

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 №2

Выполнил: _____
студент группы ИВТ-11: Дроздов Г.А
Проверил: _____
доцент, к.т.н.: Артамонов Ю.Н.

Содержание

Введение	3
Глава I . РАЗРАБОТКА ЧИСЛЕННЫХ АЛГОРИТМОВ.....	4
1.1 Суммирование рядов и вычисление элементарных функций	4
1.2 Приближенные методы нахождения корней уравнения	9
1.2.1 Метод касательных.....	9
1.2.2 Метод секущих	15
Глава II . РАЗРАБОТКА ИГРОВОЙ ПРОГРАММЫ	21
Заключение.....	32
Список литературы.....	32
Ссылки	33

Введение

C замечательный язык. Разработанный первоначально одним человеком, Деннисом Ритчи, работающим в AT & T Bell Laboratories в Нью-Джерси, он расширился в использовании, и до сих пор он может быть одним из наиболее широко написанных компьютерных языков в мире. Успех C обусловлен рядом факторов, ни один из которых не является ключевым, но все они важны. Возможно, самым значительным из всех является то, что C был разработан настоящими практиками программирования и был разработан для практического повседневного использования, а не для демонстрации. Как и любой хорошо разработанный “инструмент”, он “легко ложится на руку и приятен в использовании”.

Многие другие языки не смогли обеспечить доступ к файлам и общие функции ввода-вывода, необходимые для приложений промышленного уровня.

Особенности языка C:

- Это надежный язык с богатым набором встроенных функций и операторов, которые можно использовать для написания любой сложной программы.
- Компилятор C сочетает в себе возможности языка ассемблера с функциями языка высокого уровня.
- Программы, написанные на C, эффективны и быстры. Это связано с разнообразием типов данных и мощных операторов.
- C обладает высокой переносимостью, это означает, что однажды написанные программы могут быть запущены на других машинах практически без изменений.
- Еще одна важная особенность C-программы - ее способность расширяться.
- Язык C является наиболее широко используемым языком в операционных системах и разработке встроенных систем сегодня.

Все эти особенности языка позволили выйти ему на мировой уровень. C все также используется в наше время.

В данной курсовой работе мы рассмотрим разработку методов численных алгоритмов и разработку игровых программ. Задания данной курсовой работы требует познания в таких дисциплинах как «Математический анализ», «Линейная алгебра».

Глава I . РАЗРАБОТКА ЧИСЛЕННЫХ АЛГОРИТМОВ

1.1 Суммирование рядов и вычисление элементарных функций

В данном разделе мы рассмотрим вычисления количества членов ряда и цепной дроби, с точностью необходимой для вычисления суммы рядов по формуле соотношения С.Рамануджанова.

Перед тем, как начать выполнение работы, нам необходимо будет узнать, что же такое числовой ряд и цепная дробь.

Числовой ряд — это сумма членов числовой последовательности вида. Ряд записывается как бесконечная сумма.

$$a_1 + a_2 + a_3 + \dots + a_n + \dots = \sum_{n=1}^{\infty} a_n$$

Где a_n - n -й или общий член ряда.

Также стоит рассмотреть понятие частичная сумма числового ряда.

Частичная сумма числового ряда — это сумма вида

$$S_n = a_1 + a_2 + \dots + a_n$$

Где n - натуральное число. S_n называют также n -ой частичной суммой числового ряда.

Рассмотрим пример. Четвертая частичная сумма ряда

$$\sum_{k=1}^{\infty} (16) \cdot \left(-\frac{1}{2}\right)^k$$

Есть $S_4 = 8 - 4 + 2 - 1 = 5$

После того, как мы рассмотрели понятия числового ряда, можно перейти к непрерывная дроби (или цепная дробь). Давайте разберем, что такое непрерывная дробь.

Непрерывная дробь — это конечное или бесконечное математическое выражение вида

$$[a_0; a_1, a_2, a_3, \dots] = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \dots}}}$$

Где a_0 — целое число а все остальные a_n — натуральные числа.

При этом числа $a_0, a_1, a_2, a_3, \dots$ называются **неполными частными** или **элементами** цепной дроби.

После того, как мы разобрались, мы можем перейти к решению задачи.

Дано: Соотношение С.Рамануджанова, имеющего вид:

$$\sqrt{\frac{e \cdot \pi}{2}}$$

Или же имеем иной вид:

$$\sqrt{\frac{e \cdot \pi}{2}} = 1 + \frac{1}{1 \cdot 3} + \frac{1}{1 \cdot 3 \cdot 5} + \frac{1}{1 \cdot 3 \cdot 5 \cdot 7} + \dots +$$
$$+ \frac{1}{1 + \frac{1}{1 + \frac{2}{1 + \frac{3}{1 + \frac{4}{1 + \dots}}}}}$$

Найти: вычислить, сколько членов ряда и цепной дроби нужно взять, чтобы достичь заданной точности.

Решение:

