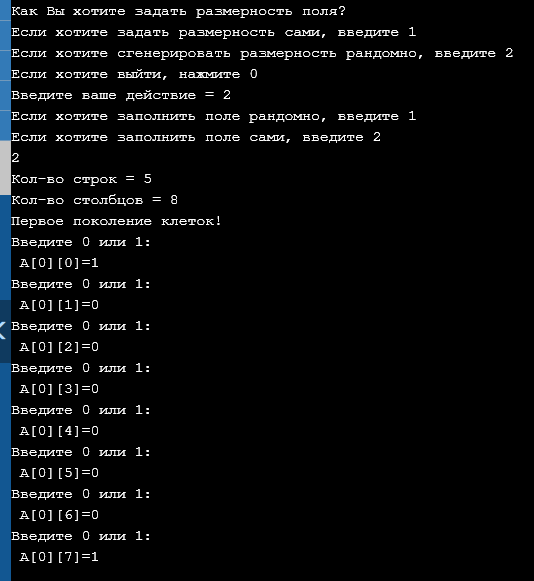
Повторная демонстрация рандомной генерации поля с рандомной генерацией размерности предложена ниже.

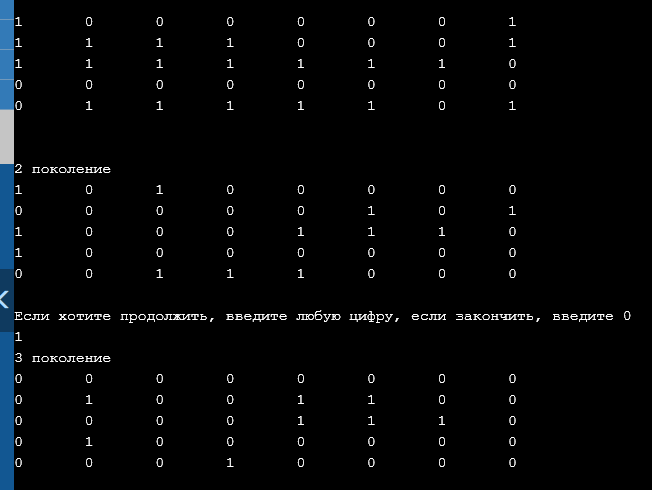


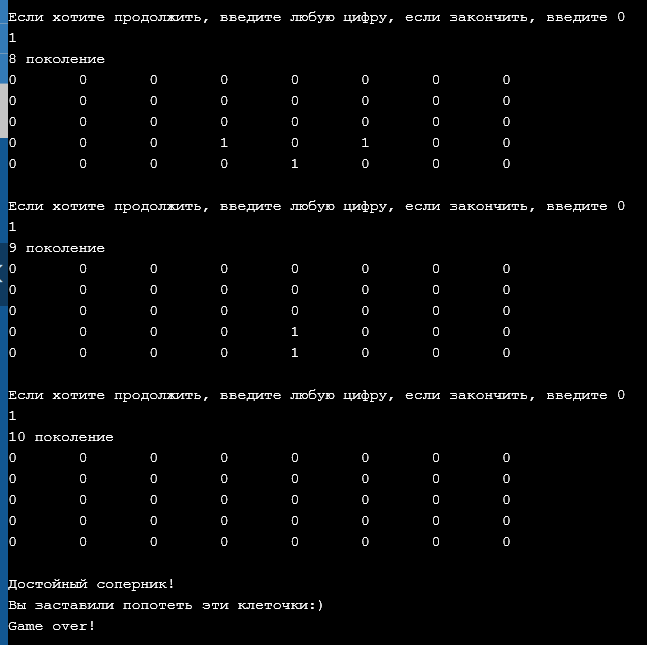


Как видно из скриншотов выше, генерация прошла успешно и без каких-либо проблем.

Теперь вариант реализации, при которой размерность будет рандомной, но все поле пользователь может ввести сам. (Примечание! Данный выбор не очень целесообразен, так как при большой размерности пользователю просто надоест вводить значения клеток)







Генерация прошла успешно. Никаких устойчивых полей замечено не было.

Теперь стоит перейти к листингу всей программы. Пояснения к каждой функции будут написаны при вставке блок-схем (стр.88).

#include<stdio.h> //стандартная библиотека ввода-вывода данных.

#include<math.h> //математическая библиотека в языке С.

#include<stdlib.h> //стандартная библотека в С, содержащая в себе функции, занимающиеся выделением памяти, контролем процесса выполнения программы, преобразованием типов и другое.

#include<time.h> //заголовочный файл стандартной библиотеки языка программирования C, содержащий типы и функции для работы с датой и временем.

int dlina, shirina; //глобальные переменные, отвечающие за длину и ширину игрового поля.

void rules\_of\_game(); //функция просто выводит на экран правила игры.

void rules\_of\_game()

{

printf("------------------------------\n");

printf("Место действия этой игры — «вселенная» — это размеченная на клетки поверхность или плоскость — безграничная, ограниченная, или замкнутая (в пределе — бесконечная плоскость).\n");

printf("Каждая клетка на этой поверхности может находиться в двух состояниях: быть «живой» (заполненной) или быть «мёртвой» (пустой).\n");

printf("Клетка имеет восемь соседей, окружающих её.\n");

printf("Распределение живых клеток в начале игры называется первым поколением.\n");

printf("Каждое следующее поколение рассчитывается на основе предыдущего по таким правилам:\n");

printf("В пустой (мёртвой) клетке, рядом с которой ровно три живые клетки, зарождается жизнь;\n");

printf("если у живой клетки есть две или три живые соседки, то эта клетка продолжает жить;\n");

printf("в противном случае, если соседей меньше двух или больше трёх, клетка умирает («от одиночества» или «от перенаселённости»).\n");

printf("Игра прекращается, если:\n");

printf("На поле не останется ни одной «живой» клетки;\n");

printf("при очередном шаге ни одна из клеток не меняет своего состояния (складывается стабильная конфигурация).\n");

printf("------------------------------\n");

}

void pole(int [][shirina], int [][shirina], int, int); //прототип фун-ии, создающей первое поколение клеток.

void pole(int A[][shirina], int B[][shirina], int choose, int dlina) //реализация этой самой фун-ии.

{

int i,j; //данные переменные служат для прохода по всему массиву.

printf("Первое поколение клеток!\n"); //вывод сообщения на экран.

if(choose==1) //сравнение,если вы выбрали единицу.

