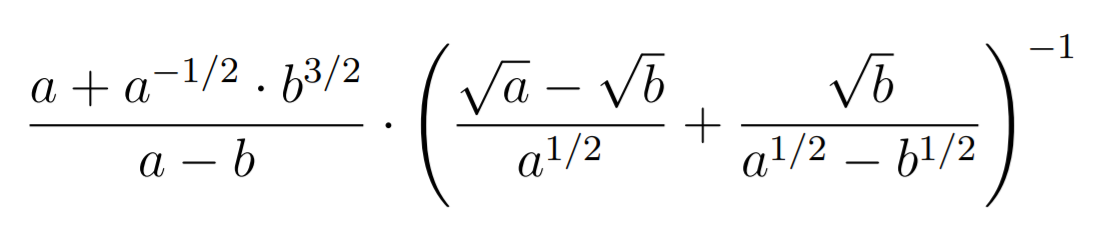
**Лабораторная работа № 3 (Часть 1)**

**Числа с плавающей точкой. Циклы.**

**Задача 1**

Постановка задачи:

Математическая модель: представлена в условии задачи

Список идентификаторов:

|  |  |  |
| --- | --- | --- |
| Имя | Тип | Смысл |
| a | float | Вводимое число |
| b | float | Вводимое число |
| res | float | Результат |

Код программы:

#include <stdio.h>

#include <math.h>

int main(void) {

    float a, b, res;

    printf("a = ");

    scanf("%f", &a);

    printf("b = ");

    scanf("%f", &b);

    res = (a + powf(a, -1./2) \* powf(b, 3./2) / (a - b)) \* pow(((sqrtf(a) - sqrtf(b)) / powf(a, 1./2) + sqrtf(b) / (powf(a, 1./2) - powf(b, 1./2))), -1);

    printf("%f", res);

    return 0;

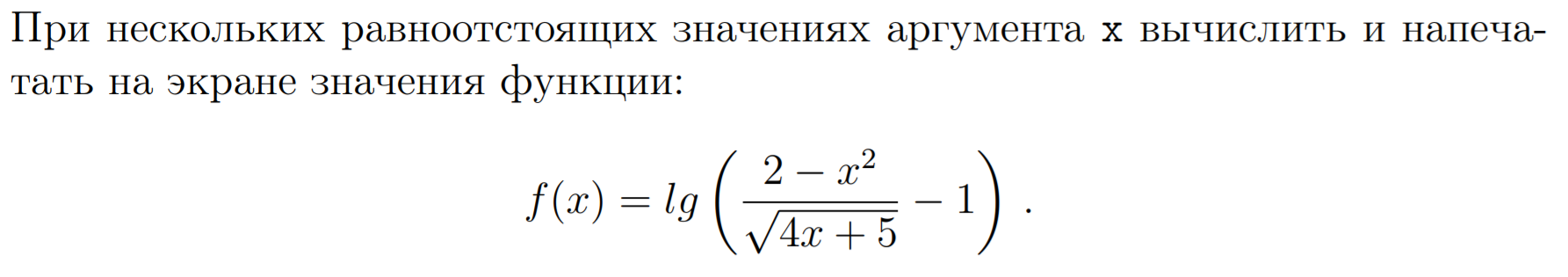
}

Результат:



**Задача 2**

Постановка задачи:



Математическая модель: представлена в условии задачи

Список идентификаторов:

|  |  |  |
| --- | --- | --- |
| Имя | Тип | Смысл |
| x | float | Аргумент |
| h | float | Шаг |
| f | float | Значение функции в т. Х |

Код программы:

#include <stdio.h>

#include <math.h>

/\* Ограничение на X:

-1.25 < x < -0.4

При этом x != -1

Возьмём начальное значение Х = -1.2

Границой будет 0.45 с шагом 0.15

\*/

int main(void) {

    float x, h, f;

    x = -1.2;

    h = 0.15;

    while (x <= -0.45) {

        f = logf((2 - x \* x) / sqrtf(4 \* x + 5) - 1);

        printf("f(%f) = %f\n", x, f);

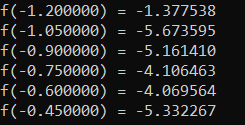
        x += h;

    }

    return 0;

