Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Лабораторная работа № 1

По дисциплине «Основы алгоритмизации и программирования»

На тему «Указатели на функции»

Выполнил:

Студент 1 курса 8 группы

Статько Герман Вячеславович

Преподаватель: асс. Андронова М.В.

Минск, 2024

В соответствии со своим вариантом написать программы по условиям задач из таблицы ниже. Программа должна содержать функцию пользователя с переменным числом параметров и не менее трех обращений к ней с различным количеством параметров.

**Вариант 12**



#include <iostream>

#include <cstdarg>

#include <iomanip>

using namespace std;

void swap(int, ...); // Объявление прототипа функции для обмена и последующей замены элементов массива

int main()

{

setlocale(LC\_CTYPE, "rus"); // Установка русской локали для консоли

int n = 6;

swap(n, -6, 8, 9, 15, -8, 11); // Вызов функции для обмена и последующей замены элементов массива

swap(n, 2, 3, -5, -6, -7, 9); // Вызов функции для обмена и последующей замены элементов массива

swap(n, -16, 12, 6, 7, -19, -12); // Вызов функции для обмена и последующей замены элементов массива

system("pause");

}

void swap(int size, ...)

{

int buffer; // Буферная переменная для обмена элементов

va\_list args; // Список аргументов переменной длины

va\_start(args, size); // Инициализация списка аргументов

int\* A;

A = new int[size]; // Создание динамического массива

cout << "Изначальный массив: ";

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

{

A[i] = va\_arg(args, int); // Получение значения аргумента

cout << setw(3) << A[i] << " "; // Вывод элементов массива

}

cout << endl;

va\_end(args); // Завершаем работу со списком аргументов

// Проверка наличия отрицательных элементов и их замена на нули

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

{

if (A[i] < 0)

{

A[i] = 0;

}

}

// Перемещение нулевых элементов в конец массива

for (int k = 0; k < size; k++) { // Дополнительные проходы основного цикла для корректного результата(случай с 0;0)

for (int i = 0; i < size - 1; i++)

{

if (A[i] == 0)

{

buffer = \*(A + i);

\*(A + i) = \*(A + i + 1);

\*(A + i + 1) = buffer;

}

}

}

cout << "Измененный массив: ";

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

{

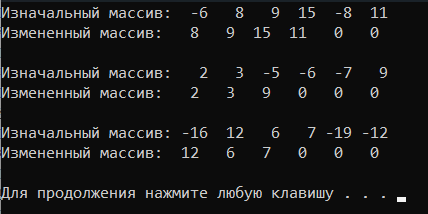
cout << setw(3) << A[i] << " "; // Вывод измененного массива

}

cout << endl << endl;

delete[] A; // Освобождение памяти от динамического массива

}



В соответствии со своим вариантом отделить корни двух уравнений и вычислить их методом дихотомии для исходных данных из таблицы, приведенной ниже. Точность вычислений принять равной e = 0,001 для всех вариантов.

Операторы метода вычисления корня оформить в виде функции пользователя, уравнения записать также в виде функций пользователя.

В главной функции предусмотреть ввод исходных данных, обращения к функции, реализующей метод дихотомии для двух уравнений. В процессе выполнения программы определить корни двух уравнений. Использовать указатель на функцию.



#include <iostream>

using namespace std;

// Объявление прототипов функций

double metodD(double (\*)(double), double, double, double, int);

double f(double);

double s(double);

int main()

{

setlocale(LC\_CTYPE, "rus"); // Установка русской локали для консоли

int a = 1, b = 4, choice;

double e = 0.001, x{};

do

{

cout << "Введите номер уравнения (1 или 2): ";

cin >> choice;

} while (choice != 1 && choice != 2); // Проверка корректности ввода номера уравнения

if (choice == 1)

{

cout << "Корень уравнения: " << metodD(f, a, b, e, choice) << endl; // Вызов метода дихотомии для уравнения f(x)

}

if (choice == 2)

{

cout << "Корень уравнения: " << metodD(s, a, b, e, choice) << endl; // Вызов метода дихотомии для уравнения s(x)