{

for(i=0;i<dlina;i++)

{ //цикл для массива.

for(j=0;j<shirina;j++)

{

A[i][j]=rand()%2; //Рандомно заполняю игровое поле нулями и единицами:0 - клетка мертва, 1 - клетка жива.

B[i][j]=A[i][j]; /\*сохраняем значения первого поколения в новый массив.

В дальнейшем массив B[i][j] будет сохранять следующие поколения,

а A[i][j] настоящие, чтобы их сравнивать можно было.

(сначала 1 поколение = A[i][j],2 поколение = B[i][j], затем 3 поколение = B[i][j], а 2 поколение A[i][j]).\*/

printf("%d\t",A[i][j]); //Вывожу поле на экран.

}

printf("\n"); //перенос строки.

}

printf("\n"); //перенос строки.

}

if(choose==2) //сравнение,если вы выбрали двойку.

{

for(i=0;i<dlina;i++)

{ //цикл для массива.

for(j=0;j<shirina;j++)

{

while(1) //пока истина.

{

printf("Введите 0 или 1:\n A[%d][%d]=",i,j); //вывод на экран значения переменных "i" и "j".

scanf("%d",&A[i][j]); //вводим с клавиатуры значение элемента массива.

if(A[i][j]==1 || A[i][j]==0) //если условие верно, выходим из цикла while.

break; //прерывание цикла.

}

}

printf("\n"); //перенос строки.

}

for(i=0;i<dlina;i++)

{ //цикл для массива.

for(j=0;j<shirina;j++)

{

B[i][j]=A[i][j]; /\*сохраняем значения первого поколения в новый массив.

В дальнейшем массив B[i][j] будет сохранять следующие поколения,

а A[i][j] настоящие, чтобы их сравнивать можно было.

(сначала 1 поколение = A[i][j],2 поколение = B[i][j], затем 3 поколение = B[i][j], а 2 поколение A[i][j]).\*/

printf("%d\t",A[i][j]); //Вывожу поле на экран.

}

printf("\n"); //перенос строки.

}

printf("\n"); //перенос строки.

}

}

void est\_sosed(int [][shirina], int [][shirina], int); //прототип фун-ии на проверку соседей у клетки.

void est\_sosed(int A[][shirina], int B[][shirina], int dlina) //ф-ия проверяет количество соседей у клетки.

{

int i,j,zhivie=0; //"i" и "j" служат для пробежки по массиву, "zhivie" для подсчета кол-ва живых клеток.

for(i=0;i<dlina;i++)

{ //цикл для пробега по всему массиву.

for(j=0;j<shirina;j++)

{

if(i==0 && j==0) //условие выполняется, если попали в ВЕРХНЮЮ ЛЕВУЮ УГЛОВУЮ клетку.

{

if(A[i][j]==0) //проверка на то, что клетка мертвая.

{

zhivie=A[i][j+1]+A[i+1][j+1]+A[i+1][j]; //подсчет живых соседей угловой клетки.

if(zhivie==3) B[i][j]=1; //если живых соседей 3, то клетка оживает.

}

else //если клетка оказалась живой

{

zhivie=A[i][j+1]+A[i+1][j+1]+A[i+1][j]; //подсчет живых соседей угловой клетки.

if(zhivie<2 || zhivie>3) B[i][j]=0; //если соседей меньше двух или больше трех, то клетка умирает.

}

}

if(i==0 && j!=0 && j!=shirina-1) //условие выполняется, если бежим по НУЛЕВОЙ СТРОКЕ(исключая угловые клетки).

{

if(A[i][j]==0) //проверка на то, что клетка мертвая.

{

zhivie=A[i][j-1]+A[i][j+1]+A[i+1][j-1]+A[i+1][j]+A[i+1][j+1]; //подсчет живых соседей побокам и снизу(три).

if(zhivie==3) B[i][j]=1; //если живых соседей 3, то клетка оживает.

}

else //если клетка оказалась живой.

{

zhivie=A[i][j-1]+A[i][j+1]+A[i+1][j-1]+A[i+1][j]+A[i+1][j+1];

if(zhivie<2 || zhivie>3) B[i][j]=0; //если соседей меньше двух или больше трех, то клетка умирает.

}

}

if(i==0 && j==shirina-1) //условие выполняется, если попали в ВЕРХНЮЮ ПРАВУЮ УГЛОВУЮ клетку.

{

if(A[i][j]==0) //проверка на то, что клетка мертвая.

{

zhivie=A[i][j-1]+A[i+1][j-1]+A[i+1][j]; //подсчет живых соседей угловой клетки.

if(zhivie==3) B[i][j]=1; //если живых соседей 3, то клетка оживает.

}

else //если клетка оказалась живой.

{

zhivie=A[i][j-1]+A[i+1][j-1]+A[i+1][j]; //подсчет живых соседей угловой клетки.

if(zhivie<2 || zhivie>3) B[i][j]=0;

}

}

if(i==dlina-1 && j==shirina-1) //условие выполняется, если попали в НИЖНЮЮ ПРАВУЮ УГЛОВУЮ клетку.

{

if(A[i][j]==0) //проверка на то, что клетка мертвая.

{

zhivie=A[i][j-1]+A[i-1][j]+A[i-1][j-1]; //подсчет живых соседей угловой клетки.

if(zhivie==3) B[i][j]=1; //если живых соседей 3, то клетка оживает.

}

else //если клетка оказалась живой.

{

zhivie=A[i][j-1]+A[i-1][j]+A[i-1][j-1]; //подсчет живых соседей угловой клетки.

if(zhivie<2 || zhivie>3) B[i][j]=0;

}

}

if(i==dlina-1 && j==0) //условие выполняется, если попали в НИЖНЮЮ ЛЕВУЮ УГЛОВУЮ клетку.

{

if(A[i][j]==0) //проверка на то, что клетка мертвая.

{

zhivie=A[i-1][j]+A[i-1][j+1]+A[i][j+1]; //подсчет живых соседей угловой клетки.

if(zhivie==3) B[i][j]=1; //если живых соседей 3, то клетка оживает.

}

else //если клетка оказалась живой.

{

zhivie=A[i-1][j]+A[i-1][j+1]+A[i][j+1]; //подсчет живых соседей угловой клетки.

if(zhivie<2 || zhivie>3) B[i][j]=0;

}

}

if(i!=0 && j==0 && i!=dlina-1) //условие выполняется, если бежим по НУЛЕВОМУ СТОЛБЦУ(исключая угловые клетки).