Для начала давайте рассмотрим Листинг программного кода.

```
#include <stdio.h>
#include <math.h>
#define e 2.7182818284590452
#define PI 3.1415926535897932384
double Ramanujan(int,int);
double Ramanujan(int countS, int countD)
{
int i;
```

```

double s = 0, a = 1, s1=0; // s = сумма ряда, s1 - сумма цепной дроби, a - член ряда
int n = 1;

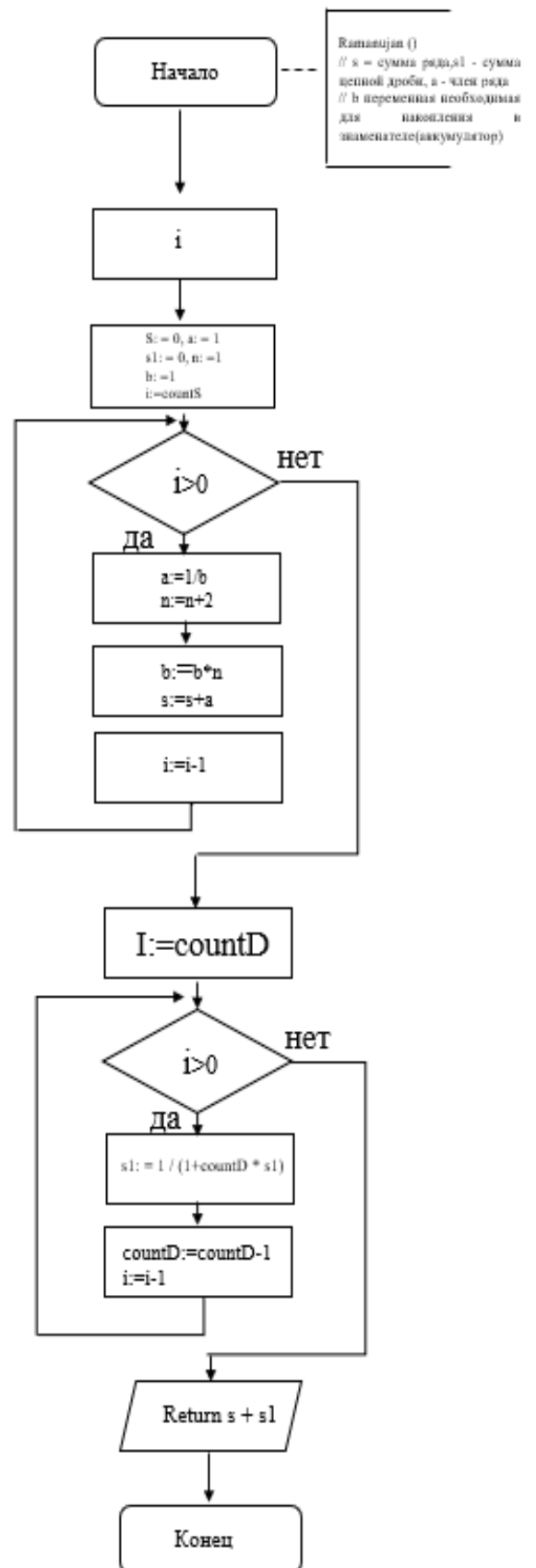
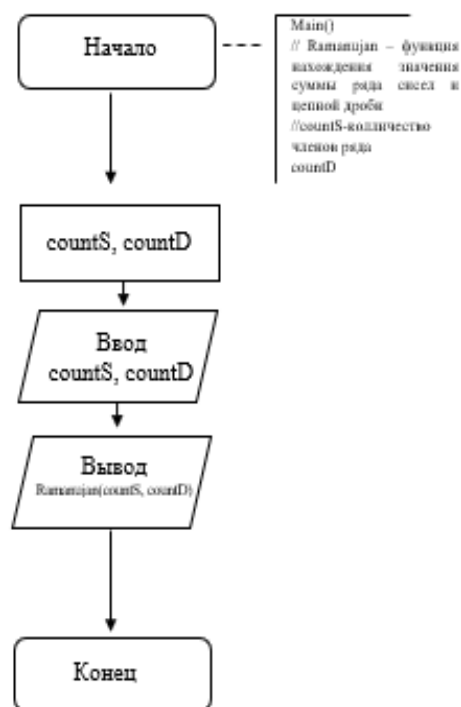
unsigned long int b=1; // b переменная необходимая для накопления в
знаменателе(аккумулятор)

for(i=countS;i>0;i--) // цикл нахождения суммы ряда
{
    a = 1 / (long double)b;
    n+=2;
    b*=n;
    s+=a;
}
for (i = countD; i > 0; i--, countD--)// цикл нахождения суммы цепной дроби
{
    s1 = 1 / (1 + countD * s1);
}
return s+s1;
}

int main()
{
    int countS,countD; // countS - счетчик членов суммы ряда, countD счетчик членов
цепной дроби
    printf("Введите количество членов ряда\n");
    scanf("%d",&countS);
    printf("Введите количество членов цепной дроби\n");
    scanf("%d", &countD);
    printf("Ответ : %.40lf \n", Ramanujan(countS,countD));
    return 0;
}

```

А также давайте рассмотрим блок схему.



Рассмотрим таблицу данных по данному коду.

Переменная	Значение
s	1.410686
s1	0.655680
countD	10000
countS	15

Полученный ответ	2.0664
------------------	--------

```

Введите количество членов ряда
15
Введите количество членов цепной дроби
10000
1.410686
0.655680
Ответ : 2.0664

```

Input:
$$\sqrt{\frac{e\pi}{2}}$$

Open code

All 2nd roots of $(e\pi)/2$:

More digits
Show trigonometric form

$$\sqrt{\frac{e\pi}{2}} e^{i0} \approx 2.0664 \text{ (real, principal root)}$$

Закключение: в данном разделе мы рассмотрели принцип работы рядов чисел и непрерывной дроби. Одним из самых замечательных результатов Рамануджана является эта формула, в соответствии с которой сумма простого числового ряда с цепной дробью в точности равна выражению, в котором присутствует произведение e на π .

1.2 Приближенные методы нахождения корней уравнения

Проблемы решения каких-либо уравнений часто возникают на практике. Например, в экономике, когда развивается бизнес, вам необходимо определить, при каких условиях доход будет равен определенной величине, в медицине, при изучении воздействия лекарств, необходимо узнать, в каком случае концентрация вещества будет доходить до необходимо уровня и т.д.

Решить уравнение — значит найти все его корни, то есть те значения x , которые обращают уравнение в тождество.

Для примера рассмотрим задачу решения уравнения

$$\sin(x) = \frac{1}{x}$$

Где угол x задан в градусах. Указанное уравнение можно переписать в виде

$$\sin\left(\frac{\pi}{180^\circ}x\right) - \frac{1}{x} = 0$$

Для графического отсечения корней достаточно построить график функции

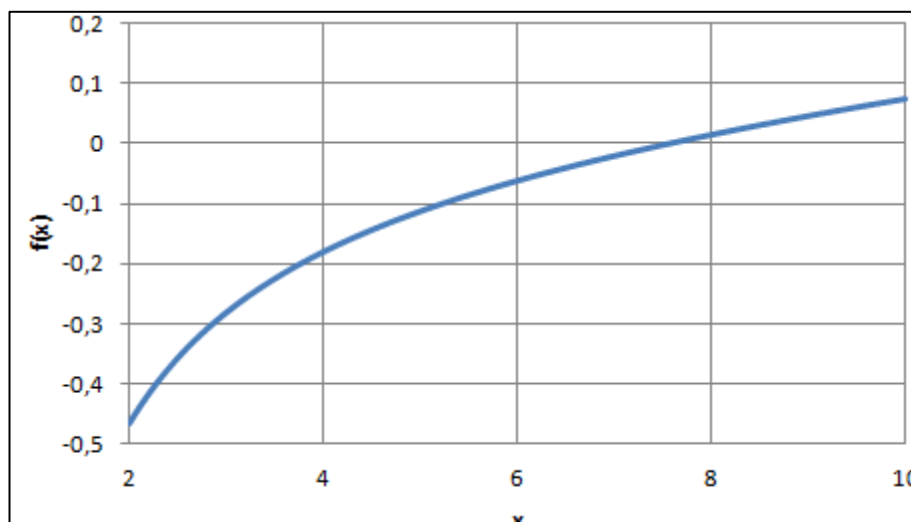


Рисунок 1.2.0.1

Из рисунка (1.2.0.1) видно, что корень нашего уравнения лежит в промежутке

$$x \in (6; 8)$$

1.2.1 Метод касательных

Метод касательных, который называют “Метод Ньютона“, предназначен для приближенного поиска нулей функции. Данный метод предложил английский физик, математик, механик, астроном Исаак Ньютон (1643—1727).