}

system("pause");

}

// Функция метода дихотомии

double metodD(double(\*func)(double), double a, double b, double e, int ch)

{

double x;

while (abs(a - b) > 2 \* e) // Цикл вычисления корня с заданной точностью

{

x = (a + b) / 2; // Вычисление середины отрезка

if (ch == 1)

{

if (func(x) \* func(a) <= 0)

{

b = x;

}

else

{

a = x;

}

}

if (ch == 2)

{

if (func(x) \* func(a) <= 0)

{

b = x;

}

else

{

a = x;

}

}

}

return x; // Возвращение найденного корня

}

// Уравнение f(x)

double f(double x)

{

return 2 \* x + pow(x, 3) - 7;

}

// Уравнение s(x)

double s(double x)

{

return exp(x) + 2 \* x;

}





Доп. Задачи

В соответствии со своим вариантом написать программы по условиям задач из таблицы ниже. Программа должна содержать функцию пользователя с переменным числом параметров и не менее трех обращений к ней с различным количеством параметров.

**Вариант 5**



#include <iostream>

#include <cstdarg>

using namespace std;

// Объявление прототипа функции, определяющей количество чисел, являющихся точными квадратами

int kvadr(int, ...);

int main()

{

setlocale(LC\_CTYPE, "rus"); // Установка русской локали для консоли

// Вызов функции kvadr с разными наборами чисел

cout << "Количество чисел, являющихся точными квадратами: " << kvadr(5, 2, 169, 12, 16, 25) << endl;

cout << "Количество чисел, являющихся точными квадратами: " << kvadr(8, 4, 36, 27, 54, 89, 49, 10, 121) << endl;

cout << "Количество чисел, являющихся точными квадратами: " << kvadr(6, 56, 98, 100, 400, 45, 89) << endl;

system("pause");

}

int kvadr(int size, ...)

{

int\* list;

list = new int[40]; // Создание динамического массива

// Заполнение массива list квадратами чисел от 1 до 40

for (int i = 0; i < 40; i++)

{

list[i] = (i + 1) \* (i + 1);

}

va\_list args; // Список аргументов переменной длины

va\_start(args, size); // Инициализация списка аргументов

int count = 0, buffer;

// Перебор аргументов переменной длины

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

{

buffer = va\_arg(args, int);

// Поиск совпадений среди квадратов чисел в массиве list

for (int k = 0; k < 40; k++)

{

if (buffer == list[k])

{

count++;

break;

}

}

}

delete[] list; // Освобождение памяти от динамического массива

va\_end(args); // Завершаем работу со списком аргументов

return count; // Функция возвращает количество точных квадратов

}

**Вариант 7**



#include <iostream>

#include <cstdarg>

using namespace std;

// Объявление прототипа функции для поиска всех простых чисел в нескольких интервалах

void isPrimeNumber(int, ...);

int main()

{

// Проверяем несколько интервалов

isPrimeNumber(6, 1, 20, 15, 21, 39, 63);

isPrimeNumber(4, 8, 23, 40, 82);

isPrimeNumber(2, 30, 67);

system("pause");

}

void isPrimeNumber(int a, ...)

{

va\_list args; // Список аргументов переменной длины

va\_start(args, a); // Инициализация списка аргументов

bool isPrime;

int x, y;

// Перебираем все интервалы

for (int k = 1; k <= a / 2; ++k)

{

x = va\_arg(args, int); // Получаем начало интервала(a)

y = va\_arg(args, int); // Получаем конец интервала(b)

// Перебираем числа в текущем интервале

for (int n = x; n <= y; n++)

{

isPrime = true; // Предполагаем, что число является простым перед проверкой

// Проверяем частные случаи и делаем проверку на простоту

if (n <= 1) {

isPrime = false;

}

else if (n == 2) {

isPrime = true;

}

else if (n % 2 == 0) {

isPrime = false;

}

else {

// Проверяем делители до квадратного корня числа

for (int i = 3; i <= sqrt(n); i += 2) {

if (n % i == 0) {

isPrime = false;

break;