{

if(A[i][j]==0) //проверка на то, что клетка мертвая.

{

zhivie=A[i+1][j]+A[i+1][j+1]+A[i][j+1]+A[i-1][j]+A[i-1][j+1]; //подсчет живых соседей сверху один, снизу один, три справа.

if(zhivie==3) B[i][j]=1; //если живых соседей 3, то клетка оживает.

}

else //если клетка оказалась живой.

{

zhivie=A[i+1][j]+A[i+1][j+1]+A[i][j+1]+A[i-1][j]+A[i-1][j+1]; //подсчет живых соседей сверху один, снизу один, три справа.

if(zhivie<2 || zhivie>3) B[i][j]=0; //если соседей меньше двух или больше трех, то клетка умирает.

}

}

if(i!=0 && j==shirina-1 && i!=dlina-1) //условие выполняется, если бежим по ПОСЛЕДНЕМУ СТОЛБЦУ(исключая угловые клетки).

{

if(A[i][j]==0) //проверка на то, что клетка мертвая.

{

zhivie=A[i][j-1]+A[i+1][j]+A[i-1][j]+A[i-1][j-1]+A[i+1][j-1]; //подсчет живых соседей сверху один, снизу один, три слева.

if(zhivie==3) B[i][j]=1; //если живых соседей 3, то клетка оживает.

}

else //если клетка оказалась живой.

{

zhivie=A[i][j-1]+A[i+1][j]+A[i-1][j]+A[i-1][j-1]+A[i+1][j-1]; //подсчет живых соседей сверху один, снизу один, три слева.

if(zhivie<2 || zhivie>3) B[i][j]=0; //если соседей меньше двух или больше трех, то клетка умирает

}

}

if(i==dlina-1 && j!=0 && j!=shirina-1) //условие выполняется, если бежим по ПОСЛЕДНЕЙ СТРОКЕ(исключая угловые клетки).

{

if(A[i][j]==0) //проверка на то, что клетка мертвая.

{

zhivie=A[i][j-1]+A[i][j+1]+A[i-1][j]+A[i-1][j-1]+A[i-1][j+1]; //подсчет живых соседей слева один, справа один, сверху трое.

if(zhivie==3) B[i][j]=1; //если живых соседей 3, то клетка оживает.

}

else //если клетка оказалась живой.

{

zhivie=A[i][j-1]+A[i][j+1]+A[i-1][j]+A[i-1][j-1]+A[i-1][j+1]; //подсчет живых соседей слева один, справа один, сверху трое.

if(zhivie<2 || zhivie>3) B[i][j]=0; //если соседей меньше двух или больше трех, то клетка умирает.

}

}

if(i!=0 && j!=0 && i!=dlina-1 && j!=shirina-1) //условие выполняется для ОБЩЕГО СЛУЧАЯ(исключая угловые клетки).

{

if(A[i][j]==0) //проверка на то, что клетка мертвая.

{

zhivie=A[i][j-1]+A[i-1][j-1]+A[i-1][j]+A[i-1][j+1]+A[i][j+1]+A[i+1][j+1]+A[i+1][j]+A[i+1][j-1]; //подсчет живых соседей слева один, справа один, сверху трое, снизу трое.

if(zhivie==3) B[i][j]=1; //если живых соседей 3, то клетка оживает.

}

else //если клетка оказалась живой.

{

zhivie=A[i][j-1]+A[i-1][j-1]+A[i-1][j]+A[i-1][j+1]+A[i][j+1]+A[i+1][j+1]+A[i+1][j]+A[i+1][j-1]; //подсчет живых соседей слева один, справа один, сверху трое, снизу трое.

if(zhivie<2 || zhivie>3) B[i][j]=0; //если соседей меньше двух или больше трех, то клетка умирает.

}

}

}

}

}

int main() //главная функция.

{

int a=1, y=2, i, j, proverka=0, m=0, choose, choose\_generation=10, choose\_path; /\*переменная a - отвечает за то хотите ли вы продолжить игру или нет,

m - отвечает за проверку поколений на совпадение, y - отвечает за номер нового поколения,

proverka - проверяет кол-во живых соседей, choose - ваш выбор для первого поколения, choose\_generation - за выбор генерации размерности

choose\_path - за то, что выбрано в гланом меню.\*/

srand(time(NULL));//привязка генерации рандомных чисел к машинному времени.

printf("------------------------------ПРИВЕТСТВУЮ ВАС В ГЛАВНОМ МЕНЮ!------------------------------\n");//вывод сообщения на экран.

printf("Если хотите узнать правила игры, нажмите 1\n"); //вывод сообщения на экран.

printf("Если не хотите читать правила, нажмите 2 и игра начнется\n"); //вывод сообщения на экран.

printf("------------------------------\n"); //вывод сообщения на экран.

printf("Ваш выбор = "); scanf("%d",&choose\_path);//вывод сообщения на экран + ввод значения переменной "choose\_path".

while(choose\_path > 2 || choose\_path < 1) //проверка на "дурака".

{

printf("Ваш выбор не соответствуюет данным!!! Попробуйте еще раз.\n");//вывод сообщения на экран.

scanf("%d",&choose\_path); //ввод значения переменной "choose\_path".

}

if(choose\_path == 1) rules\_of\_game();//если выбрана единица, вызов фун-ии "rules\_of\_game".

else //иначе, если введена двойка.

{

printf("Как Вы хотите задать размерность поля?\n"); //вывод сообщения на экран.

printf("Если хотите задать размерность сами, введите 1\n"); //вывод сообщения на экран.

printf("Если хотите сгенерировать размерность рандомно, введите 2\n"); //вывод сообщения на экран.

printf("Если хотите выйти, нажмите 0\n"); //вывод сообщения на экран.

do

{

printf("Введите ваше действие = "); //вывод сообщения на экран.

scanf("%d",&choose\_generation); //ввод значения переменной "choose\_generation"

while(choose\_generation > 2 || choose\_generation < 0) //проверка "дурака".

{

printf("Введена недопустимая операция, попробуйте еще раз.\n"); //вывод сообщения на экран.

scanf("%d",&choose\_generation); //повторный ввод значения переменной "choose\_generation".

}

if(choose\_generation==0) {printf("ВЫХОД!\n"); return 0;}//если выбран нуль, происходит выход и конец программы.

else if(choose\_generation==1) //если выбрали единицу.

