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 №12

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

Студент группы ИВТ-11 Кучук Д.А.

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

Доцент, к.т.н Артамонов Ю.Н.

Котельники – 2020

**Содержание**

[**Введение 3**](#_Toc41512694)

[**Глава I. Разработка численных алгоритмов 4**](#_Toc41512695)

[**Глава II. Разработка игровой программы 22**](#_Toc41512696)

[**Заключение 34**](#_Toc41512697)

[**Список литературы 65**](#_Toc41512698)

# **Введение**

Си (англ. C) — компилируемый статически типизированный язык программирования общего назначения, разработанный в 1969—1973 годах сотрудниками Bell Labs Деннисом Ритчи и Кеном Томпсон.

По одной легенд разработчикам языка нравилась игра, которую они запускали на главном сервере компании. Позже они захотели перенести её на офисный компьютер, но как оказалось, на нем не было операционной системы. Это сподвигло их написать собственную ОС на ассемблере, но её было невозможно перенести на другой компьютер. Деннис и Кен решили переписать ОС на языке высокого уровня, но не смогли найти подходящий. Поэтому было принято решение создать свой язык.

C приобрел свою популярность благодаря его использованию в ОС UNIX, которая стала популярной. Из-за широкого распространения UNIX и объему написанного кода, перенос операционной системы на другой язык программирования просто невозможен. Это обеспечивает существование языка C как минимум до конца существования UNIX. Но применение языка не ограничивается разработкой ОС, на нем разрабатываются прикладное ПО. Так же, благодаря конструкции языка, его схожести с языками низкого уровня возможностей C хватает даже для разработки ПО суперкомпьютеров!

Достоинства языка C:

1. Эффективность. Структура C позволяет лучше всего использовать возможности компьютера. Программы, реализованные на данном языке, отличаются своим быстродействием.
2. Кроссплатформенность. Программу, написанную на языке C легко можно перенести на другую платформу, при этом внося незначительные изменения или не внося их вовсе.
3. Мощность и гибкость. C нашел применение в различных областях программирования.

В данной курсовой работе мы рассмотрим разработку численных алгоритмов и игровой программы.

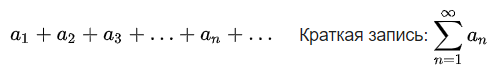
Глава I. Разработка численных алгоритмов

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

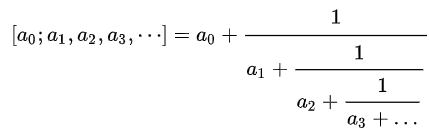
В данном разделе рассматривается алгоритм вычисления числа *e* в виде ряда и бесконечной дроби.

Для начала узнаем, что такое числовой ряд и бесконечная дробь.

**Числовой ряд** — одно из центральных понятий математического анализа. В простейшем случае ряд записывается как бесконечная сумма чисел.



**Непрерывная дробь** (или **цепная дробь**) – это конечное или математическое выражение вида:

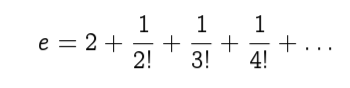


где *an*{\displaystyle a\_{0}} есть целое число, а все остальные *an* {\displaystyle a\_{n}} — натуральные числа (положительные целые).

Теперь, как мы узнали, что такое числовой ряд и непрерывная дробь, можно перейти к решению задач.

**Задача 1.1**

Дано: Представление числа *e* в виде ряда:

****

Найти: кол-во членов ряда, используемое для вычисления заданной точности.

Решение:

Рассмотрим листинг кода программы:

Входные данные: кол-во знаков после запятой.

Выходные данные: число *e* с заданным кол-вом знаков после запятой.

#include <stdio.h>//подключаем стандартную библиотеку ввода/вывода;

#include <math.h> //подключаем математическую библиотеку;

double calcE(double);//прототип функции;

int main()

{

int eps; //кол-во знаков после ,;

double res; //переменная для хранения результатов функции;

printf("Введите кол-во знаков после запятой: ");//узнаем кол-во

scanf("%d",&eps); //знаков;

res = calcE(eps); //вызываем функции и сохраняем возвращенно значение;

printf("res = %.10f", res); //выводим результат;

return 0; //выходим из программы;

}

double calcE(double eps)

{

if (!eps) //если запрошенно 0 знаков;

return 2;

double e = 2, z; //е - число e, z - значение члена ряда;

int x = 2, n = x; //x, n - для вычисления факториала;

eps = pow(10, -eps); //вычисляем точность;

do

{

for (int i = x - 1; i > 0; i--)//вычисляем факториал;

x \*= i;

z = (double)1 / x; //получаем значение члена ряда;

e += z; //считаем значение e;

n++; //счетчик членов ряда;

x = n; //сбрасываем значение факториала;

} while (z > eps); //повторяем пока не получим заданную точность;

printf("кол-во членов: %d\n", n);

return e; //возвращаем число e;

}

Блок-схема, построенная по данному коду, выглядит так:

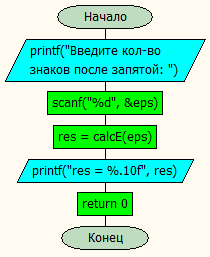
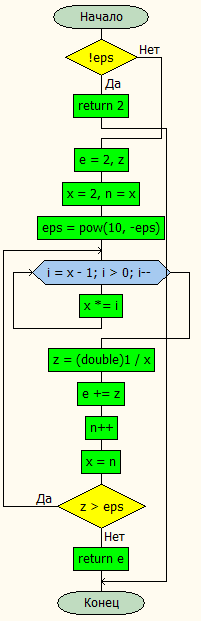
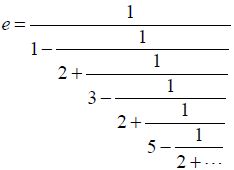
 

Таблица данных составленная по исходному коду.

|  |  |
| --- | --- |
| Точность | 0,00001 |
| Количество членов ряда | 10 |
| Значение | 2.7182815256 |

**Задача 1.2**

Дано: представление числа *e* в виде бесконечной(цепной) дроби:



Найти: количество правильных цифр после запятой.

Решение:

Для начала мы рассмотрим листинг кода:

Входные данные: количество членов дроби.

Выходные данные: приближенное число *e*.

Код программы выглядит так:

#include <stdio.h> //подключаем стандартную библиотеку ввода/вывода;

#include <math.h> //подключаем математическую библиотеку;

double calcE(double); //прототип функции;

double calc(int, double); //прототип функции;

int main(int argc, char const \*argv[])

{

int eps; //количество членов дроби;

double res; //переменная для хранения результатов функции;

printf("Введите кол-во членов дроби: "); //узнаем кол-во

scanf("%d",&eps); //знаков;

res = calcE(eps); //вызываем функцию и сохраняем возвращенно значение;

printf("res = %.10f", res); //вывод результатa;

return 0; //выходим из программы;

}

double calcE(double eps)

{

if (!eps) //если запрошенно 0 знаков;

return 2; //

double e = 1; //переменная хранящая значение e

double z;

int count = 1; //переменная-счетчик

e += 1 / calc(count, eps); //вычисляем число e

return e; //возвращаем число e

}

double calc(int count, double eps)

{

if (count > eps) //если значение счетчика больше или равен точности

return count; //возвращаем значение счетчика

if (count % 2) //если значение счетчика

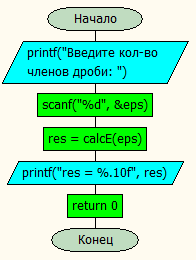
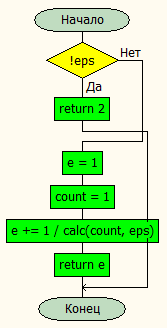
return count - 1 / calc(count + 1, eps);

else

return 2 + 1 / calc(count + 1, eps);

}

Блок-схема данного кода приведена ниже:

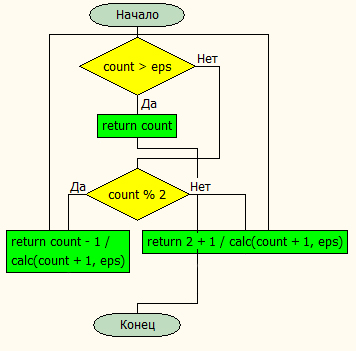


Таблица данных для этого кода выглядит так:

|  |  |
| --- | --- |
| Точность | 0,001‬ |
| Количество членов дроби | 5 |
| Значение | 2.7180616740 |

Заключение: в данном задании мы рассмотрели принцип работы ряда и бесконечной дроби.