Рассмотрим уравнения, корни которых, нам необходимо найти.

Дано:

$$\sin(cx) - d = 0$$

Уравнение 1.2.1.1

$$x^4 + cx^3 - dx = 0$$

$$x^5 + cx^2 - d = 0$$

Найти: провести тестирование метода касательных. Сравнить число итераций этого метода при одном и том же значении точности. Придумать три уравнения и использовать данный метод для нахождения их корней.

Решение: для начала нам необходимо определиться с параметрами c и d для данных уравнений. Рассмотрим листинг программного кода, на примере уравнения (1.2.1.2), благодаря которому находятся корни уравнения.

```
#include <stdio.h>
#include <math.h>
#define eps 0.0000001
typedef double (*func)(double x, double c, double d); // задаем тип func
//Объявим прототипы функций
double decision(func,func,double,double,double); // прототип нахождения корня
функции
double fx(double, double, double); // прототип вычисляемой функции
double dfx(double, double, double); // прототип производной функции

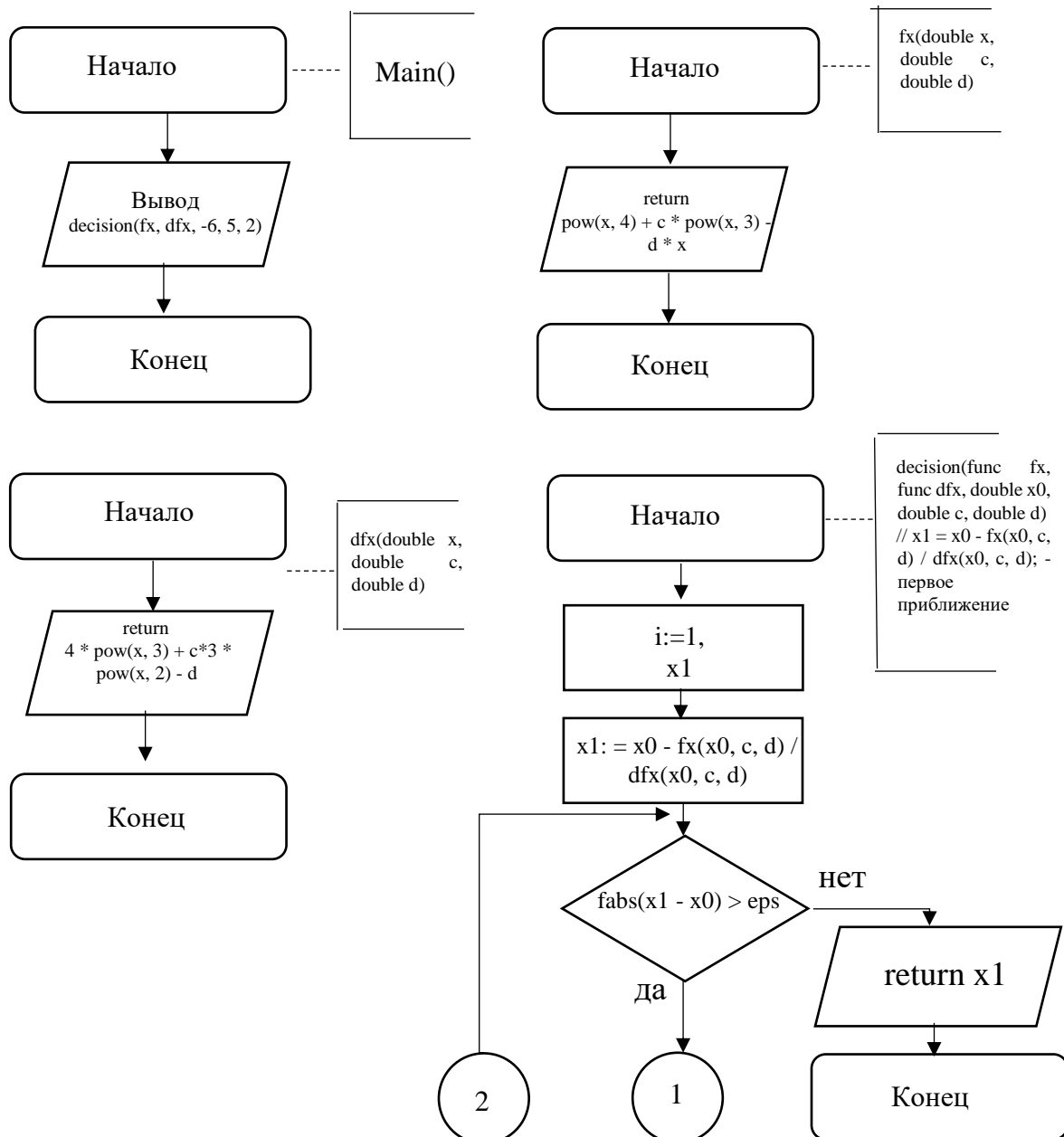
double fx(double x, double c, double d) // вычисляемая функция
{
    return pow(x,4) +c*pow(x,3)-d*x;
}
double dfx(double x, double c, double d) // производная функции
{
    return 4 * pow(x, 3) + c*3 * pow(x, 2) - d;
}
double decision(func fx, func dfx, double x0, double c, double d)
{
    double x1;
    x1 = x0 - fx(x0, c, d) / dfx(x0, c, d); // первое приближение
    while (fabs(x1 - x0) > eps) // пока не достигнута точность eps(0.0000001)
    {
        x0 = x1;
        x1 = x1 - fx(x1, c, d) / dfx(x1, c, d); // последующие приближения
    }
}
```

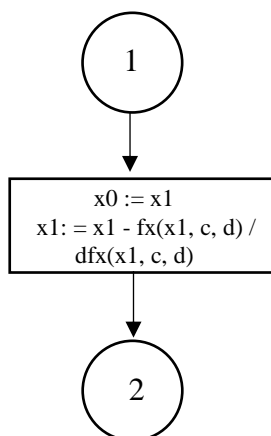
```

}
return x1;
}
int main()
{
printf("Ответ : %lf\n", decision(fx, dfx, -6, 5, 2)); // Вывод в консоль ответа
return 0;
}

```

А также рассмотрим блок-схему данного программного кода, на примере уравнения (1.2.1.2).





Рассмотрим таблицу данных для уравнения (1.2.1.2) по данному коду.

Переменные	Значения
c	5
d	2
x_0	-6
eps	0.0000001

А также рассмотрим итерации и корни, которые получились.