{

printf("Если хотите задать поле рандомно, нажмите 1\n"); //вывод сообщения на экран.

printf("Если хотите заполнить поле сами, введите 2\n"); //вывод сообщения на экран.

scanf("%d",&choose); //ввод значения переменной "choose"

while(choose > 2 || choose < 1) //цикл на проверку "дурака".

{

printf("Введите соответствующий номер\n"); //вывод сообщения на экран.

scanf("%d",&choose); //повторный ввод переменной "choose".

}

if(choose==1) //если выбрали единицу.

{

printf("Длина поля=\n"); //вывод сообщения на экран.

scanf("%d",&dlina); //вводим кол-во строк поля.

printf("Ширина поля=\n"); //вывод сообщения на экран.

scanf("%d",&shirina); //вводим кол-во столбцов поля.

while(dlina < 3 || shirina < 3) //цикл на проверку "дурака".

{

printf("Нехватка соседей, введите минимум 3х3.\n"); //вывод сообщения на экран.

scanf("%d",&dlina); //повторный ввод переменной "dlina".

scanf("%d",&shirina); //повторный ввод переменной "shirina".

}

}

if(choose==2) //если выбрали двойку.

{

printf("Длина поля=\n"); //вывод сообщения на экран.

scanf("%d",&dlina); //вводим кол-во строк поля.

printf("Ширина поля=\n"); //вывод сообщения на экран.

scanf("%d",&shirina); //вводим кол-во столбцов поля.

while(dlina < 3 || shirina < 3) //цикл на проверку "дурака".

{

printf("Нехватка соседей, введите минимум 3х3.\n"); //вывод сообщения на экран.

scanf("%d",&dlina); //повторный ввод переменной "dlina".

scanf("%d",&shirina); //повторный ввод переменной "shirina".

}

}

}

else //если выбрана ни единица, ни нуль.

{

printf("Если хотите заполнить поле рандомно, введите 1\n"); //вывод сообщения на экран.

printf("Если хотите заполнить поле сами, введите 2\n"); //вывод сообщения на экран.

scanf("%d",&choose); //ввод значения переменной "choose"

while(choose>2 || choose<1) //цикл на проверку "дурака".

{

printf("Введите соответствующий номер\n"); //вывод сообщения на экран.

scanf("%d",&choose); //повторный ввод переменной "choose".

}

if(choose==1) //если выбрали единицу.

{

dlina=3+rand()%20; //задаю диапазон генерации значения переменной "dlina".

printf("Кол-во строк = %d\n",dlina); //вывод этого значения.

shirina=3+rand()%20; //задаю диапазон генерации значения переменной "shirina".

printf("Кол-во столбцов = %d\n",shirina); //вывод этого значения.

}

if(choose==2) //если выбрали двойку.

{

dlina=3+rand()%20; //задаю диапазон генерации значения переменной "dlina".

printf("Кол-во строк = %d\n",dlina); //вывод этого значения.

shirina=3+rand()%20; //задаю диапазон генерации значения переменной "shirina".

printf("Кол-во столбцов = %d\n",shirina); //вывод этого значения.

}

}

} while(choose\_generation !=1 && choose\_generation !=2);//пока не введено соответствующее значение.

int A[dlina][shirina],B[dlina][shirina]; //объявляем два массива (массив A отвечает за текущее поколение, массив B за следующее).

pole(A,B,choose,dlina); //вызов фун-ии, задающей изначальное поколение(клетки располагаются рандомно на поле).

printf("\n"); //перенос строки.

while(a!=0) //как только переменная "a" примет нуль, цикл прервется и игра закончится.

{

est\_sosed(A,B,dlina); //Вызов функции с генерацией следующего поколения.

for(i=0;i<dlina;i++)

{ //цикл для двумерного массива (пройдемся по всему полю).

for(j=0;j<shirina;j++)

{

if(A[i][j]==B[i][j]) //условие, если значения клеток настоящего поколения равны значениям следующего поколения, то переменную 'm' увеличиваем на единицу.

{

m++; //увеличение значения переменной на единицу.

}

if(m==dlina\*shirina) //если число совпадений равно площади поля, игра заканчивается, т.к новые поколения не обновляются, а просто зацикливаются(это не является ошибкой программы, просто клетки так встали).

{

a=0; //цикл прерывается и игра заканчивается.

printf("Игра зациклилась, конец!\n"); //вывод сообщения на экран.

break; //выход из цикла.

}

else //иначе.

{

for(i=0;i<dlina;i++) //Перезаписываем первый массив после выполнения правил.

{ //цикл для двумерного массива (пройдемся по всему полю).

for(j=0;j<shirina;j++)

{

A[i][j]=B[i][j]; /\*данное действие перезапись массива делаем для того,

чтобы можно было в дальнейшем сравнить предыдущее поколение и нынешнее.\*/

}

}

printf("%d поколение\n",y); //номер поколения.

for(i=0;i<dlina;i++)

{ //цикл для двумерного массива (пройдемся по всему полю).

for(j=0;j<shirina;j++)

{

printf("%d\t",A[i][j]); //печатаем текущее поколение.

}

printf("\n"); //перенос строки.

}

printf("\n"); //перенос строки.

proverka=0; //присваиваем переменной "proverka" значение нуля.

for(i=0;i<dlina;i++)

{ //цикл для двумерного массива (пройдемся по всему полю).

for(j=0;j<shirina;j++)

{

proverka+=A[i][j]; //проверяем, есть ли живые клетки на поле.

}

}

if(proverka!=0) /\*условие, если значение переменной 'proverka' не равно нулю(больше нуля, т.к клетка либо 1-жива,

0-мертва), поколение имеет хотя бы одну живую клетку.\*/

{

printf("Если хотите продолжить, введите любую цифру, если закончить, введите 0\n"); //вывод сообщения на экран.

scanf("%d",&a); //вводим повторно значение переменной "a".

}

else a=0; //цикл прерывается и игра заканчивается.

y++; //увеличиваем номер поколения на единицу.

//если условие соблюдается, вывод сообщения на экран.

if(y < 8 && a == 0) printf("It was very easy game, man\nВы слишком рано сдались:(\n");

//если условие соблюдается, вывод сообщения на экран.

if(y >= 8 && y < 16 && a == 0) printf("Достойный соперник!\nВы заставили попотеть эти клеточки:)\n");

//если условие соблюдается, вывод сообщения на экран.

if(y >=16 && a == 0) printf("Enough, please...\nВы получаете золотую медаль столь долгое выживание!!!\n");

}

}

if(a==0) //если условие соблюдается.