**1.2 Приближенные методы нахождения корней уравнения**

Решение уравнений используются во многих областях науки.

**Решение уравнения** — это задача по нахождению таких значений аргументов (чисел, функций, наборов и т. д.), при которых выполняется равенство (выражения слева и справа от знака равенства становятся эквивалентными). Значения неизвестных переменных, при которых это равенство достигается, называются решениями или корнями данного уравнения. Решить уравнение означает найти множество всех его решений (корней) или доказать, что корней нет вовсе.

**1.2.1 Метод деления отрезка пополам**

**Метод деления отрезка пополам (метод бисекции)** — самый простой численный метод для решения нелинейных уравнений вида *f*(*x*)=0. Предполагается, что функция *f*(*x*) непрерывна.

Дано:

Придуманные уравнения:

Найти: корни данных уравнений с помощью метода деления отрезка пополам.

Решение:

Рассмотрим листинг программного кода:

#include <stdio.h>

#include <math.h>

#define EPS 0.0001

#define PI 3.141592653589793238462643383279

double bisectMethod(double, double);

double fx(double, double, double);

int main(int argc, char const \*argv[])

{

double left;

double right;

printf("Введите границы: ");

scanf("%lf %lf", &left, &right);

printf("Ответ : %f\n", bisectMethod(left, right));

return 0;

}

double bisectMethod(double left, double right)

{

double middle, s\_left, s\_midle;

double fx\_left = fx(left, 4, 2);

double fx\_right = fx(right, 4, 2);

int count = 1000;

if (fx\_left \* fx\_right > 0)

{

printf("Ведены неверные границы!");

return 0;

}

if (!fx\_left)

return left;

if (!fx\_right)

return right;

while (right - left > EPS && count)

{

count--;

middle = (left + right) / 2;

s\_left = copysign( 1, fx(left, 4, 2) );

s\_midle = copysign( 1, fx(middle, 4, 2) );

if ( s\_left != s\_midle )

right = middle;

else

left = middle;

};

printf("Число итераций: %d\n", 1000 - count);

return middle;

}

double fx(double x, double c, double d)

{

return sin(c \* x \* 180 / PI) - d;

// return exp(c \* x) - d;

// return pow(x, 5) + c \* pow(x, 2) - d;

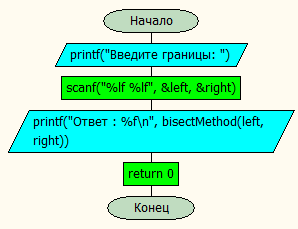
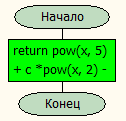
// return pow(x, 3) - c \* d;

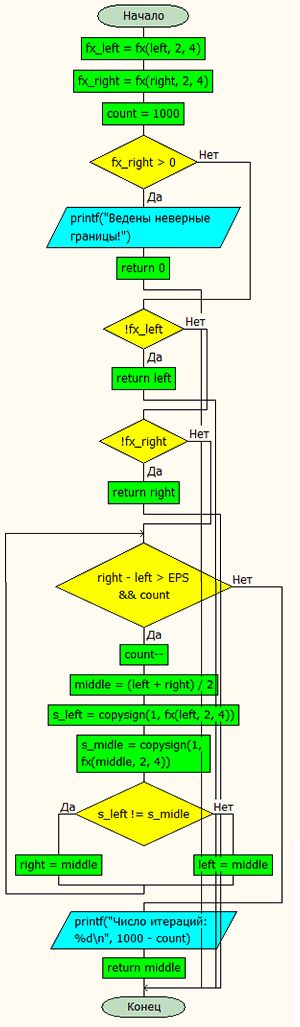
// return c \* pow(x, c) - d \* pow(x, d);

// return pow(x, 3) + pow(x, c) - d \* x - c;

}

И составленную по нему блок-схему:

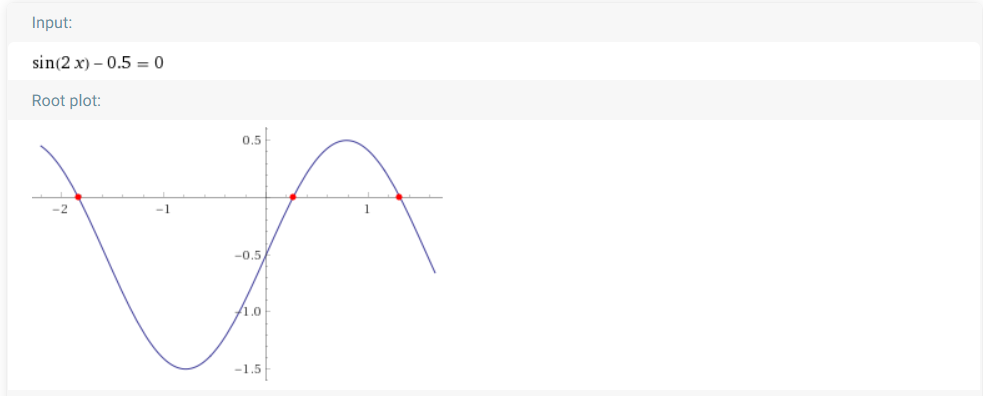


Для уравнения получается такая таблица:

|  |  |
| --- | --- |
| Переменная | Значение |
| c | 2 |
| d | 0,5 |
| x0 | -3 |
| EPS | 0,0001 |

|  |  |
| --- | --- |
| Количество итераций | 17 |
| Ответ | 1.229149 |

Сравним ответ, полученный нами с “Wolframalpha”.



Для уравнения можно составить такую таблицу:

|  |  |
| --- | --- |
| Переменная | Значение |
| c | 4 |
| d | 2 |
| x0 | -3 |
| EPS | 0,0001 |

|  |  |
| --- | --- |
| Количество итераций | 16 |
| Ответ | 0.173279 |

Сравниваем результат с “Wolframalpha”.

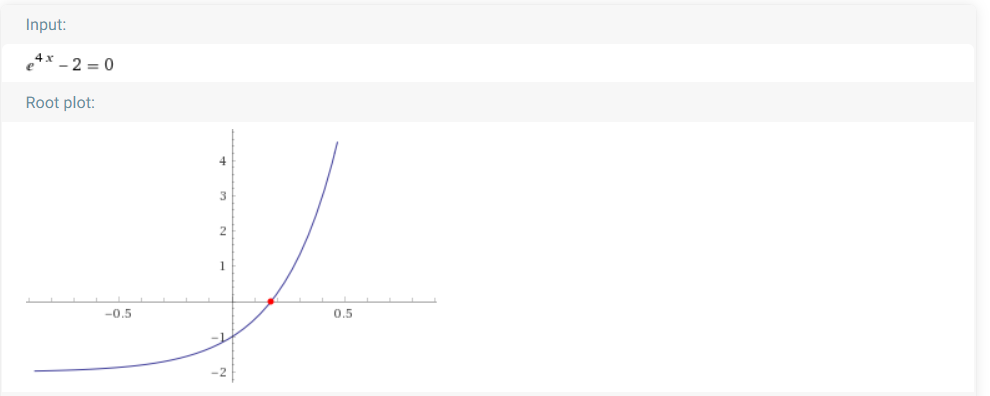
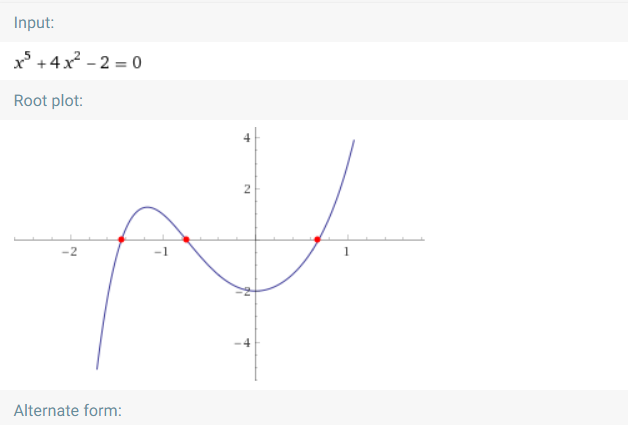


Таблица значений для уравнения выглядит так:

|  |  |
| --- | --- |
| Переменная | Значение |
| c | 4 |
| d | 2 |
| x0 | -3 |
| EPS | 0,0001 |

|  |  |
| --- | --- |
| Количество итераций | 17 |
| Ответ | 0.680779 |

Ответ, полученный с помощью калькулятора “Wolframalpha”.