Итерация	Корень
1	-4.9864814067
2	-4.9200965463
3	-4.9172908848
4	-4.9172859931
5	-4.9172859930

Ответ	-4.9172859930
-------	---------------

Input interpretation:

solve

$x^4 + 5x^3 - 2x = 0$

using

Newton's method

starting at $x_0 = -6$

to machine precision

Result:

$x = -4.917285993093533$

Open code

Также, давайте, рассмотрим другие уравнения.

Начнем с уравнения (1.2.1.1).

Заполним таблицу данных.

Переменные	Значения
c	1
d	1
x_0	3.14159
eps	0.0000001

Рассмотрим итерации и корни, которые получились.

Итерация	Корень
1	1.848185
2	1.708594
3	1.639586
4	1.605178
5	1.587985
6	1.579391
7	1.575093
8	1.572945
9	1.571871
10	1.571333
11	1.571065
12	1.570931
13	1.570863
14	1.570830
15	1.570813
16	1.570805
17	1.570801
18	1.570798
19	1.570797
20	1.570797
21	1.570797
22	1.570796

Ответ	1.570796
-------	----------

Сравним результаты с “Wolframalpha”.

Input interpretation:

solve

$\sin(x) - 1 = 0$

using Newton's method

starting at $x_0 = 3.14159$

to machine precision

Open code

Result:

$x = 1.570796343187027$

Open code

И в заключении рассмотрим последнее уравнение (1.2.1.3).

Заполним таблицу данных.

Переменные	Значения
c	2
d	1
x_0	1
eps	0.0000001

Рассмотрим итерации и корни, которые получились.

Итерация	Корень
1	0.6776931
2	0.6613656
3	0.6609927
4	0.6609925

Ответ	0.6609925
-------	-----------

Сравним результаты с “Wolframalpha”.

Input interpretation:

solve

$x^5 + 2x^2 - 1 = 0$

using Newton's method

starting at $x_0 = 1$

to machine precision

Open code

Result:

$x = 0.6609925318901200$

Open code

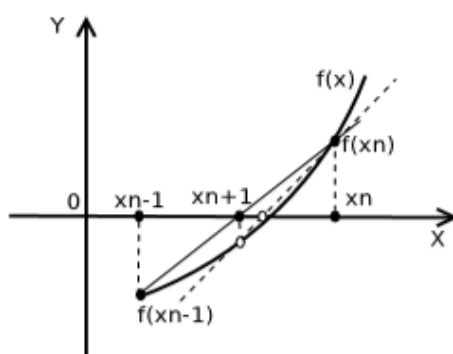
[Enlarge](#) |
 [Data](#) |
 [Customize](#) |
 [Plaintext](#) |
 [Interactive](#)

Заключение: в данном разделе мы рассмотрели метод касательных, а также научились находить корни с помощью данного метода. Исходя из результатов, можно сделать вывод о том, что тригонометрические функции требуют больших итераций для точных вычислений.

1.2.2 Метод секущих

Данный метод также называют модификацией метода касательных. Давайте рассмотрим определение метода секущих (метода хорд)

Метод хорд — итерационный численный метод приближённого нахождения корня уравнения.



Из рисунка 1.2.2.1 имеем формулу:

$$x_n = x_{n-1} - f(x_{n-1}) \frac{x_{n-1} - x_{n-2}}{f(x_{n-1}) - f(x_{n-2})} = \frac{x_{n-2}f(x_{n-1}) - x_{n-1}f(x_{n-2})}{f(x_{n-1}) - f(x_{n-2})}$$

Дано:

$$\sin(cx) - d = 0$$

Уравнение 1.2.2.1

$$x^4 + cx^3 - dx = 0$$

Уравнение 1.2.2.2

$$x^5 + cx^2 - d = 0$$

Уравнение 1.2.2.3

Найти: провести тестирование метода касательных. Сравнить число итераций этого метода при одном и том же значении точности. Придумать три уравнения и использовать данный метод для нахождения их корней.

Решение: чтобы более точно разобраться с данным методом рассмотрим листинг программного кода, на примере уравнения (1.2.2.1), благодаря которому находятся корни уравнения.

```
#include <stdio.h>
#include <math.h>
#define eps 0.0000001
```