{

printf("Game over!\n"); //вывод сообщения на экран.

break; //если a==0 выходим из цикла.

}

}

}

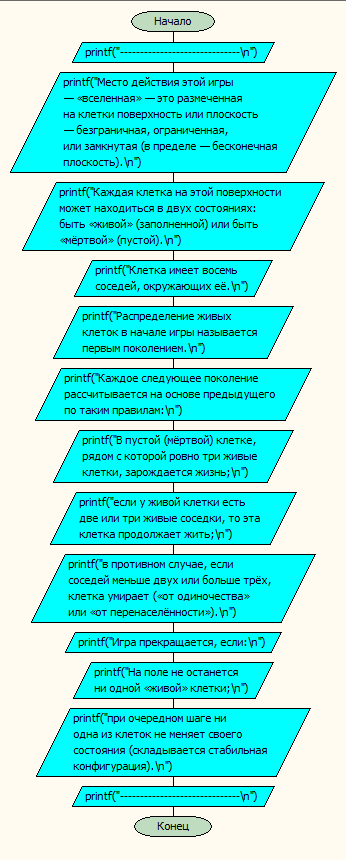
}

return 0; //функция возвращает нуль => выходим из функции, конец программы.

}

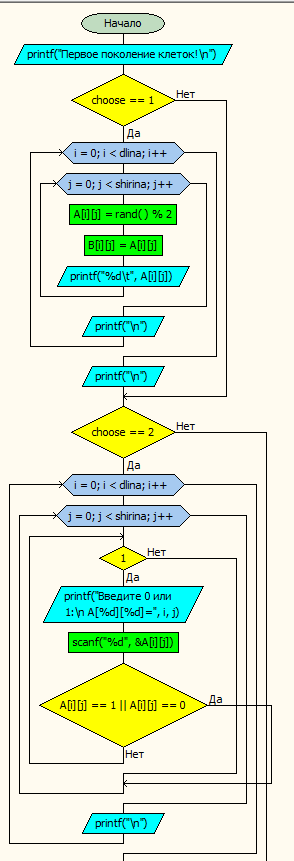
Теперь представляю блок-схемы.

1). Блок-схема функции rules\_of\_game.

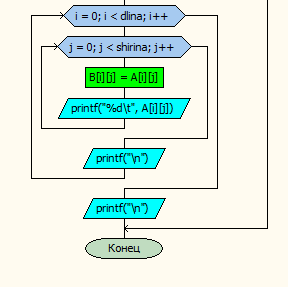


В данной функции особенного нечего нет, пользователь просто может прочитать правила, по которым работает игра.

2). Блок-схема функции pole.

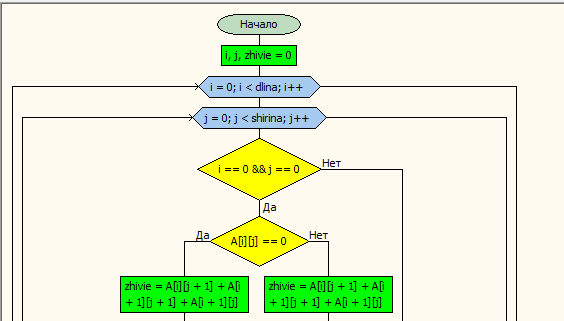


Продолжение ниже…

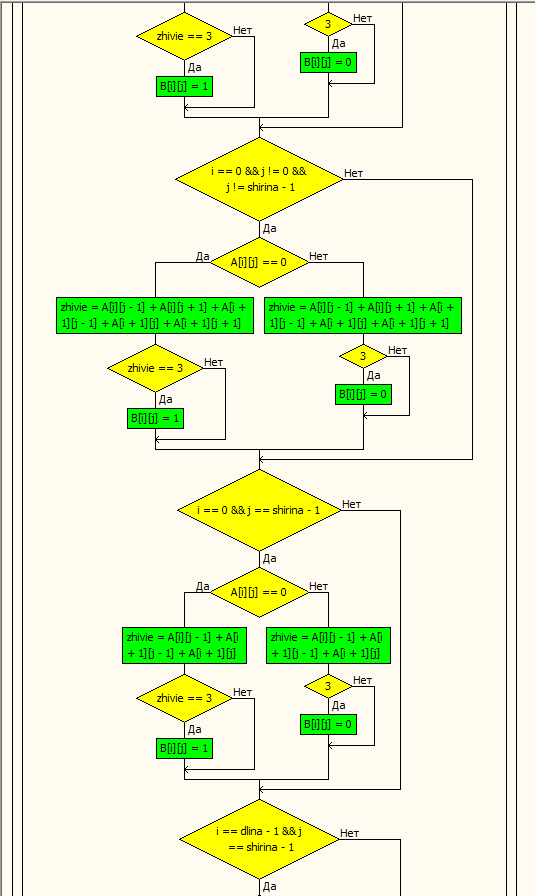


В данной функции пользователю предоставляется выбор того. как он хочет поступить, а именно, сгенерировать самое первое поколение клеток рандомно, то есть заполнить его нулями и единицами. Где нуль означает что клетка мертва, а единица – жива. Или пользователь может заполнить все поле самостоятельно, но при большой размерности данное действие будет нецелесообразным, как я подмечал выше. Здесь так же используется два двумерных массива, в массив A будет записано текущее поколение, а в массив B – следующее. Массив B будет генерироваться на основе массива A согласно правилам игры, описанным в блок-схеме rules\_of\_game.