****

****

Для уравнения получается такая таблица:

|  |  |
| --- | --- |
| Переменная | Значение |
| c | 4 |
| d | 2 |
| x0 | -3 |
| EPS | 0,0001 |

|  |  |
| --- | --- |
| Количество итераций | 17 |
| Ответ | 2.0000 |

Сравним ответ, полученный нами с “Wolframalpha”.

****

Для уравнения можно составить такую таблицу:

|  |  |
| --- | --- |
| Переменная | Значение |
| c | 1 |
| d | 3 |
| x0 | -3 |
| EPS | 0,0001 |

|  |  |
| --- | --- |
| Количество итераций | 17 |
| Ответ | 0.577385 |

Сравниваем результат с “Wolframalpha”.

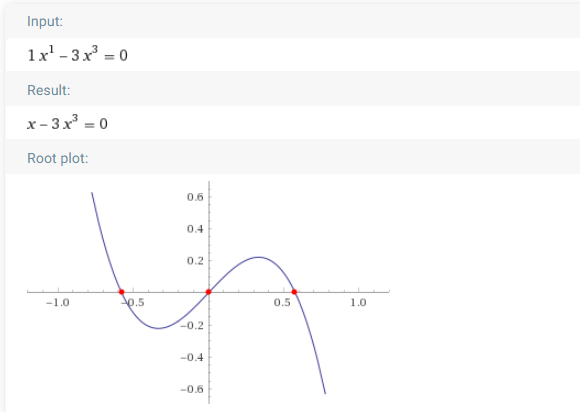
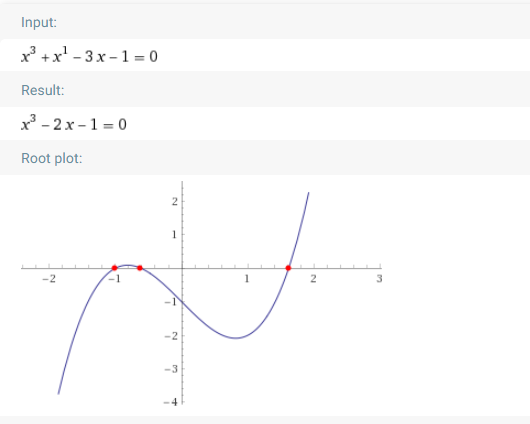


Таблица значений для уравнения выглядит так:

|  |  |
| --- | --- |
| Переменная | Значение |
| c | 1 |
| d | 3 |
| x0 | -3 |
| EPS | 0,0001 |

|  |  |
| --- | --- |
| Количество итераций | 17 |
| Ответ | 1.618050 |

Ответ, полученный с помощью калькулятора “Wolframalpha”.

****

**1.2.2 Метод касательной**

**Метод касательной (**или **метод Ньютона) -** это итерационный численный метод нахождения корня (нуля) заданной функции. Впервые данный метод предложил английский физик, математик и астроном Исаак Ньютон. Поиск решения осуществляется путём построения последовательных приближений и основан на принципах простой итерации. Метод обладает квадратичной сходимостью.

Дано:

Придуманные уравнения:

Найти: корни данных уравнений с помощью метода касательной.

Решение:

Рассмотрим листинг программного кода:

#include <stdio.h>

#include <math.h>

#define EPS 0.0000001

#define PI 3.141592653589793238462643383279

double tangetMethod(double, double, double);

double fx(double, double, double);

double dfx(double, double, double);

int main(int argc, char const \*argv[])

{

printf("Ответ : %lf\n", tangetMethod(4, 8, 2));

return 0;

}

double tangetMethod(double x, double c, double d)

{

double x1;

int count = 0;

x1 = x - fx(x, c, d) / dfx(x, c, d);

while (fabs(x1 - x) > EPS && count <= 1001)

{

x = x1;

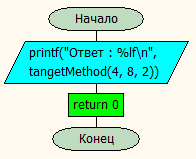
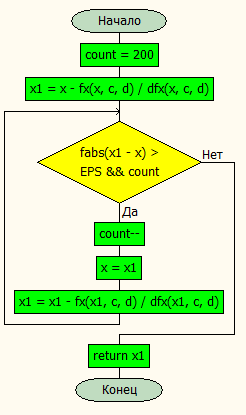
x1 = x1 - fx(x1, c, d) / dfx(x1, c, d);

}

return x1;

}

Блок-схема кода выглядит так:

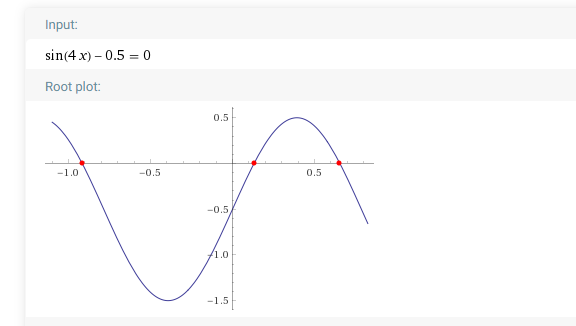
 

После ознакомления с исходным кодом и блок-схемой, перейдем к выполнению задачи.

Таблица данных для уравнения выглядит так.

|  |  |
| --- | --- |
| Переменные | Значения |
| c | 4 |
| d | 0.5 |
| x | 6 |
| EPS | 0.0000001 |

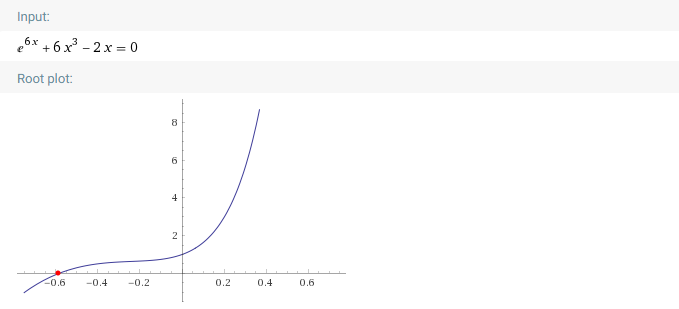
|  |  |
| --- | --- |
| Количество итераций | 16 |
| Ответ | 0.680751 |

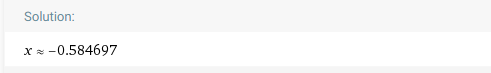


Для уравнения строим таблицу.

|  |  |
| --- | --- |
| Переменные | Значения |
| c | 4 |
| d | 6 |
| x | 2 |
| EPS | 0.0000001 |

|  |  |
| --- | --- |
| Количество итераций | 32 |
| Ответ | -0.584697 |





Построим таблицу данных для уравнения .

|  |  |
| --- | --- |
| Переменные | Значения |
| c | 8 |
| d | 2 |
| x | 4 |
| EPS | 0.0000001 |

|  |  |
| --- | --- |
| Количество итераций | 18 |
| Ответ | 0.496225 |

Сравним полученный ответ с Wolframalpha”.

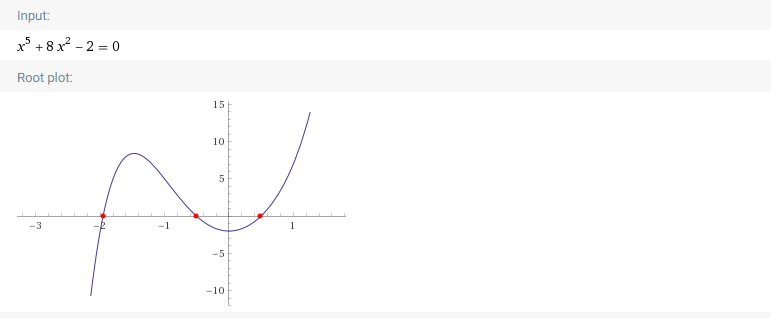
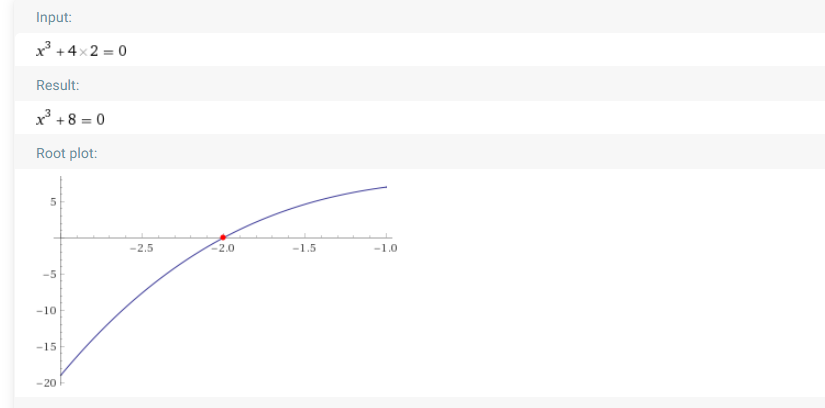
****