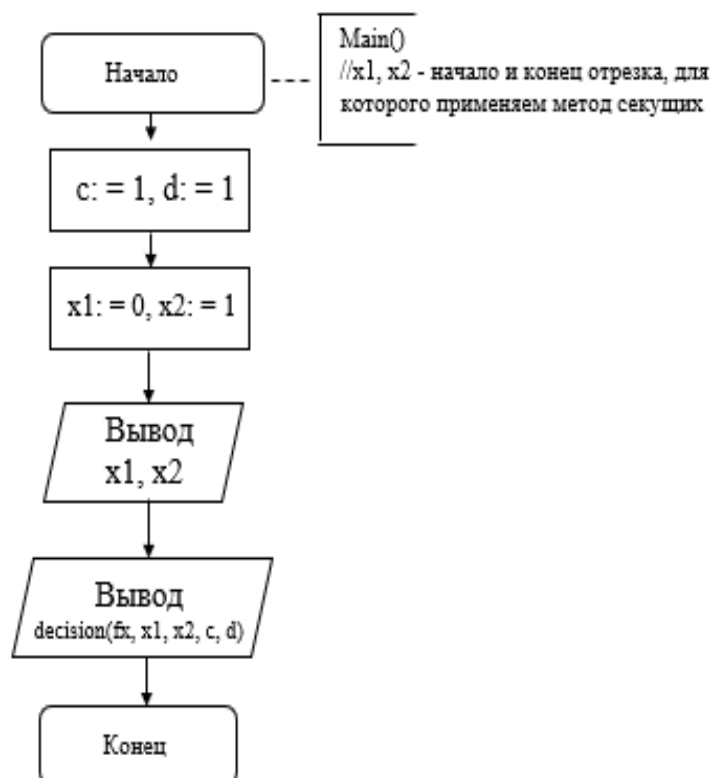
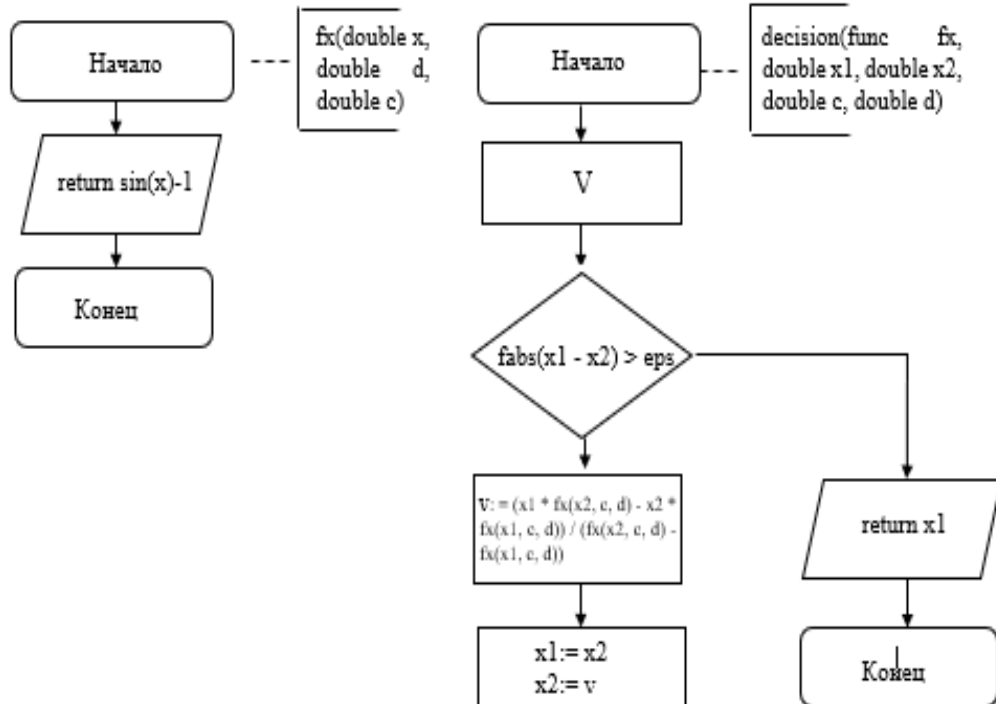
```

typedef double (*func)(double x, double c, double d); // задаем тип func
double fx(double, double, double); // прототип вычисляемой функции
double fx(double x, double d, double c) // вычисляемая функция
{
    return sin(c*x)-d;
}
double decision(func fx, double x1, double x2, double c, double d)
{
    double v;
    while (fabs(x1 - x2) > eps) // пока не достигнута точность eps(0.0000001)
    {
        v = (x1 * fx(x2, c, d) - x2 * fx(x1, c, d)) / (fx(x2, c, d) - fx(x1, c, d));
        x1 = x2;
        x2 = v;
    }
    return x1;
}
int main()
{
    double c = 1, d = 1;
    double x1 = 0, x2 = 1; //x1, x2 - начало и конец отрезка, для которого применяем
    метод секущих
    printf("Интервал:[%.2lf,%.2lf] \n", x1, x2); // Вывод в консоль интервала
    printf("Ответ:%f\n", decision(fx, x1, x2, c, d)); // Вывод в консоль ответа
    return 0;
}

```

И рассмотрим блок-схему данного кода.

Блок-схема представлена на следующей странице.



После того, как мы разобрались с кодом, можно перейти к выполнению задачи.

Рассмотрим таблицу данных для уравнения (1.2.2.1) по данному коду.

Переменные	Значения
c	1
d	1
x_2	2
x_1	0
eps	0.0000000001

Рассмотрим итерации и корни, которые получились.

Итерация	Корень
1	2.0000
2	2.1995
3	1.8200
4	1.7469
5	1.6736
6	1.6356
7	1.6105
8	1.5954
9	1.5860
10	1.5802
11	1.5766
12	1.5744
13	1.5730
14	1.5722
15	1.5716
16	1.5713
17	1.5711
18	1.5710
19	1.5709
20	1.5708

Ответ	1.5708
-------	--------

Сравним результаты с “Wolframalpha”.

Input interpretation:

solve

$\sin(x) - 1 = 0$

using
secant method

starting at $x_0 = 0$ and $x_1 = 2$
to machine precision

Open code

Result:

$x = 1.570868899057758$

Open code

Enlarge

Data

Customize

Plaintext

Interactive

Рассмотрим следующее уравнение (1.2.2.3), и построим таблицу данных.

Переменные	Значения
c	15
d	2
x_2	3
x_1	1
eps	0.000000001

Рассмотрим итерации и корни, которые получились.

Итерация	Корень
1	3.00000000
2	0.92265193
3	0.85747628
4	0.53945041
5	0.43118758
6	0.37704090
7	0.36561834
8	0.36457829
9	0.36456009
10	0.36456006

Ответ	0.36456006
-------	------------

Сравним результаты с “Wolframalpha”.

Input interpretation:

solve	$x^5 + 15x^2 - 2 = 0$	using secant method	starting at $x_0 = 1$ and $x_1 = 3$ to machine precision
-------	-----------------------	---------------------	---

Units »

Result:

$x = 0.3645600635661707$

Open code »

Units »

Рассмотрим последнее уравнение (1.2.2.2), и построим таблицу данных.

Переменные	Значения
c	5
d	2
x_2	1
x_1	-2
eps	0.0000000001

Рассмотрим итерации и корни, которые получились.


Итерация	Корень
1	1.0000000000
2	0.5000000000
3	0.536231884
4	0.620874609
5	0.593688682
6	0.597497327
7	0.597737748
8	0.597735187
9	0.597735188

Ответ	0.597735188
-------	-------------

Сравним результаты с “Wolframalpha”.


Input interpretation:

solve	$x^4 + 5x^3 - 2x = 0$	using secant method	starting at $x_0 = -2$ and $x_1 = 1$ to machine precision
-------	-----------------------	------------------------	--

[Open code](#) 

Result:

$x = 0.5977351881188707$

[Open code](#) 

Вывод по главе: В данной главе мы разобрали два метода нахождения корней уравнения. Из данной работы я сделал вывод, что оба метода более менее схожи по скорости работы. По моему мнению они не сильно отличаются на практике друг от друга.

Глава II. РАЗРАБОТКА ИГРОВОЙ ПРОГРАММЫ

В данной главе мы разберем создание игры на языке С. Написание игр является креативным делом, и у каждого одна и та же игра может быть написана по-разному. Мне попалась игра под названием “Морской бой”. Суть игры состоит в том, что игрок должен уничтожить флот компьютера.

Каждый флот состоит из:

- Четырёхпалубный – 1 корабль
- Трёхпалубный – 2 корабля
- Двухпалубный – 3 корабля
- Однопалубный – 4 корабля

После того как игрок разгромит все корабли, игра будет закончена и игрок победит компьютер. Но побеждать умеет не только игрок, если повезет, то может победить компьютер, все зависит от везения игрока.

Теперь мы можем перейти к коду.

Игра приветствует нас красивым меню.

[illegible]

В самом начале мы можем прочитать правила игры, а также начать игру.

Игра начинается с расстановки кораблей. У игрока есть 2 варианта расстановки:

- Вручную
- Автоматически

Вариант вручную требует особого терпения и знаний правил, а также внимательности. Пожалуй, начнем с ручного расстановки кораблей.