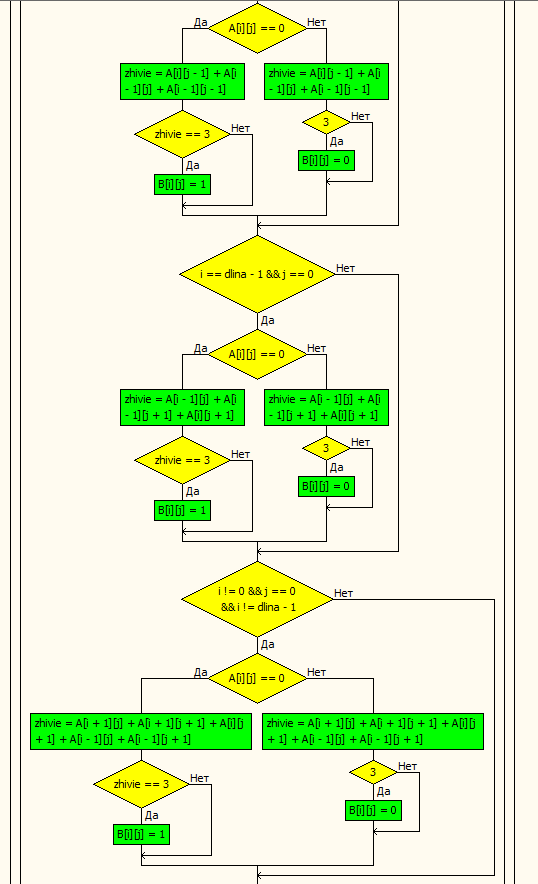
3). Блок-схема функции est\_sosed.



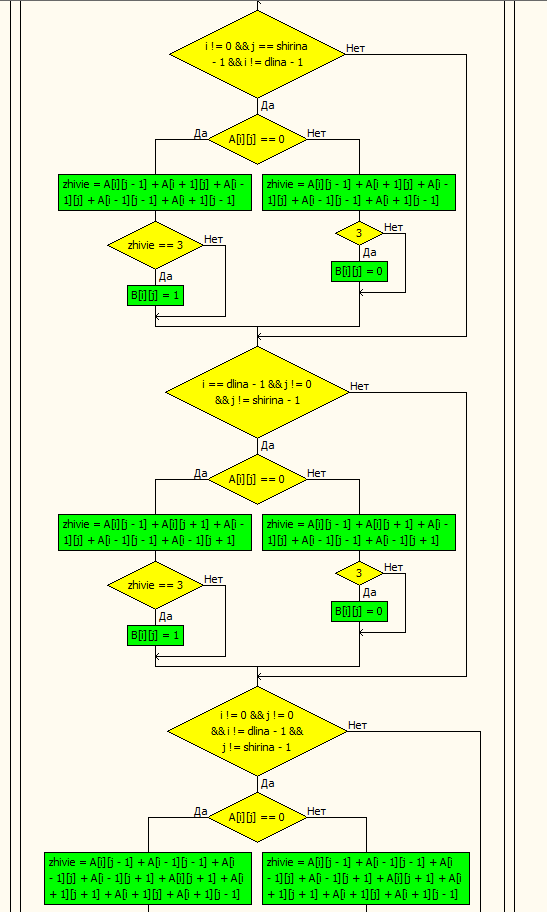
Продолжение ниже…



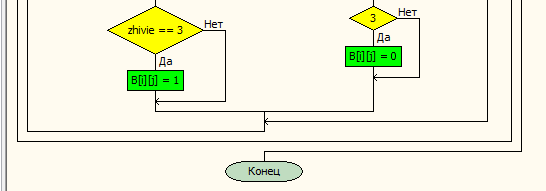
Продолжение ниже…



Продолжение ниже…



Продолжение ниже…

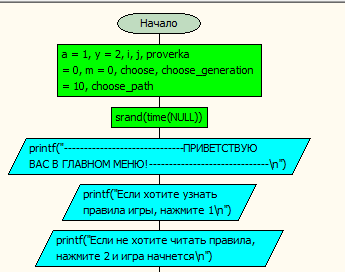


В данной функции идет проверка на наличие у каждой клетки соседей. Были рассмотрены все случаи расположения клеток:

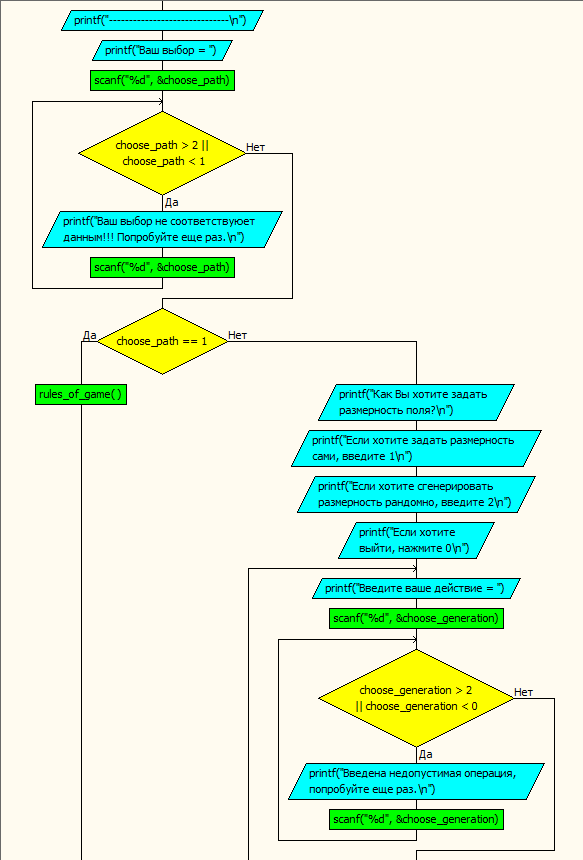
* верхняя левая угловая клетка;
* вся нулевая строка, за исключением угловых клеток;
* верхняя правая угловая клетка;
* нижняя правая угловая клетка;
* нижняя левая угловая клетка;
* весь нулевой столбец, за исключением угловых клеток;
* последний столбец, за исключением угловых клеток;
* последняя строка, за исключением угловых клеток;
* общий случай, за исключением угловых клеток.

Подсчитывается кол-во живых соседей и в соответствии с правилами игры генерируется новое поколение клеток.