Таблица данных для уравнения выглядит так.

|  |  |
| --- | --- |
| Переменные | Значения |
| c | 4 |
| d | 2 |
| x | 4 |
| EPS | 0.0001 |

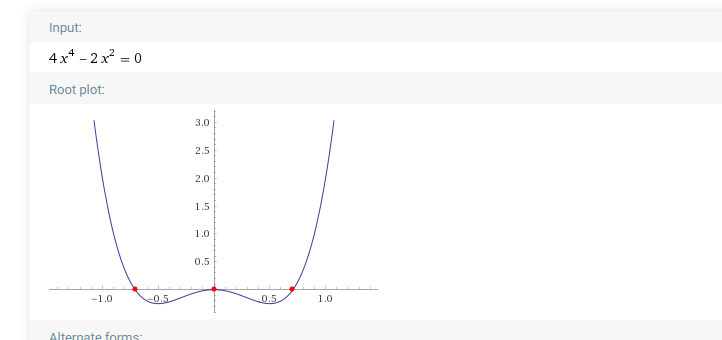
|  |  |
| --- | --- |
| Количество итераций | 32 |
| Ответ | -2.00000 |



Для уравнения строим таблицу.

|  |  |
| --- | --- |
| Переменные | Значения |
| c | 4 |
| d | 6 |
| x | 2 |
| EPS | 0. 0001 |

|  |  |
| --- | --- |
| Количество итераций | 13 |
| Ответ | 0.707118 |



Построим таблицу данных для уравнения .

|  |  |
| --- | --- |
| Переменные | Значения |
| c | 4 |
| d | 2 |
| x | 6 |
| EPS | 0.0001 |

|  |  |
| --- | --- |
| Количество итераций | 9 |
| Ответ | 1.000000 |

Сравним полученный ответ с Wolframalpha”.

****

Заключение: В данной главе мы рассмотрели метод деления отрезка пополам и метод касательных, научились находить корни уравнений с помощью этих методов. Можно заметить что метод касательных требует больше итераций чем метод деления отрезка пополам.

Глава II. Разработка игровой программы

В данной главе мы рассмотрим создание игровой программы на языке C.

Мне досталась игра “Морской бой”. Суть игры зависит от выбранного режима, всего их 3:

1. Игрок против другого игрока.
2. Игрок против компьютера.
3. Компьютер против компьютера.

В первом режиме игрокам предстоит потопить флот своего оппонента для того чтобы победить.

Второй режим – игрок сражается против компьютера, победителем становиться тот кто первый уничтожит все корабли противника.

Третий режим предлагает нам понаблюдать за игрой противников, находящихся под управление компьютера.

Все режимы подчиняются правилам классического “морского боя”.

Теперь мы можем перейти к рассмотрению строения программы.

Для начала введем обозначения:

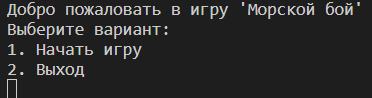
# - клетка занятая кораблем;

~ - поле;

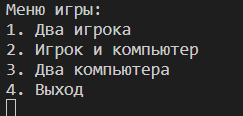
x – попадание по кораблю;

o - промах;

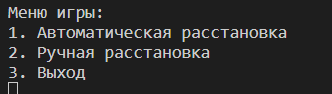
При запуске программы появляется меню игры, где мы можем либо начать игру, либо выйти из неё.



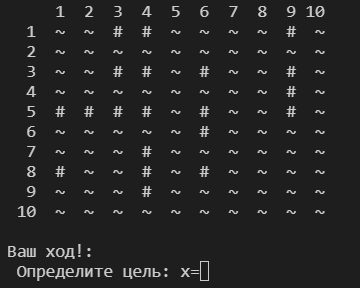
После начала игры нам предлагают выбрать игровой режим.



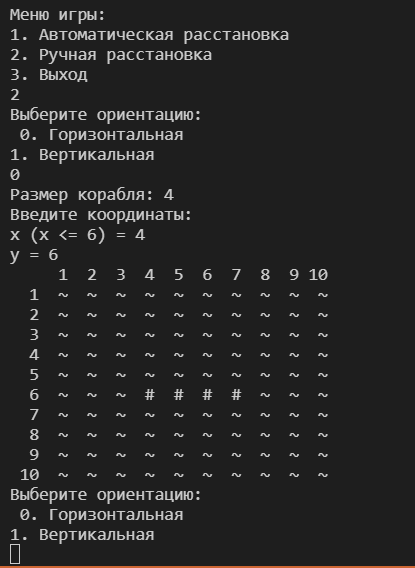
В зависимости от режима, выбираем способ расстановки, автоматический или ручной (только для двух игроков и игрок против компьютера).



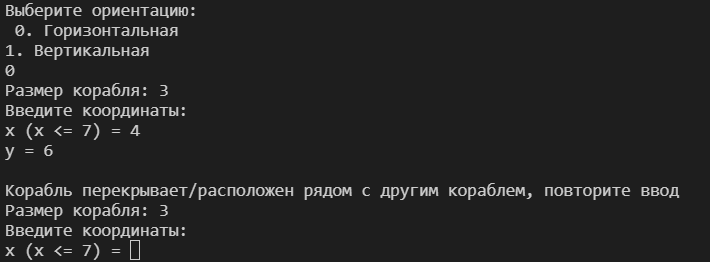
После расстановки игроку показывается его поле, и игра начинается.



Если выбрать ручную расстановку, то игрок сможет сам расположить свои корабли на поле в соответствии с правилами “морского боя”. Главное правило игры заключается в том, что корабли не должны соприкасаться/перекрывать друг друга.

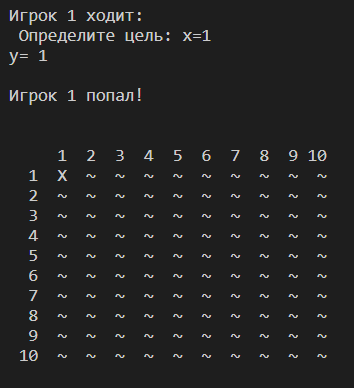
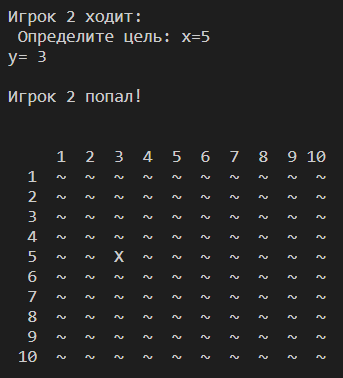


Если же мы попытаемся нарушить правила игры, то программа предупредит нас об этом.

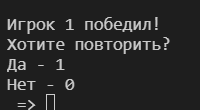


При выборе режима игрок против игрока Вам предложат выбрать ручную или автоматическую расстановку. Играющим не будут показаны их поля, они увидят лишь поле для выстрелов, так игроки не смогут узнать расположение кораблей друг друга.

После каждого выстрела будет сообщение о попадании/промахе и поле содержащие все ходы играющего.

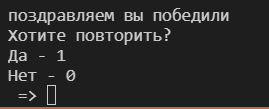
В конце игры игроки увидят победителя и предложение сыграть ещё раз.



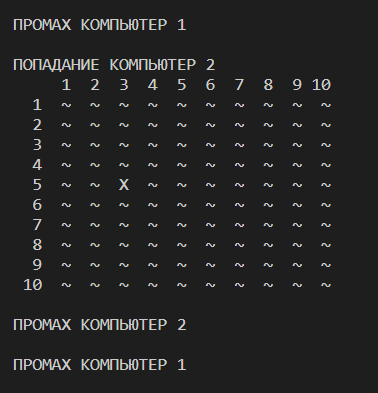
Если выбрать режим игрок против компьютера, то после выбора расстановки, вы увидите своё поле и игра начнется.