Начать, я думаю, стоит с правил игр. Их всего 3, но они являются очень важными.

- 1- Построение кораблей идет от начала координат в направлении вниз, если выбрано вертикальное расположение, и вправо, если выбрано горизонтальное расположение.

Пример:

```
Установка 4-х палубного корабля
0- вертикальное расположение 1-горизонтальное расположение
0
Введите координату начала корабля, так, чтобы корабли не уходили за край поля
4
4
  0 1 2 3 4 5 6 7 8 9
0 * * * * * * * * *
1 * * * * * * * * *
2 * * * * * * * * *
3 * * * * * * * * *
4 * * * * # * * * *
5 * * * * # * * * *
6 * * * * # * * * *
7 * * * * # * * * *
8 * * * * * * * * *
9 * * * * * * * * *
```

Игрок выбрал вертикальную ориентацию, затем он написал координаты начала корабля.

- 2- Если вы поставили корабль рядом с другим, то игра будет закончена и вам будет необходимо запустить игру заново. Проще сказать, что нельзя ставить корабли по периметру другого корабля.

Пример:

```
  0 1 2 3 4 5 6 7 8 9
0 * * * * * * * * *
1 * * * * * * * * *
2 * * * * * * * * *
3 * * * * * * * * *
4 * * * * # * * * *
5 * * * * # # * * *
6 * * * * # # * * *
7 * * * * # # * * *
8 * * * * * * * * *
9 * * * * * * * * *
```

Нельзя

```
  0 1 2 3 4 5 6 7 8 9
0 * * * * * * * * *
1 * * * * * * * * *
2 * * * * * * * * *
3 * * * * # * * * *
4 * * * * # * * * *
5 * * * * # * # * *
6 * * * * # * # * *
7 * * * * # * # * *
8 * * * * * * * * *
9 * * * * * * * * *
```

Можно

- 3- Если вы разгромили флот противника, разгромленная часть флота отображаться не будет. Например, если вы сломали флот, по периметру должны были быть расположены отметки в виде 'O'. Это является особенностью игры. Дабы игрок развивал свою внимательность и сильно не расслаблялся.

Пример:

	0	1	2	3	4	5	6	7	8	9
0	*	X	X	X	X	*	*	*	*	*
1	*	*	*	*	*	*	*	*	*	*
2	*	*	*	*	*	*	*	*	*	*
3	*	*	*	*	*	*	*	*	*	*
4	*	*	*	*	*	*	*	*	*	*
5	*	*	*	*	*	*	*	*	*	*
6	*	*	*	*	*	*	*	*	*	*
7	*	*	*	*	*	*	*	*	*	*
8	*	*	*	*	*	*	*	*	*	*
9	*	*	*	*	*	*	*	*	*	*

Как устроено

	0	1	2	3	4	5	6	7	8	9
0	O	X	X	X	X	O	*	*	*	*
1	O	O	O	O	O	O	*	*	*	*
2	*	*	*	*	*	*	*	*	*	*
3	*	*	*	*	*	*	*	*	*	*
4	*	*	*	*	*	*	*	*	*	*
5	*	*	*	*	*	*	*	*	*	*
6	*	*	*	*	*	*	*	*	*	*
7	*	*	*	*	*	*	*	*	*	*
8	*	*	*	*	*	*	*	*	*	*
9	*	*	*	*	*	*	*	*	*	*

Как должно

Далее перейдем к обозначениям.

- корабль

* - поле

x- попадание по кораблю

o- промах

После того, как игрок расставил корабли, начинается игра.

Игрок вводит координаты, если попадает, то на этом месте появляется 'x', игрок продолжает ход. Если промахивается, то ходить начинает компьютер. Игра продолжается до тех пор, пока не кончатся корабли из какой-либо сторон. Чтобы игрок не считал количество кораблей, в игру было введен счетчик, который отображает оставшиеся корабли.

```
Ваши корабли : 20
Корабли компьютера : 16
```

В конце, если мы победим, нас будет приветствовать надпись:

```
Поздравляю!! Вы выиграли.
```

Если проиграем, то:

```
К сожаления, вы проиграли...
```

Теперь можно рассмотреть детально код нашей программы, а он вышел весьма большой, для такой простой игры, как кажется.

Начнем с функции Welcome.

На следующей странице показан листинг данной функции.

[illegible]

Функция до жути проста. Мы просто выводим строки, которые образуют картинку. Также можно заметить “\x1b[4m\x1b[31m” – это ansicolor.

Например: “\x1b[4m” отвечает за то, чтобы текст подчеркивался снизу, а “\x1b[31m” чтобы текст становился красным. Далее в коде игры можно будет увидеть такие “ansicolor” как \x1b[33m – желтый, \x1b[94m – синий , \x1b[35m – фиолетовый.

Не будем особо заострять внимание на данную функцию. Перейдем к следующей.

А следующая функция, которая также проста в понимании это `showfield`.

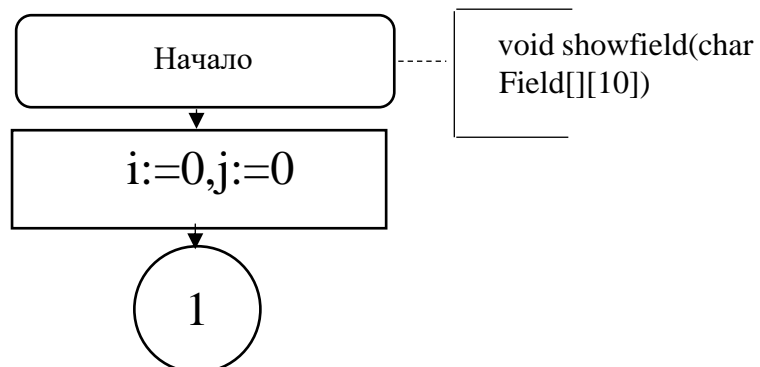
Благодаря этой функции, мы можем увидеть наше поле либо поле противника, если разработчику это будет необходимо. Давайте посмотрим листинг данной функции, который будет располагаться на следующей странице, и блок-схему.