}

}

}

// Если число простое, выводим его

if (isPrime)

{

cout << n << " ";

}

}

cout << " "; // Разделяем результаты для разных интервалов

}

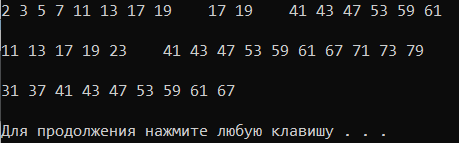
va\_end(args); // Завершаем работу со списком аргументов

printf("\n");

printf("\n");

}

// P.S. Алгоритм поиска простого числа частично взят из интернета и доработан



**Вариант 8**

****

#include <iostream>

#include <cstdarg>

using namespace std;

// Объявление прототипа функции для нахождения минимального числа из ряда

int mn(int, ...);

int main()

{

setlocale(LC\_CTYPE, "rus"); // Устанавливаем русскую локаль для консоли

// Вызов функций для разных рядов

cout << "Минимальное число из ряда: " << mn(6, 6, 7, -9, 11, 28, -40) << endl;

cout << "Минимальное число из ряда: " << mn(8, 1, 7, 6, 8, 2, -9, 11, 30) << endl;

cout << "Минимальное число из ряда: " << mn(7, -24, 76, 5, -6, 22, 8, 9) << endl;

system("pause");

}

// Функция для нахождения минимального числа из ряда

int mn(int size, ...)

{

va\_list args; // Список аргументов переменной длины

va\_start(args, size); // Инициализация списка аргументов

int min = va\_arg(args, int), buffer; // Получаем первый аргумент и обозначаем предположительный минимум

for (int i = 1; i < size; i++)

{

buffer = va\_arg(args, int); // Получаем следующий аргумент

if (buffer < min)

{

min = buffer; // Если текущий аргумент меньше минимума, обновляем минимум

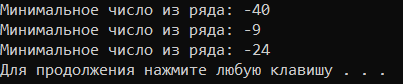
}

}

va\_end(args); // Завершаем работу со списком аргументов

return min; // Возвращаем найденный минимум

}

****

В соответствии со своим вариантом отделить корни двух уравнений и вычислить их методом дихотомии для исходных данных из таблицы, приведенной ниже. Точность вычислений принять равной e = 0,001 для всех вариантов.

Операторы метода вычисления корня оформить в виде функции пользователя, уравнения записать также в виде функций пользователя.

В главной функции предусмотреть ввод исходных данных, обращения к функции, реализующей метод дихотомии для двух уравнений. В процессе выполнения программы определить корни двух уравнений. Использовать указатель на функцию.

**Вариант 5**

****

#include <iostream>

using namespace std;

// Объявление прототипов функций

double metodD(double (\*)(double), double, double, double, int);

double f(double);

double s(double);

int main()

{

setlocale(LC\_CTYPE, "rus"); // Установка русской локали для консоли

int a = 1, b = 4, choice;

double e = 0.001, x{};

do

{

cout << "Введите номер уравнения (1 или 2): ";

cin >> choice;

} while (choice != 1 && choice != 2); // Проверка корректности ввода номера уравнения

if (choice == 1)

{

cout << "Корень уравнения: " << metodD(f, a, b, e, choice) << endl; // Вызов метода дихотомии для уравнения f(x)

}

if (choice == 2)

{

cout << "Корень уравнения: " << metodD(s, a, b, e, choice) << endl; // Вызов метода дихотомии для уравнения s(x)

}

system("pause");

}

// Функция метода дихотомии

double metodD(double(\*func)(double), double a, double b, double e, int ch)

{

double x;

while (abs(a - b) > 2 \* e) // Цикл вычисления корня с заданной точностью

{

x = (a + b) / 2; // Вычисление середины отрезка

if (ch == 1)

{

if (func(x) \* func(a) <= 0)

{

b = x;

}

else

{

a = x;

}

}

if (ch == 2)

{

if (func(x) \* func(a) <= 0)