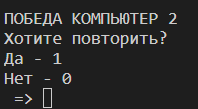
В конце игры будет выведено сообщение о победе или поражении и предложение повторить игру.



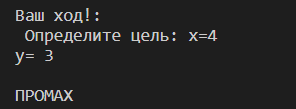
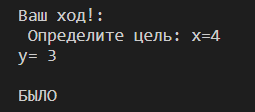
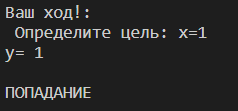
В режиме компьютер против компьютера мы можем понаблюдать за тем, как программа играет сама с собой.



В конце игры пользователь увидит победителя и предложение начать заново.



В каждом режиме игроку выводятся подсказки попал/промах, так же при повторном выстреле в одну точку.

После того как мы поняли принцип игры можно подробно рассмотреть код.

Начнем с функции game().

void game(int menu)

{

char f1[SIZE][SIZE];

char f2[SIZE][SIZE];

char fm1[SIZE][SIZE];

char fm2[SIZE][SIZE];

int x,y, count = 0;

int quantsh=4;

bool check;

int size; // размерность корабля

int r;

int menu2;

// system("clear");

maskMaps(f1,f2,fm1,fm2);

if (menu == 1)

{

// system("clear");

printf( " Меню игры:\n 1. Автоматическая расстановка\n 2. Ручная расстановка\n 3. Выход\n");

scanf("%d",&menu2);

// system("clear");

if (menu2==1)

{

initialization(f2);

initialization(f1);

battleUu(f2, fm2, f1, fm1);

}

if (menu2 == 2)

{

manualPlacement(f2);

manualPlacement(f1);

battleUu(f2, fm2, f1, fm1);

}

if (menu2 == 3)

{

printf(" Вы вышли!");

exit(0);

}

}

if (menu == 2)

{

printf( " Меню игры:\n 1. Автоматическая расстановка\n 2. Ручная расстановка\n 3. Выход\n");

scanf("%d",&menu2);

// system("clear");

if (menu2==1)

{

initialization(f2);

initialization(f1);

}

if (menu2 == 2)

{

printMaps(f2);

manualPlacement(f2);

}

else

printf(" Вы вышли!\n");

exit(0);

// system("clear");

check = true;

battleuc(f2,fm2, f1, fm1);

}

if (menu == 3)

{

initialization(f2);

initialization(f1);

int count1=20;

int count2=20;

do {

count1 = battcc(f1,fm1, 1,count1, printMaps);

count2 = battcc(f2,fm2, 2, count2, printMaps);

} while (count1!=0 && count2!=0);

battlecc(count2);

}

else

{

printf("Вы вышли!");

exit(0);

}

}

Эта функция играет роль меню и предлагает пользователю выбрать режим игры.

Следующие функции, которые мы рассмотрим – функция initialization() и verticalShip().

Они отвечают за автоматическую расстановку кораблей.

void initialization(char field[][10])

{

int quentsh[4]={1, 2, 3, 4};

int ship\_size[4]={4, 3, 2, 1};

for (int i = 0; i < 4; i++)

while (quentsh[i])

{

if (rand() % 2)

verticalShip(field, ship\_size[i]);

else

horizontalShip(field, ship\_size[i]);

quentsh[i]--;

}

}

Эта функция отвечает за вертикальное расположение кораблей на поле.

void verticalShip(char arr[][SIZE], int ship\_size)

{

while (1)

{

if (ship\_size == 4)

{

int x = rand() % 7;

int y = rand() % 10;

for (int i = x; i < x + 4; i++)

arr[i][y] = '#';

break;

}

if (ship\_size == 3)

{

int x = rand() % 8;

int y = rand() % 10;

if ( arr[x][y] == '#' || arr[x - 1][y] == '#' || arr[x - 1][y + 1] == '#' || arr[x][y + 1] == '#' || arr[x + 1][y + 1] == '#' || [x + 2][y + 1] == '#' || arr[x + 3][y + 1] == '#' || arr[x + 3][y] == '#' || arr[x + 3][y - 1] == '#' || arr[x + 2][y - 1] == '#' || arr[x + 1][y - 1] == '#' || arr[x][y - 1] == '#' || arr[x - 1][y - 1] == '#' )

continue;

else

for (int i = x; i < x + 3; i++)

arr[i][y] = '#';

break;

}

…

}

}

Данная функция “рисует” корабли на поле, проверяя чтобы они не соприкасались друг с другом. Для четырехпалубного корабля никаких условий не проверяется, так как это первый корабль на поле. Для всех остальных проверяется периметр и координата внутри корабля, чтобы избежать наложения кораблей. Функция horizontalShip() похожа на эту, поэтому её рассматривать мы не будем.

Функции rHorizontalShip() и rVerticalShip() отвечают за ручную расстановку кораблей и

ничем особенным от рассмотренной функции не отличаются.

Перейдем к функции battleUu().

void battleUu(char field2[][SIZE], char field\_m2[][SIZE], char field[][SIZE] ,char field\_m[][SIZE])

{

int x, y;

int menu2;

bool step;

int usr\_count\_1 = 20;

int usr\_count\_2 = 20;

do {

step = true;

while (step && usr\_count\_1)

{

printf( "\nИгрок 1 ходит:\n Определите цель: х=");

scanf("%d", &x);

printf( "y= ");

scanf("%d", &y);

if(x<1 || x>10 || y<1 || y>10)

{

printf( "\nВЫХОД ЗА ПРЕДЕЛЫ\n");

continue;

}

if (field[y-1][x-1] == 'X' || field\_m[y-1][x-1] == 'X' || field[y-1][x-1] == 'O' || field\_m[y-1][x-1] == 'O')

{

printf( "\nБЫЛО\n");

continue;

}

x--;

y--;

while (step)

{

step = shot(field\_m, field, step, x, y);

if (step)

{

printf("\nИгрок 1 попал!\n");

usr\_count\_1--;

printf("\n");

printf("\n");

printMaps(field\_m);

if ( (field[x][y] == '#' || field[x - 1][y] == '#' || field[x - 1][y + 1] == '#' || field[x][y + 1] == '#' || field[x + 1][y + 1] == '#' || field[x + 1][y] == '#' || field[x + 1][y - 1] == '#' || field[x][y - 1] == '#' || field[x - 1][y - 1] == '#') || (x<1 || x>10 || y<1 || y>10) )

break;

else

printf("\nУБИТ\n");

break;

}

else{

printf("\nИгрок 1 не попал \n");

break;

}

}

}

…

}

Данная функция отвечает за режим игры игрок против игрока. Она предоставляет поочерёдный ход каждому игроку, проверяя при этом попадание/промах. При повтороном попадании в точку игроку предложат выбрать другие координаты, так исключается пропуск хода ирока. Функция battleuc() отвечает за режим игрок против компьютера, а функция battcc() за режим компьютер против компьютера. Из-за их схожести с функцией battleUu() они рассматриваться не будут.

Одна из самых важных функций – shot(). Она отвечает за выстрел игрока по полю противника. Рассмотрим листинг этой функции.

bool shot(char fiedl\_m[][SIZE], char field[][SIZE], bool step, int x, int y)

{

if (field[y][x] == '#')

{

field[y][x] = 'X';

fiedl\_m[y][x] = 'X';

} else {

field[y][x] = 'O';

fiedl\_m[y][x] = 'O';

step = false;

}

return step;

}

Она принимает координаты и ставит метку на поле противника.

На этом можно закончить листинг кода программы, так как все основные функции были рассмотрены.

Заключение

Язык C был разработан для создания ОС, но как мы успели убедиться, он нашел применение практически в любых сферах программирования. Он отлично подходит тем, кто не любит ограничения и кому важна скорость выполнения программ. Но стоит быть внимательным, отсутствие ограничений в работе с памятью может привести к ошибкам в программе.