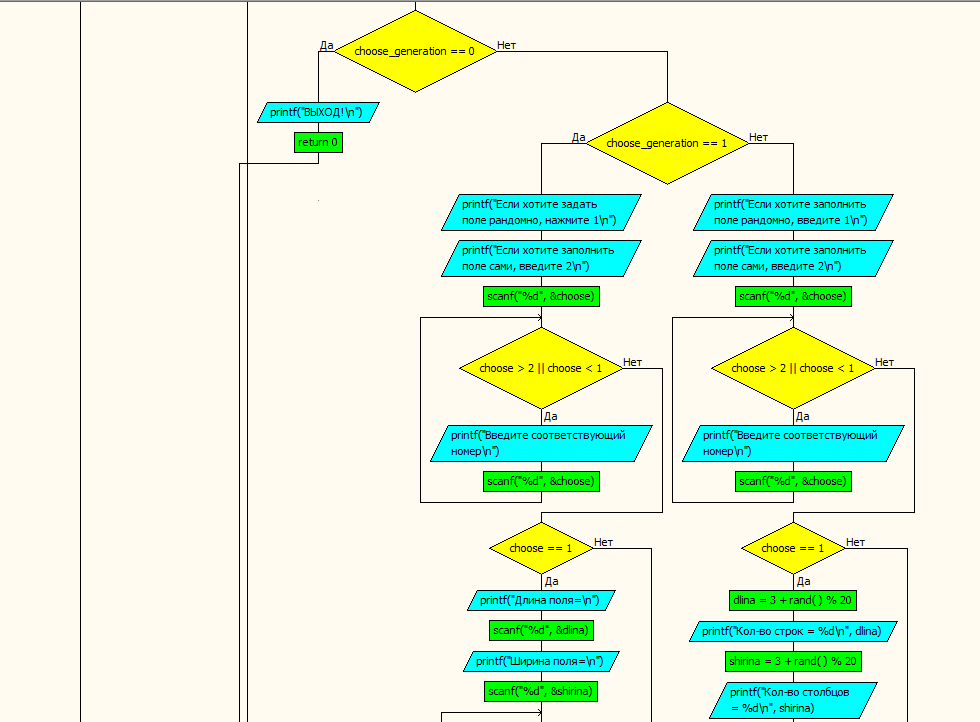
4). Блок-схема главной функции main.