{

b = x;

}

else

{

a = x;

}

}

}

return x; // Возвращение найденного корня

}

// Уравнение f(x)

double f(double x)

{

return exp(x) - 3 - 1 / x;

}

// Уравнение s(x)

double s(double x)

{

return 0.2 - pow(x, 2);

}

****

****

**Вариант 7**

****

#include <iostream>

using namespace std;

// Объявление прототипов функций

double metodD(double (\*)(double), double, double, double, int);

double f(double);

double s(double);

int main()

{

setlocale(LC\_CTYPE, "rus"); // Установка русской локали для консоли

int a = 1, b = 4, choice;

double e = 0.001, x{};

do

{

cout << "Введите номер уравнения (1 или 2): ";

cin >> choice;

} while (choice != 1 && choice != 2); // Проверка корректности ввода номера уравнения

if (choice == 1)

{

cout << "Корень уравнения: " << metodD(f, a, b, e, choice) << endl; // Вызов метода дихотомии для уравнения f(x)

}

if (choice == 2)

{

cout << "Корень уравнения: " << metodD(s, a, b, e, choice) << endl; // Вызов метода дихотомии для уравнения s(x)

}

system("pause");

}

// Функция метода дихотомии

double metodD(double(\*func)(double), double a, double b, double e, int ch)

{

double x;

while (abs(a - b) > 2 \* e) // Цикл вычисления корня с заданной точностью

{

x = (a + b) / 2; // Вычисление середины отрезка

if (ch == 1)

{

if (func(x) \* func(a) <= 0)

{

b = x;

}

else

{

a = x;

}

}

if (ch == 2)

{

if (func(x) \* func(a) <= 0)

{

b = x;

}

else

{

a = x;

}

}

}

return x; // Возвращение найденного корня

}

// Уравнение f(x)

double f(double x)

{

return pow(x, 3) + 2 \* x - 4;

}

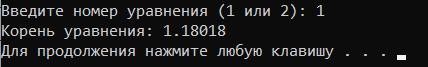
// Уравнение s(x)

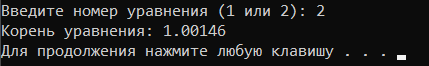
double s(double x)

{

return pow(x, 2) - 1;

}

****

****

**Вариант 8**

****

#include <iostream>

using namespace std;

// Объявление прототипов функций

double metodD(double (\*)(double), double, double, double, int);

double f(double);

double s(double);

int main()

{

setlocale(LC\_CTYPE, "rus"); // Установка русской локали для консоли

int a = 1, b = 4, choice;

double e = 0.001, x{};

do

{

cout << "Введите номер уравнения (1 или 2): ";

cin >> choice;

} while (choice != 1 && choice != 2); // Проверка корректности ввода номера уравнения

if (choice == 1)

{

cout << "Корень уравнения: " << metodD(f, a, b, e, choice) << endl; // Вызов метода дихотомии для уравнения f(x)

}

if (choice == 2)

{

cout << "Корень уравнения: " << metodD(s, a, b, e, choice) << endl; // Вызов метода дихотомии для уравнения s(x)

}

system("pause");

}

// Функция метода дихотомии

double metodD(double(\*func)(double), double a, double b, double e, int ch)

{

double x;

while (abs(a - b) > 2 \* e) // Цикл вычисления корня с заданной точностью

{

x = (a + b) / 2; // Вычисление середины отрезка

if (ch == 1)

{

if (func(x) \* func(a) <= 0)

{

b = x;

}

else

{

a = x;

}

}

if (ch == 2)

{

if (func(x) \* func(a) <= 0)

{

b = x;

}

else

{

a = x;

}

}

}

return x; // Возвращение найденного корня

}

// Уравнение f(x)

double f(double x)

{

return pow(x, 3) + 3 \* x - 1;

}

// Уравнение s(x)

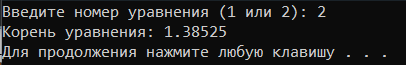
double s(double x)

{

return exp(x) - 4;

}

****

****