Приложение

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <stdbool.h>

#include <math.h>

#define SIZE 10

typedef void (\*u)(char fm[][SIZE]); // указатель на функцию printMaps printMaps

int menu();

void printMaps(char [][SIZE]);

void game(int );

void initialization(char [][10]);

void verticalShip(char [][SIZE], int);

void horizontalShip(char [][SIZE], int);

void maskMaps(char [][SIZE], char [][SIZE], char [][SIZE], char [][SIZE]);

void battleuc(char [][SIZE], char [][SIZE], char [][SIZE], char [][SIZE]) ;

int battcc(char [][SIZE], char [][SIZE], int, int, u );

void battleUu(char [][SIZE], char [][SIZE], char [][SIZE] ,char [][SIZE]);

void battlecc(int);

bool shot(char [][SIZE], char [][SIZE], bool , int ,int );

void manualPlacement(char [][SIZE]);

void rVerticalShip(char [][SIZE], int);

void rHorizontalShip(char [][SIZE], int);

int main(int argc, char const \*argv[])

{

int menu;

int menu2;

srand( time(NULL) );

printf( "Игра 'Морской бой'\nВыберите вариант:\n"

" 1. Начать игру\n 2. Выход\n");

while(1)

{

scanf("%d",&menu);

// system("clear");

switch (menu)

{

case 1: printf( " Меню игры:\n 1. Два игрока\n 2. Игрок и компьютер\n 3. Два компьютера\n 4. Выход\n");

scanf("%d",&menu2);

game(menu2);

return menu2;

case 2 : printf(" BB");

return 0;

default:

printf(" ERROR\n");

continue;

}

}

return 0;

}

void printMaps(char field[][SIZE]) // реализация ффункции после заполнения компуктера

{

printf("%3c", ' ');

for (int i = 0; i < 10; i++)

printf("%3d", i + 1);

printf("\n");

for(int j = 0; j < 10; j++)

{

printf("%3d", j + 1);

for (int i = 0; i < 10; i++)

printf("%3c", field[i][j]);

printf("\n");

}

}

void maskMaps(char field[][SIZE], char field2[][SIZE], char field\_m[][SIZE], char field\_m2[][SIZE]) // функция отображения карты и маски на нее

{

for (int j = 0; j < 10; j++)

for (int i = 0; i < 10; i++)

{

field[i][j] = '~';

field2[i][j] = '~';

}

for (int j = 0; j < 10; j++)

for (int i = 0; i < 10; i++)

{

field\_m[i][j] = '~';

field\_m2[i][j] = '~';

}

}

void game(int menu)

{

char f1[SIZE][SIZE];

char f2[SIZE][SIZE];

char fm1[SIZE][SIZE];

char fm2[SIZE][SIZE];

int x,y, count = 0;

int quantsh=4;

bool check;

int size; // размерность корабля

int r;

int menu2;

// system("clear");

maskMaps(f1,f2,fm1,fm2);

if (menu == 1)

{

// system("clear");

printf( " Меню игры:\n 1. Автоматическая расстановка\n 2. Ручная расстановка\n 3. Выход\n");

scanf("%d",&menu2);

// system("clear");

if (menu2==1)

{

initialization(f2);

initialization(f1);

battleUu(f2, fm2, f1, fm1);

}

if (menu2 == 2)

{

manualPlacement(f2);

manualPlacement(f1);

battleUu(f2, fm2, f1, fm1);

}

if (menu2 == 3)

{

printf(" Вы вышли!");

exit(0);

}

}

if (menu == 2)

{

printf( " Меню игры:\n 1. Автоматическая расстановка\n 2. Ручная расстановка\n 3. Выход\n");

scanf("%d",&menu2);

// system("clear");

if (menu2==1)

{

initialization(f2);

initialization(f1);

}

if (menu2 == 2)

{

printMaps(f2);

manualPlacement(f2);

}

else

printf(" Вы вышли!\n");

exit(0);

// system("clear");

check = true;

battleuc(f2,fm2, f1, fm1);

}

if (menu == 3)

{

initialization(f2);

initialization(f1);

int count1=20;

int count2=20;

do {

count1 = battcc(f1,fm1, 1,count1, printMaps);

count2 = battcc(f2,fm2, 2, count2, printMaps);

} while (count1!=0 && count2!=0);

battlecc(count2);

}

else

{

printf("Вы вышли!");

exit(0);

}

}

void battleuc(char field2[][SIZE],char field\_m2[][SIZE], char field[][SIZE] ,char field\_m1[][SIZE])

{

int x, y, menu2;

int usr\_count = 20, cmp\_count = 20;

bool step;

printMaps(field2);

do {

step = true;

while (step && usr\_count)

{

printf( "\nВаш ход!:\n Определите цель: х="); scanf("%d", &x);

printf( "y= "); scanf("%d", &y);

if(x<1 || x>10 || y<1 || y>10)

{

printf( "\nВЫХОД ЗА ПРЕДЕЛЫ\n");

continue;

}

if (field[y-1][x-1] == 'X' || field\_m1[y-1][x-1] == 'X' || field[y-1][x-1] == 'O' || field\_m1[y-1][x-1] == 'O')

{

printf( "\nБЫЛО\n");

continue;

}

x--;y--;

while (step)

{

step = shot(field\_m1, field, step, x, y);

if (step)

{

printf("\nПОПАДАНИЕ\n");

usr\_count--;

printf("\n");

printf("\n");

printMaps(field\_m1);

if ( (field[x][y] == '#' || field[x - 1][y] == '#' || field[x - 1][y + 1] == '#' || field[x][y + 1] == '#' || field[x + 1][y + 1] == '#' ||

field[x + 1][y] == '#' || field[x + 1][y - 1] == '#' || field[x][y - 1] == '#' || field[x - 1][y - 1] == '#') || (x<1 || x>10 || y<1 || y>10) )

break;

else

printf("\nУБИТ\n");

break;

}

else

{

printf("\nПРОМАХ\n");

printMaps(field\_m1);

break;

}

}

}

if (!usr\_count)

{

printf( "\nпоздравляем вы победили\n");

break;

}

step = true;

while (step)

{

int xc = 1 + rand() % 11;

int yc = 1 + rand() % 11;

xc--; yc--;

while(1)

{

if (field2[yc][xc] == 'X' || field\_m2[yc][xc] == 'X' || field2[yc][xc] == 'O' || field\_m2[yc][xc] == 'O')

{

printf("БЫЛО");

xc =1 + rand() % 11;

yc =1 + rand() % 11;

xc--; yc--;

}

else

break;

}

step=shot(field\_m2, field2, step, xc, yc);

if (step)

{

printf( "\nКомпьютер попал!\n");

cmp\_count--;

bool f = !step;

printMaps(field\_m2);

}

else

printf( "\nКомпьютер не попал!\n");

}

} while (usr\_count || cmp\_count);

int reset;

printf( "Хотите повторить?\n");

printf( "Да - 1\nНет - 0\n => ");

scanf("%d", &reset);

if (!reset)

{

printf( "Вы вышли!");

exit(0);

}

else

{

reset = !reset;//??

maskMaps(field, field2, field\_m1, field\_m2);

game(menu2);

}

}

bool shot(char fiedl\_m[][SIZE], char field[][SIZE], bool step, int x, int y)

{

if (field[y][x] == '#')

{

field[y][x] = 'X';

fiedl\_m[y][x] = 'X';

} else {

field[y][x] = 'O';

fiedl\_m[y][x] = 'O';

step = false;

}

return step;

}

void initialization(char field[][10])

{

int quentsh[4]={1, 2, 3, 4};

int ship\_size[4]={4, 3, 2, 1};

for (int i = 0; i < 4; i++)

while (quentsh[i])

{

if (rand() % 2)

verticalShip(field, ship\_size[i]);

else

horizontalShip(field, ship\_size[i]);

quentsh[i]--;

}

}

void verticalShip(char arr[][SIZE], int ship\_size)

{

while (1)

{

if (ship\_size == 4)

{

int x = rand() % 7;

int y = rand() % 10;

for (int i = x; i < x + 4; i++)

arr[i][y] = '#';

break;

}

if (ship\_size == 3)

{

int x = rand() % 8;

int y = rand() % 10;

if ( arr[x][y] == '#' || arr[x - 1][y] == '#' || arr[x - 1][y + 1] == '#' || arr[x][y + 1] == '#' || arr[x + 1][y + 1] == '#' ||

arr[x + 2][y + 1] == '#' || arr[x + 3][y + 1] == '#' || arr[x + 3][y] == '#' || arr[x + 3][y - 1] == '#' || arr[x + 2][y - 1] == '#' ||

arr[x + 1][y - 1] == '#' || arr[x][y - 1] == '#' || arr[x - 1][y - 1] == '#' )

continue;

else

for (int i = x; i < x + 3; i++)

arr[i][y] = '#';

break;

}

if (ship\_size == 2)