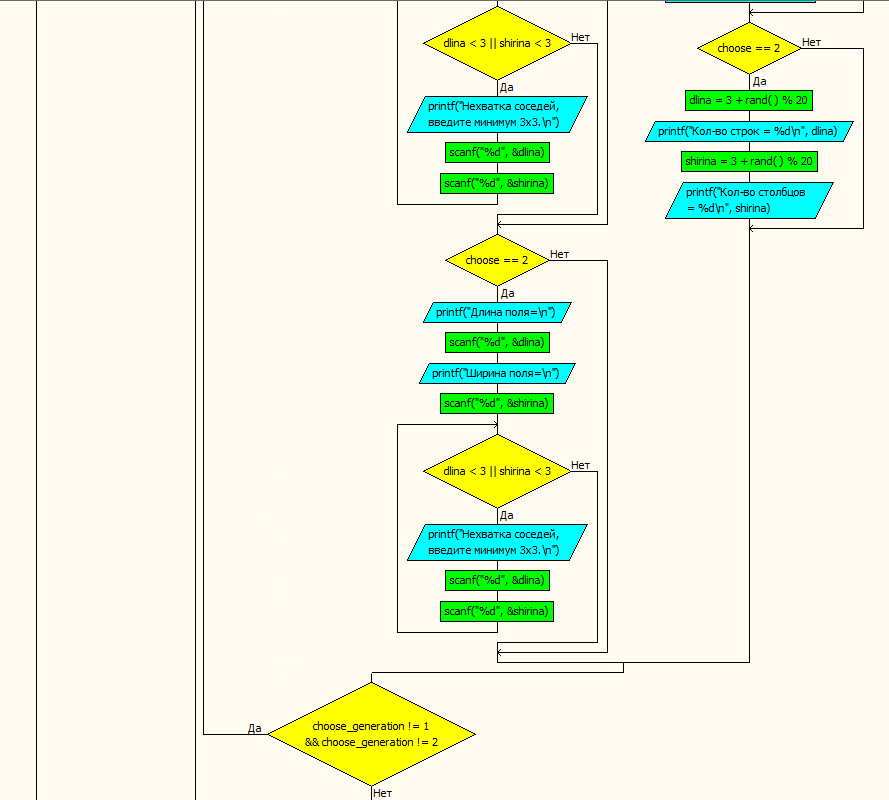
Продолжение ниже…



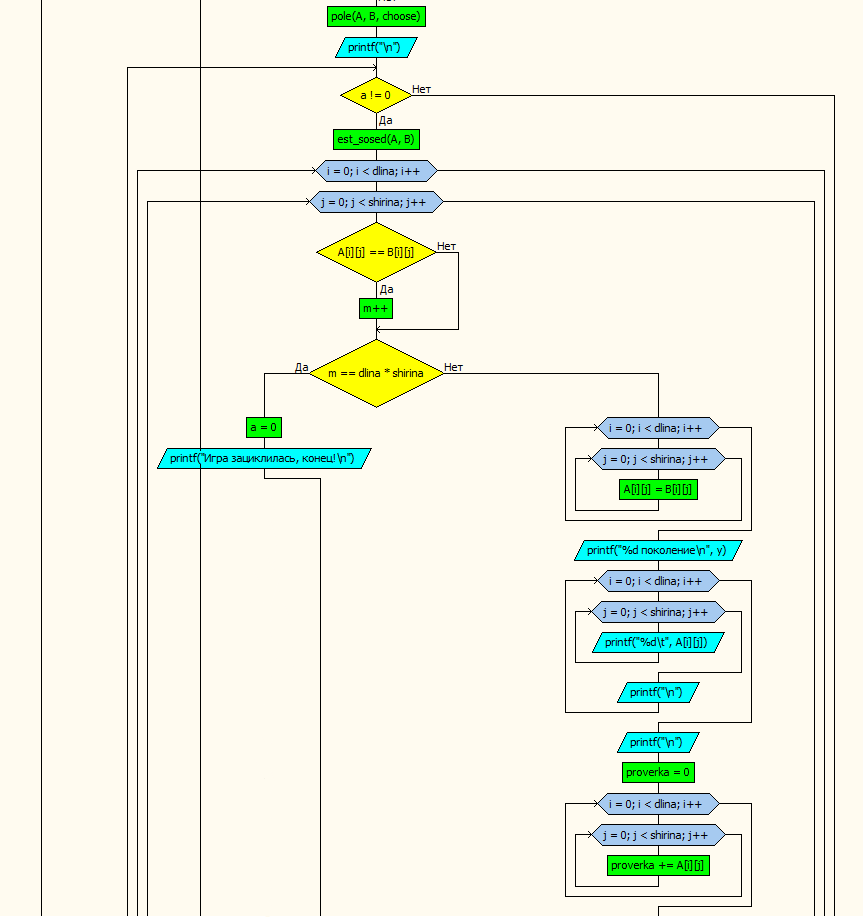
Продолжение ниже…



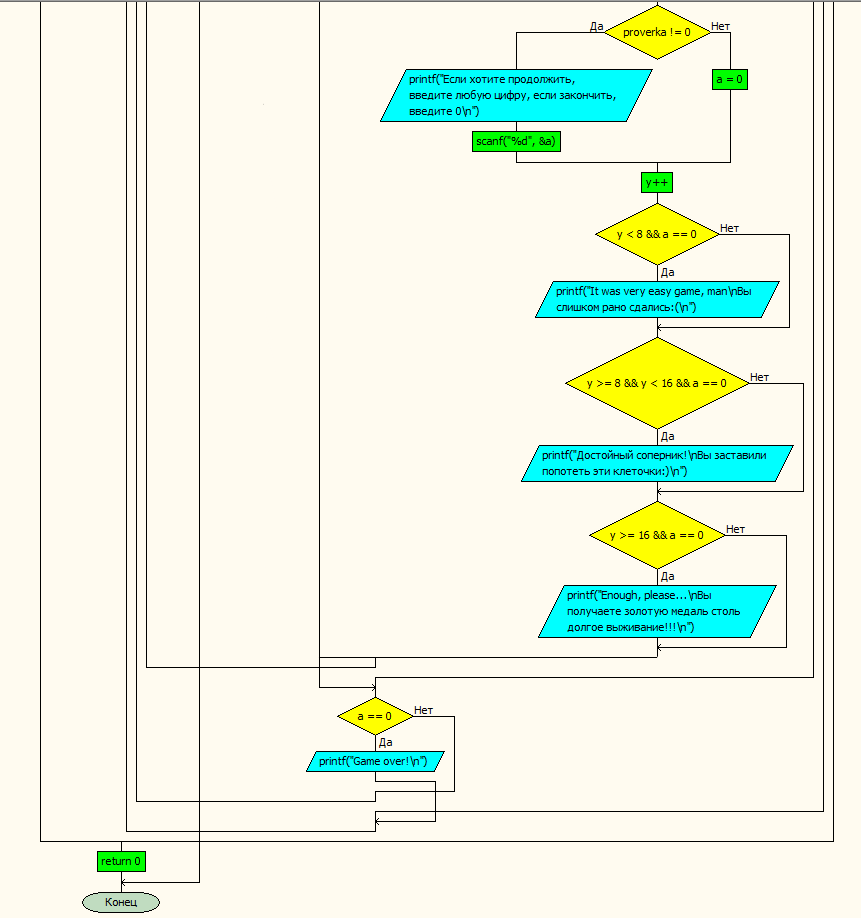
Продолжение ниже…



Продолжение ниже…



Продолжение ниже…



Данная функция начинается с “Главного меню”. Пользователь может либо почитать правила игры, либо начать сразу игру. Дальше игрок выбирает как хочет задать размерность и как задать самое первое поле. После этого происходит вызов функции pole, отвечающей за самое первое поколение, потом генерируется новое поколение с помощью функции est\_sosed. Дальше пользователь может выбрать либо продолжить игру, либо сразу же закончить. Вся игра закончится, как только на поле не останется живых клеток, либо, поколения начнут зацикливаться, то есть начнет появляться стабильная конфигурация.

Вывод: пользователь своими глазами увидел, как зарождается как таковая жизнь.

***4. Заключение***

Если вы только начинаете знакомиться со сферой IT и хотите заняться изучением программирования, язык Си – неплохое начало, этот язык даст вам толчок вперед. Лично на своем опыте я убедился, что Си – является высокоскоростным языком программирования. С его помощью в данной курсовой работе я смог написать игру, найти приближенные корни уравнений разными методами, посчитать сумму ряда, цепной дроби и сравнить их скорости сходимости.

***5. Список литературы***

[1] Б. Керниган, Д. Ритчи Язык программирования Си издание 3-е, исправленное. Санкт-Петербург 2003 г. – Режим доступа: <https://www.r-5.org/files/books/computers/languages/c/kr/Brian_Kernighan_Dennis_Ritchie-The_C_Programming_Language-RU.pdf>

[2] М. Э. Абрамян 1000 задач по программированию Часть 1. Ростов-на-Дону 2004 г. – Режим доступа: <https://docplayer.ru/26270004-M-e-abramyan-1000-zadach-po-programmirovaniyu-chast-i-skalyarnye-tipy-dannyh-upravlyayushchie-operatory-procedury-i-funkcii.html>

[3] М. Э. Абрамян 1000 задач по программированию Часть 2. Ростов-на-Дону 2004 г. – Режим доступа: <https://www.studmed.ru/view/abramyan-me-1000-zadach-po-programmirovaniyu-chast-ii_58d5082b0d7.html>

[4] М. Э. Абрамян 1000 задач по программированию Часть 3. Ростов-на-Дону 2004 г. – Режим доступа: <https://www.studmed.ru/view/abramyan-me-1000-zadach-po-programmirovaniyu-chast-iii_363bbeebb1b.html>

[5] [https://ru.wikipedia.org/wiki/Игра\_«Жизнь»](https://ru.wikipedia.org/wiki/Игра_)

[6] <https://prog-cpp.ru/digital-find/>

[7] А. В. Мезенцев Сборник задач по программированию с примерами. Иркутский Государственный университет, 2011 г. – Режим доступа: <https://vk.com/doc137181225_441506448?hash=bbc9337262a5f943b4&dl=eead91428fed264c40>

[8] Майк МакГрат Программирование на C для начинающих. Москва 2016 г. – Режим доступа: <https://codernet.ru/books/c_plus/programmirovanie_na_s_dlya_nachinayushhix_makgrat/>

[9] Грег Перри, Дин Миллер Программирование на C для начинающих 3-е издание. Москва 2015 г. – Режим доступа: <https://codernet.ru/books/c_plus/programmirovanie_na_c_dlya_nachinayushhix_3-e_izdanie/>

[10] Сборник задач по программированию. – Одесса: ОНАС им. А. С. Попова, 2011 г. – Режим доступа: <https://it.onat.edu.ua/docs/C++_Сборник%20задач%20по%20программированию.pdf.pdf>

[11] <https://ru.wikipedia.org/wiki/Ряд_(математика)>

[12] <https://ru.wikipedia.org/wiki/Непрерывная_дробь>