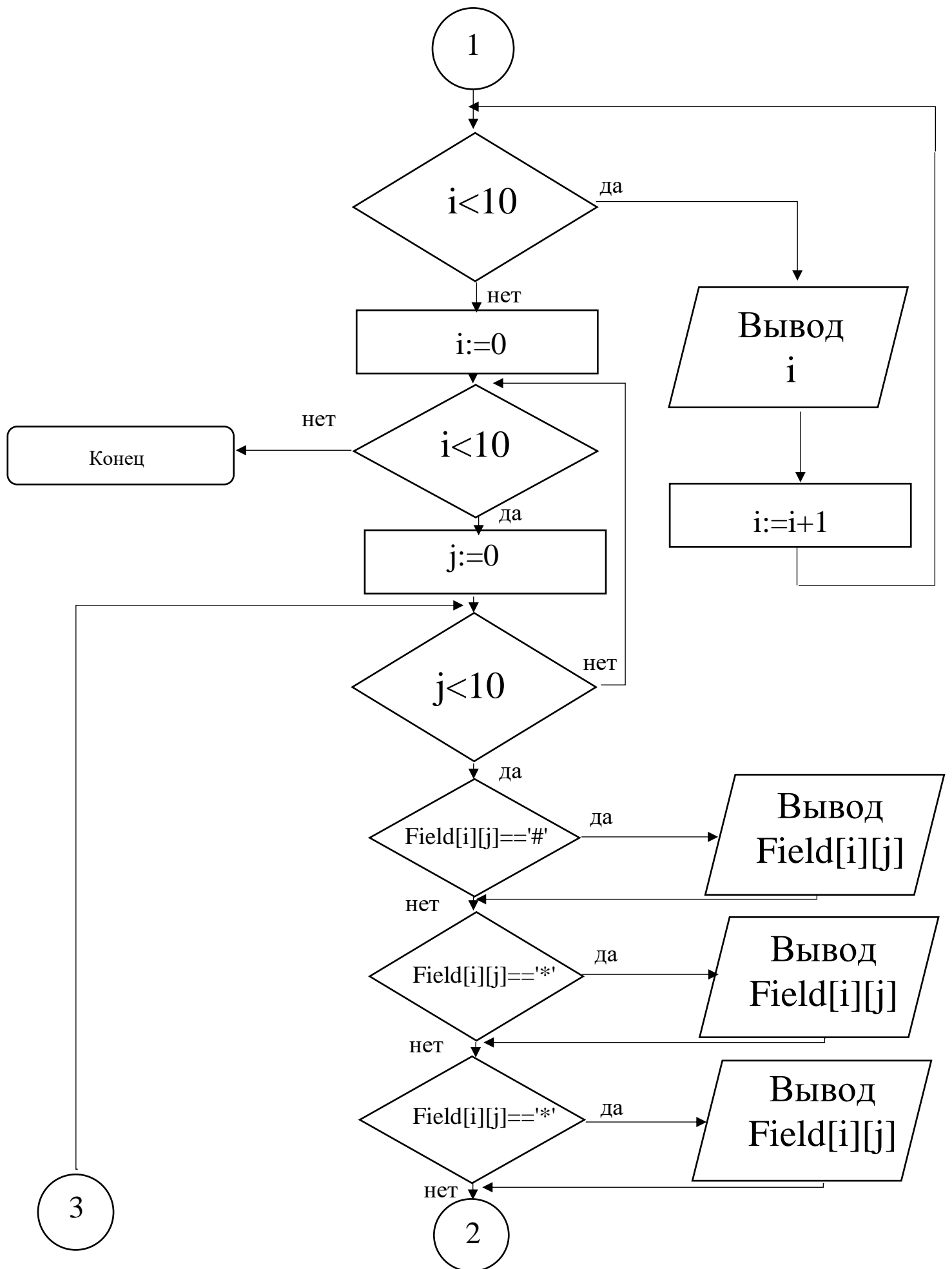

```

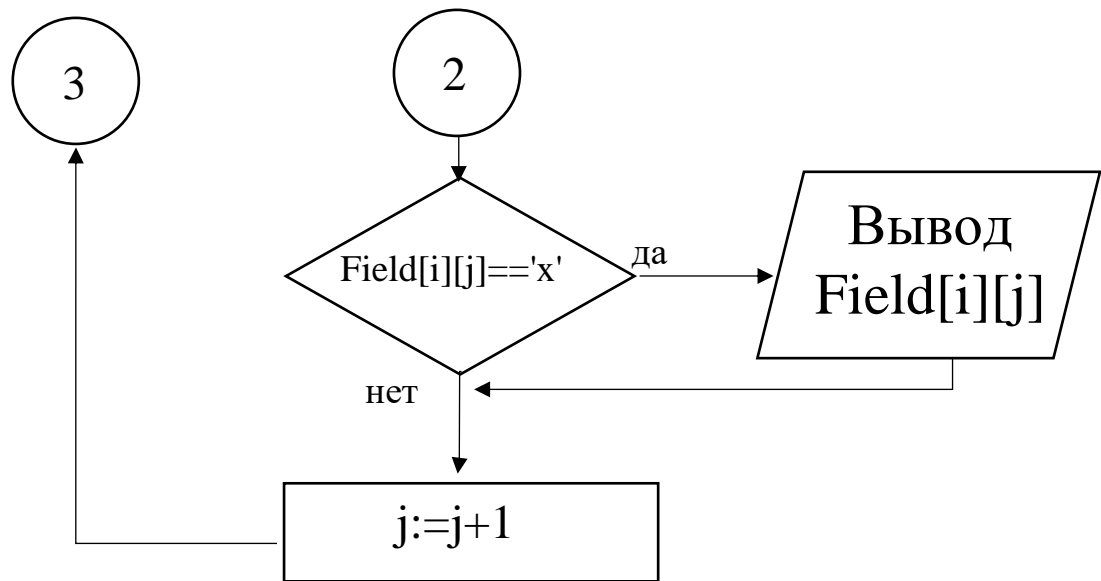
void showfield(char Field[][10]) // ВЫВОД массива полей на экран
{
    int i,j;
    printf(" ");
    for (i = 0; i < 10; i++) printf("%2d",i);
    printf("\n");
    for (i = 0; i < 10; i++)
    {
        printf("%d",i);
        for (j = 0; j < 10; j++)
        {
            if(Field[i][j]=='#')
                printf("\x1b[33m%2c\x1b[0m", Field[i][j]);
            if(Field[i][j]=='*')
                printf("\x1b[94m%2c\x1b[0m", Field[i][j]);
            if (Field[i][j] == 'O')
                printf("\x1b[35m%2c\x1b[0m", Field[i][j]);
            if (Field[i][j] == 'X')
                printf("\x1b[31m%2c\x1b[0m", Field[i][j]);
        }
        printf("\n");
    }
}

```

Рассмотрим блок-схему данной функции.







Данная блок-схема работает по принципу «Цикл в цикле», что не является чем-то новым.