{

int x = rand() % 9;

int y = rand() % 10;

if ( arr[x][y] == '#' || arr[x - 1][y] == '#' || arr[x - 1][y + 1] == '#' || arr[x][y + 1] == '#' || arr[x + 1][y + 1] == '#' ||

arr[x + 2][y + 1] == '#' || arr[x + 2][y] == '#' || arr[x + 2][y - 1] == '#' || arr[x + 1][y - 1] == '#' || arr[x][y - 1] == '#' ||

arr[x - 1][y - 1] == '#' )

continue;

else

for (int i = x; i < x + 2; i++)

arr[i][y] = '#';

break;

}

if (ship\_size == 1)

{

int x = rand() % 10;

int y = rand() % 10;

if (arr[x][y] == '#' || arr[x - 1][y] == '#' || arr[x - 1][y + 1] == '#' || arr[x][y + 1] == '#' || arr[x + 1][y + 1] == '#' ||

arr[x + 1][y] == '#' || arr[x + 1][y - 1] == '#' || arr[x][y - 1] == '#' || arr[x - 1][y - 1] == '#')

continue;

else

arr[x][y] = '#';

break;

}

}

}

void horizontalShip(char arr[][SIZE], int ship\_size)

{

while (1)

{

if (ship\_size == 4)

{

int x = rand() % 10;

int y = rand() % 7;

for (int i = y; i < y + 4; i++)

arr[x][i] = '#';

break;

}

if (ship\_size == 3)

{

int x = rand() % 10;

int y = rand() % 8;

if (arr[x][y] == '#' || arr[x][y - 1] == '#' || arr[x - 1][y - 1] == '#' || arr[x - 1][y] == '#' || arr[x - 1][y + 1] == '#' ||

arr[x - 1][y + 2] == '#' || arr[x - 1][y + 3] == '#' || arr[x][y + 3] == '#' || arr[x + 1][y + 3] == '#' || arr[x + 1][y + 2] == '#' ||

arr[x + 1][y + 1] == '#' || arr[x + 1][y] == '#' || arr[x + 1][y + 1] == '#' )

continue;

else

for (int i = y; i < y + 3; i++)

arr[x][i] = '#';

break;

}

if (ship\_size == 2)

{

int x = rand() % 10;

int y = rand() % 9;

if (arr[x][y] == '#' || arr[x][y - 1] == '#' || arr[x - 1][y - 1] == '#' || arr[x - 1][y] == '#' || arr[x - 1][y + 1] == '#' ||

arr[x - 1][y + 2] == '#' || arr[x][y + 2] == '#' || arr[x + 1][y + 2] == '#' || arr[x + 1][y + 1] == '#' || arr[x + 1][y] == '#' ||

arr[x + 1][y - 1] == '#' )

continue;

else

for (int i = y; i < y + 2; i++)

arr[x][i] = '#';

break;

}

if (ship\_size == 1)

{

int x = rand() % 10;

int y = rand() % 10;

if (arr[x][y] == '#' || arr[x - 1][y] == '#' || arr[x - 1][y + 1] == '#' || arr[x][y + 1] == '#' || arr[x + 1][y + 1] == '#' ||

arr[x + 1][y] == '#' || arr[x + 1][y - 1] == '#' || arr[x][y - 1] == '#' || arr[x - 1][y - 1] == '#')

continue;

else

arr[x][y] = '#';

break;

}

}

}

int battcc(char field[][SIZE],char field\_m[][SIZE], int cmp, int count, u out)

{

int x, y;

bool step;

step = true;

while (step)

{

int x = 1 + rand() % 11;

int y = 1 + rand() % 11;

x--; y--;

while(1)

{

if (field[y][x] == 'X' || field\_m[y][x] == 'X' || field[y][x] == 'O' || field\_m[y][x] == 'O')

{

printf("БЫЛО ");

x = 1 + rand() % 11;

y = 1 + rand() % 11;

x--; y--;

}

else

break;

}

step=shot(field\_m, field, step, x, y);

if (step)

{

printf("\nПОПАДАНИЕ КОМПЬЮТЕР %d\n", cmp);

count--;

bool f = !step;

out(field\_m);

}

else

printf("\nПРОМАХ КОМПЬЮТЕР %d\n", cmp);

}

return count;

}

void battlecc(int count)

{

int menu2=3;

if (!count)

printf( "\nПОБЕДА КОМПЬЮТЕР 1\n");

else

printf( "\nПОБЕДА КОМПЬЮТЕР 2\n");

printf( "Хотите повторить?\n");

printf( "Да - 1\nНет - 0\n => ");

int var;

scanf("%d", &var);

if (!var)

{

printf( "Вы вышли!");

exit(0);

}

else

{

var = !var;

game(menu2);

}

}

void manualPlacement(char field[][SIZE])

{

int n[4] = {1, 2, 3, 4};

int k[4] = {4, 3, 2, 1};

int orient;

for (int i = 0; i < 4; i++)

while (n[i])

{

printf( "Выберите ориентацию:\n 0. Горизонтальная\n1. Вертикальная\n");

scanf("%d", &orient);

if (!orient)

rVerticalShip(field, k[i]);

else

rHorizontalShip(field, k[i]);

n[i]--;

printMaps(field);

getchar();

}

}

void rVerticalShip(char arr[][SIZE], int ship\_size)

{

int x, y;

while (1)

{

printf("Размер корабля: %d", ship\_size);

printf("\nВведите координаты:\n");

if (ship\_size == 4)

{

printf("x (x <= 6) = "); scanf("%d", &x);

printf("y = "); scanf("%d", &y);

if (x < 1 || x > 6 || y < 1 || y > 10)

{

printf("\nВы ввели неверную координату x > 6 или вышли за пределы поля, повторите ввод!\n");

continue;

}

x--; y--;

for (int i = x; i < x + 4; i++)

arr[i][y] = '#';

break;

}

if (ship\_size == 3)

{

printf("x (x <= 7) = "); scanf("%d", &x);

printf("y = "); scanf("%d", &y);

if (x < 1 || x > 7 || y < 1 || y > 10)

{

printf("\nВы ввели неверную координату x > 7 или вышли за пределы поля, повторите ввод!\n");

continue;

}

x--; y--;

if ( arr[x][y] == '#' || arr[x - 1][y] == '#' || arr[x - 1][y + 1] == '#' || arr[x][y + 1] == '#' || arr[x + 1][y + 1] == '#' ||

arr[x + 2][y + 1] == '#' || arr[x + 3][y + 1] == '#' || arr[x + 3][y] == '#' || arr[x + 3][y - 1] == '#' || arr[x + 2][y - 1] == '#' ||

arr[x + 1][y - 1] == '#' || arr[x][y - 1] == '#' || arr[x - 1][y - 1] == '#' )

{

printf("\nКорабль перекрывает/расположен рядом с другим кораблем, повторите ввод\n");

continue;

}

else

for (int i = x; i < x + 3; i++)

arr[i][y] = '#';

break;

}

if (ship\_size == 2)

{

printf("x (x <= 8) = "); scanf("%d", &x);

printf("y = "); scanf("%d", &y);

if (x < 1 || x > 8 || y < 1 || y > 10)

{

printf("\nВы ввели неверную координату x > 8 или вышли за пределы поля, повторите ввод!\n");

continue;

}

x--; y--;

if ( arr[x][y] == '#' || arr[x - 1][y] == '#' || arr[x - 1][y + 1] == '#' || arr[x][y + 1] == '#' || arr[x + 1][y + 1] == '#' ||

arr[x + 2][y + 1] == '#' || arr[x + 2][y] == '#' || arr[x + 2][y - 1] == '#' || arr[x + 1][y - 1] == '#' || arr[x][y - 1] == '#' ||

arr[x - 1][y - 1] == '#' )

{

printf("\nКорабль перекрывает/расположен рядом с другим кораблем, повторите ввод\n");

continue;

}

else

for (int i = x; i < x + 2; i++)

arr[i][y] = '#';

break;

}

if (ship\_size == 1)

{

printf("x = "); scanf("%d", &x);

printf("y = "); scanf("%d", &y);

if (x < 1 || x > 10 || y < 1 || y > 10)

{

printf("\nВы вышли за пределы поля, повторите ввод!\n");

continue;

}

x--; y--;

if (arr[x][y] == '#' || arr[x - 1][y] == '#' || arr[x - 1][y + 1] == '#' || arr[x][y + 1] == '#' || arr[x + 1][y + 1] == '#' ||

arr[x + 1][y] == '#' || arr[x + 1][y - 1] == '#' || arr[x][y - 1] == '#' || arr[x - 1][y - 1] == '#')

{

printf("\nКорабль перекрывает/расположен рядом с другим кораблем, повторите ввод\n");

continue;

}

else

arr[x][y] = '#';

break;

}

}

}

void rHorizontalShip(char arr[][SIZE], int ship\_size)

{

int x, y;

while (1)

{

printf("Размер корабля: %d", ship\_size);

printf("\nВведите координаты:\n");

if (ship\_size == 4)

{

printf("x = "); scanf("%d", &x);

printf("y (y <= 6) = "); scanf("%d", &y);

if (x < 1 || x > 10 || y < 1 || y > 6)

{

printf("\nВы ввели неверную координату y > 6 или вышли за пределы поля, повторите ввод!\n");

continue;

}

x--; y--;

for (int i = y; i < y + 4; i++)

arr[x][i] = '#';

break;

}

if (ship\_size == 3)

{

printf("x = "); scanf("%d", &x);

printf("y (y <= 7) = "); scanf("%d", &y);

if (x < 1 || x > 10 || y < 1 || y > 7)

{

printf("\nВы ввели неверную координату y > 7 или вышли за пределы поля, повторите ввод!\n");

continue;

}

x--; y--;

if (arr[x][y] == '#' || arr[x][y - 1] == '#' || arr[x - 1][y - 1] == '#' || arr[x - 1][y] == '#' || arr[x - 1][y + 1] == '#' ||

arr[x - 1][y + 2] == '#' || arr[x - 1][y + 3] == '#' || arr[x][y + 3] == '#' || arr[x + 1][y + 3] == '#' || arr[x + 1][y + 2] == '#' ||

arr[x + 1][y + 1] == '#' || arr[x + 1][y] == '#' || arr[x + 1][y + 1] == '#' )

{

printf("\nКорабль перекрывает/расположен рядом с другим кораблем, повторите ввод\n");

continue;

}

else

for (int i = y; i < y + 3; i++)

arr[x][i] = '#';

break;

}

if (ship\_size == 2)

{

printf("x = "); scanf("%d", &x);

printf("y (y <= 8) = "); scanf("%d", &y);

if (x < 1 || x > 10 || y < 1 || y > 8)

{

printf("\nВы ввели неверную координату y > 8 или вышли за пределы поля, повторите ввод!\n");

continue;

}

x--; y--;

if (arr[x][y] == '#' || arr[x][y - 1] == '#' || arr[x - 1][y - 1] == '#' || arr[x - 1][y] == '#' || arr[x - 1][y + 1] == '#' ||

arr[x - 1][y + 2] == '#' || arr[x][y + 2] == '#' || arr[x + 1][y + 2] == '#' || arr[x + 1][y + 1] == '#' || arr[x + 1][y] == '#' ||

arr[x + 1][y - 1] == '#' )

{

printf("\nКорабль перекрывает/расположен рядом с другим кораблем, повторите ввод\n");

continue;

}

else

for (int i = y; i < y + 2; i++)

arr[x][i] = '#';

break;

}

if (ship\_size == 1)

{

printf("x = "); scanf("%d", &x);

printf("y = "); scanf("%d", &y);

if (x < 1 || x > 10 || y < 1 || y > 10)

{

printf("\nВы вышли за пределы поля, повторите ввод!\n");

continue;

}

x--; y--;

if (arr[x][y] == '#' || arr[x - 1][y] == '#' || arr[x - 1][y + 1] == '#' || arr[x][y + 1] == '#' || arr[x + 1][y + 1] == '#' ||

arr[x + 1][y] == '#' || arr[x + 1][y - 1] == '#' || arr[x][y - 1] == '#' || arr[x - 1][y - 1] == '#')

{

printf("\nКорабль перекрывает/расположен рядом с другим кораблем, повторите ввод\n");

continue;

}

else

arr[x][y] = '#';

break;

}

}

}

void battleUu(char field2[][SIZE], char field\_m2[][SIZE], char field[][SIZE] ,char field\_m[][SIZE])

{

int x, y;

int menu2;

bool step;

int usr\_count\_1 = 20;

int usr\_count\_2 = 20;

do {

step = true;

while (step && usr\_count\_1)

{

printf( "\nИгрок 1 ходит:\n Определите цель: х=");

scanf("%d", &x);

printf( "y= ");

scanf("%d", &y);

if(x<1 || x>10 || y<1 || y>10)

{

printf( "\nВЫХОД ЗА ПРЕДЕЛЫ\n");

continue;

}

if (field[y-1][x-1] == 'X' || field\_m[y-1][x-1] == 'X' || field[y-1][x-1] == 'O' || field\_m[y-1][x-1] == 'O')

{

printf( "\nБЫЛО\n");

continue;

}

x--;

y--;

while (step)

{

step = shot(field\_m, field, step, x, y);

if (step)

{

printf("\nИгрок 1 попал!\n");

usr\_count\_1--;

printf("\n");

printf("\n");

printMaps(field\_m);

if ( (field[x][y] == '#' || field[x - 1][y] == '#' || field[x - 1][y + 1] == '#' || field[x][y + 1] == '#' || field[x + 1][y + 1] == '#' || field[x + 1][y] == '#' || field[x + 1][y - 1] == '#' || field[x][y - 1] == '#' || field[x - 1][y - 1] == '#') || (x<1 || x>10 || y<1 || y>10) )

break;

else

printf("\nУБИТ\n");

break;

}

else{

printf("\nИгрок 1 не попал \n");

break;

}

}

}

if (!usr\_count\_1)

{

printf( "\nИгрок 1 победил!\n");

break;

}

step = true;

while (step && usr\_count\_2)

{

printf( "\nИгрок 2 ходит:\n Определите цель: х=");

scanf("%d", &x);

printf( "y= ");

scanf("%d", &y);

if(x<1 || x>10 || y<1 || y>10)

{

printf( "\nВЫХОД ЗА ПРЕДЕЛЫ\n");

continue;

}

if (field2[y-1][x-1] == 'X' || field\_m2[y-1][x-1] == 'X' || field2[y-1][x-1] == 'O' || field\_m2[y-1][x-1] == 'O')

{

printf( "\nБЫЛО\n");

continue;

}

x--;

y--;

while (step)

{

step = shot(field\_m2, field2, step, x, y);

if (step)

{

printf("\nИгрок 2 попал!\n");

usr\_count\_2--;

printf("\n");

printf("\n");

printMaps(field\_m2);

if ( (field2[x][y] == '#' || field2[x - 1][y] == '#' || field2[x - 1][y + 1] == '#' || field2[x][y + 1] == '#' || field2[x + 1][y + 1] == '#' || field2[x + 1][y] == '#' || field2[x + 1][y - 1] == '#' || field2[x][y - 1] == '#' || field2[x - 1][y - 1] == '#') || (x<1 || x>10 || y<1 || y>10) )

break;

else

printf("\nУБИТ\n");

break;

}

else{

printf("\nИгрок 2 не попал \n");

break;

}

}

}

} while (usr\_count\_1 || usr\_count\_2);

printf( "Хотите повторить?\n");

printf( "Да - 1\nНет - 0\n => ");

int var;

scanf("%d", &var);

if (!var)

{

printf( "Вы вышли!");

exit(0);

}

else

{

var = !var;

maskMaps(field, field2, field\_m, field\_m2);

game(menu2);

}

}

# **Список литературы**

1. Си (язык программирования) <https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)>
2. История языка Си <https://habr.com/ru/post/114588/>
3. Ряд математика <https://ru.wikipedia.org/wiki/%D0%A0%D1%8F%D0%B4_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)>
4. Уравнение

<https://ru.wikipedia.org/wiki/%D0%A3%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5>

1. Метод бисекции <https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%B1%D0%B8%D1%81%D0%B5%D0%BA%D1%86%D0%B8%D0%B8>
2. Метод Ньютона <https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%9D%D1%8C%D1%8E%D1%82%D0%BE%D0%BD%D0%B0>
3. Морской бой <https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D1%80%D1%81%D0%BA%D0%BE%D0%B9_%D0%B1%D0%BE%D0%B9_(%D0%B8%D0%B3%D1%80%D0%B0)>
4. Майк МакГрат. Программирование на С для начинающих. ­–Эксмо.: Москва, 2016. – 192 с.
5. Стивен Скиена Алгоритмы. Руководство по разработке, 2011. – 719 c.
6. Дэвид Гриффитс Дон Гриффитс. Изучаем программирование на C. ­–Эксмо.: Москва, 2013. – 624 с.