Также я хочу рассказать про автоматическую установки кораблей. Этим занимается функция `placement_random`. Рассмотрим листинг фрагмента функции.

```

for(i=0;i<2;i++){
countcheck = 0;
size = 3;
check = rand() % 2; // случайный выбор расположения корабля(верт. или гор.)
x = rand() % 8;
y = rand() % 8;

//-----верт расположение-----
while (countcheck != 3 && check == 0)
{
if (check == 0 && x < 8) //Проверка на гор. расположение и выход за пределы массива(поля)
{
if (Field[x][y] != '#' && Field[x + 1][y] != '#' && Field[x + 2][y] != '#' &&
Field[x - 1][y - 1] != '#' && Field[x][y - 1] != '#' && Field[x + 1][y - 1] != '#' && Field[x + 2][y - 1]
!= '#' && Field[x + 3][y - 1] != '#' &&
Field[x - 1][y + 1] != '#' && Field[x][y + 1] != '#' && Field[x + 1][y + 1] != '#' && Field[x + 2][y +
1] != '#' && Field[x + 3][y + 1] != '#' &&
Field[x - 1][y] != '#' && Field[x + 3][y] != '#')
{

```

```

    for (; countcheck < size; x++, countcheck++)
        Field[x][y] = '#';
    }
else
{
    x = rand() % 8;
    y = rand() % 8;
    countcheck = 0;
}
}

//-----гор. расположение-----
while (countcheck != 3 && check == 1)
{
    if (check == 1 && y < 8) //Проверка на гор. расположение и выход за пределы массива(поля)
    {
        if (Field[x][y] != '#' && Field[x][y + 1] != '#' && Field[x][y + 2] != '#' &&
            Field[x - 1][y - 1] != '#' && Field[x - 1][y] != '#' && Field[x - 1][y + 1] != '#' && Field[x - 1][y + 2]
            != '#' && Field[x - 1][y + 3] != '#' &&
            Field[x + 1][y - 1] != '#' && Field[x + 1][y] != '#' && Field[x + 1][y + 1] != '#' && Field[x + 1][y +
            2] != '#' && Field[x + 1][y + 3] != '#' &&
            Field[x][y - 1] != '#' && Field[x][y + 3] != '#')
        {
            for (; countcheck < size; y++, countcheck++)
                Field[x][y] = '#';
        }
    }
else
{
    x = rand() % 8;
    y = rand() % 8;
    countcheck=0;
}
}
}
}

```

В данном фрагменте компьютер выбирает случайную ориентацию корабля, а также его координату. Самым интересным в данном коде является условие if. Оно проверяет наличие каких-либо кораблей вокруг корабля. Для трёхпалубного

корабля этих условий аж 10 штук и при этом только для одной ориентации, а для другой еще 10.

Самой главной функцией является Game.В ней происходит весь игровой процесс. Для начала необходимо ознакомиться с листингом данной функции.

```
int Game(func showfield, char MyField[][10], char Enemy[][10], char
HiddenEnemy[][10])
{
    srand(time(NULL));
    int x, y; //x,y координаты
    // Массив второго поля(Чтобы игрок видел только попадания по кораблям)
    // int NextRound=1; // Переменная для проверка попадания
    int Round_by_player = 1; // Ход игрока
    int Round_by_Enemy = 0; // Ход компьютера
    int Enemy_Ships=20; // Количество кораблей противника
    int MyField_Ships = 0; // Количество кораблей игрока
    while(Enemy_Ships != 0 || MyField_Ships != 0)
    {
        if (MyField_Ships == 0)
        {
            return 1;
        }
        else if (Enemy_Ships == 0)
        {
            printf("Число 2 \n");
            return 2;
        }
        printf("===== Ваш ход
===== \n");
        if(Round_by_player )
        {
            printf("Введите координаты x,y\n");
            scanf("%d%d", &x, &y);
            while ((x < 0 || x > 9) || (y < 0 || y > 9))
            {
```

```

    printf("Ошибка , введите заново координаты(от 0 до 9),так,чтобы корабли не уходили
за край поля \n");
    scanf("%d%d", &x, &y);
}
if(Enemy[x][y]=='#')
{
    printf("Вы попали,продолжайте ход!!\n");
    if (HiddenEnemy[x][y] == '*'){
        Enemy_Ships--;
        printf("Ваши корабли : %d\n", MyField_Ships);
        printf("Корабли компьютера : %d\n", Enemy_Ships);
        HiddenEnemy[x][y] = 'X';
        showfieldAll(showfield, MyField, HiddenEnemy);
        continue;
    }

    if (HiddenEnemy[x][y] == 'X')
    {
        continue;
        showfieldAll(showfield, MyField, HiddenEnemy);
    }
}
else if (Enemy [x][y] == '*')
{
    HiddenEnemy[x][y]='O';
    printf("Ход компьютера\n");
    Round_by_player=0;
    Round_by_Enemy=1;
}
}
if(Round_by_Enemy)
{
    x = rand() % 10;
    y = rand() % 10;

```

```

if (MyField[x][y] == '#')
{
    MyField[x][y] = 'X';
    MyField_Ships--;
}
else if(MyField[x][y]== '*'){
    MyField[x][y] = 'O';
    Round_by_player = 1;
    Round_by_Enemy = 0;
}
}
printf("Ваши корабли : %d\n", MyField_Ships);
printf("Корабли компьютера : %d\n", Enemy_Ships);
showfieldAll(showfield, MyField, HiddenEnemy);
}
}

```

Мы объявляем переменные необходимые для игры. Самыми важными являются Enemy_Ships и MyField_Ships. Благодаря им производится проверка кораблей двух сторон. В случае, когда кораблей становится 0, программа возвращает определенное значение. Например, если проиграл компьютер, программа возвращает двойку, а если игрок, то единицу. Также стоит добавить, что компьютер случайно выбирает координату от 0 до 9. Если компьютер попал в точку, в которую уже стрелял, то он начинает ходить заново. Данная особенность позволяет компьютеру не пропускать ход.

После разбора кода игры, можно ознакомиться со скриншотами игры.

```

===== Ваш ход =====
Ваши корабли : 19
Корабли компьютера : 19
Ваши корабли
 0 1 2 3 4 5 6 7 8 9
0 * * * # # # * * *
1 * # * * * * * * *
2 * # * * * # 0 * *
3 * * * * # * * * *
4 * * # * * * * * *
5 0 * * * * 0 * * *
6 * * # # * * X # *
7 * * * * * * * * *
8 * # * # * * * * #
9 * # * 0 0 * * * *
Вражеские корабли
 0 1 2 3 4 5 6 7 8 9
0 * 0 0 * * * X * *
1 * * * * * * * * *
2 * * * * * * * * *
3 * * * * 0 * * * *
4 * * * * * * * * *
5 * * * * 0 * * * *
6 * * * * * * * * *
7 * * * * * * * 0 *
8 * * * * * * * * *
9 * * * * * * * * *

```

Рис 2.1 Игровой процесс.

Ссылки

Ссылка на Github: <https://github.com/griha01/programming1>