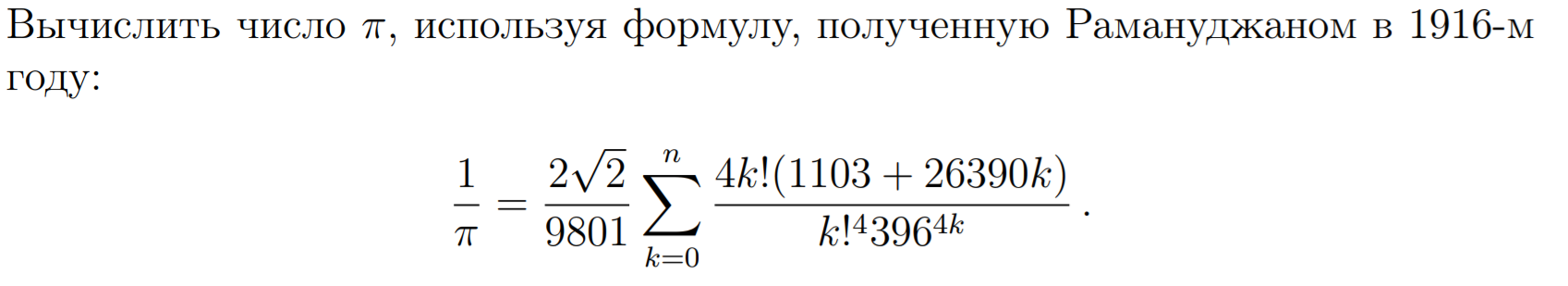
}

Результат:



**Задача 3**

Постановка задачи



Математическая модель: представлена в условии задачи

Список идентификаторов:

|  |  |  |
| --- | --- | --- |
| Имя | Тип | Смысл |
| sum | double | Сигма |
| res | double | Результат |
| f | long int | Факториал от К |

Код программы:

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

long int factorial(long int m) {

    if (m == 0)

        return 1;

    else {

        m = m \* factorial(m - 1);

        return m;

    }

}

void result(int n) {

    double sum, res;

    long int f;

    sum = 0;

    for (int k = 0; k <= n; k++) {

        f = factorial(k);

        sum += factorial(4 \* k) \* (1103 + 26390 \* k) / pow(f, 4) / pow(396, 4\*k);

    }

    res = 9801 / (2 \* sqrt(2) \* sum);

    printf("%.14Lf\n", res);

}

int main(void) {

    result(2);

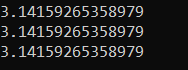
    result(3);

    result(5);

    return 0;

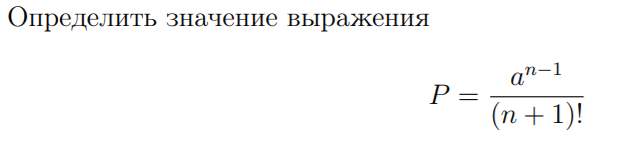
}

Результат:



**Задача 4**

Постановка задачи:



Математическая модель: представлена в условии задачи

Список идентификаторов:

|  |  |  |
| --- | --- | --- |
| Имя | Тип | Смысл |
| n | int | Вводимое число |
| a | int | Вводимое число |
| p | double | Искомое число |

Код программы:

#include <stdio.h>

#include <math.h>

long factorial(long m) {

    if (m == 1 || m == 0)

        return m = 1;

    else {

        m = m \* factorial(m - 1);

        return m;

    }

}

int main(void) {

    int n, a;

    double p;

    printf("a = ");

    scanf("%d", &a);

    printf("n = ");

    scanf("%d", &n);

    p = (pow(a, (n-1))) / factorial(n+1);

    printf("%lf", p);

    return 0;

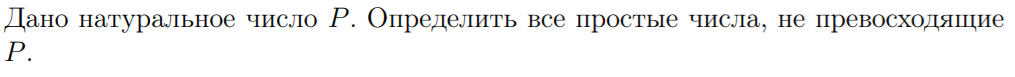
}

Результат:



**Задача 5**

Постановка задачи:

Математическая модель:

P – просто, если для любого 1 < x < P остаток от деления P на x будет > 0

Список идентификаторов:

|  |  |  |
| --- | --- | --- |
| Имя | Тип | Смысл |
| p | int | Вводимое число |
| k | int | Очередное число |
| i | int | Аргумент цикла |
| j | int | Аргумент цикла |

Код программы:

#include <stdio.h>

#include <math.h>

/\*

Время работы алгоритма будет составлять О(sqrt(n)), однако

это эффективно по памяти. Наиболее быстрый алгоритм

это «Решето Эратосфена», однако оно не эффективно по

памяти для относительно больших чисел

\*/

int main() {

    int p, k = 0;

    printf("p = ");

    scanf("%d", &p);

    for (int i = 3; i <= p; i++) {

        for (int j = 2; j\*j <= i; j++) {

            if (i % j == 0)

                k += 1;

        }

        if (k == 0)

            printf("%d ", i);

        else

            k = 0;

    }

    return 0;

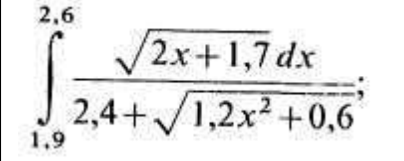
}

Результат:



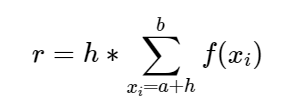
**Часть 2 (вычисление интегралов)**

Вычислить интеграл



**1. Методом правых частей**

Математическая модель:



Список идентификаторов:

|  |  |  |
| --- | --- | --- |
| Имя | Тип | Смысл |
| a | double | Нижняя граница |
| b | double | Верхняя граница |
| x | double | Аргумент |
| sum | double | Сигма |
| h | double | Шаг |
| res | double | Результат |
| n | int | Количество разбиений |

Код программы:

#include <stdio.h>

#include <math.h>

/\*

Интеграл:

sqrt(2x + 1.7) / (2.4 + sqrt(1.2x\*x + 0.6))

Отрезок:

[1.9 ; 2.6]

Метод прямоугольников правых частей

\*/

void integral(int n) {

    double a = 1.9, b = 2.6;

    double x, sum, h, res;

    sum = 0;

    h = (b - a) / n;

    x = a + h;

    while (x <= b) {

        sum += sqrt(2 \* x + 1.7) / (2.4 + sqrt(1.2 \* x \* x + 0.6));

        x = x + h;

    }

    res = h \* sum;

    printf("n = %d, integral = %lf\n", n, res);

}

int main(void) {

    integral(10);

    integral(100);

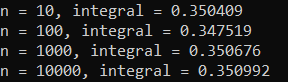
    integral(1000);

    integral(10000);

 return 0;

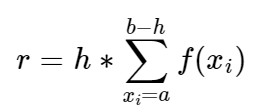
}

Результат:



**2. Методом левых частей**

Математическая модель:



Список идентификаторов:

|  |  |  |
| --- | --- | --- |
| Имя | Тип | Смысл |
| a | double | Нижняя граница |
| b | double | Верхняя граница |
| x | double | Аргумент |
| sum | double | Сигма |
| h | double | Шаг |
| res | double | Результат |
| n | int | Количество разбиений |

Код программы:

#include <stdio.h>

#include <math.h>

/\*

Интеграл:

sqrt(2x + 1.7) / (2.4 + sqrt(1.2x\*x + 0.5))

Отрезок:

[1.9 ; 2.6]

Метод прямоугольников левых частей

\*/

void integral(int n) {

    double a = 1.9;

    double b = 2.6;

    double x, sum, h, res;

    sum = 0;

    h = (b - a) / n;

    x = a;

    while (x <= (b - h)) {

        sum += sqrt(2 \* x + 1.7) / (2.4 + sqrt(1.2 \* x \* x + 0.5));

        x += h;

    }

    res = h \* sum;

    printf("n = %d, integral = %lf\n", n, res);

}

int main(void) {

    integral(10);

    integral(100);

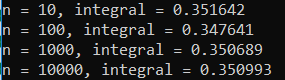
    integral(1000);

    integral(10000);

    return 0;

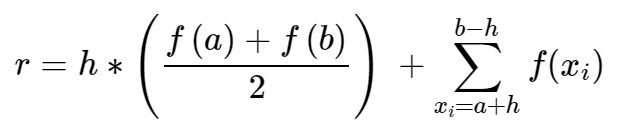
}

Результат:



**3. Методом трапеций**

Математическая модель:



Список идентификаторов:

|  |  |  |
| --- | --- | --- |
| Имя | Тип | Смысл |
| a | double | Нижняя граница |
| b | double | Верхняя граница |
| x | double | Аргумент |
| sum | double | Сигма |
| h | double | Шаг |
| res | double | Результат |
| n | int | Количество разбиений |

Код программы:

/\*

Интеграл:

sqrt(2x + 1.7) / (2.4 + sqrt(1.2x\*x + 0.5))

Отрезок:

[1.9 ; 2.6]

Метод трапеций

\*/

#include <stdio.h>

#include <math.h>

double function(double y) {

    return (sqrt(2 \* y + 1.7) / (2.4 + sqrt(1.2 \* y \* y + 0.5)));

}

void integral(int n) {

    double a = 1.9;

    double b = 2.6;

    double x, sum, h, res;

    sum = 0;

    h = (b - a) / n;

    x = a + h;

    while (x <= (b - h)) {

        sum += function(x);

        x += h;

    }

    res = h \* ((function(a) + function(b)) / 2 + sum);

    printf("n = %d, integral = %lf\n", n, res);

}

int main(void) {

    integral(10);

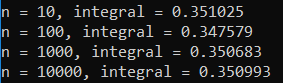
    integral(100);

    integral(1000);

    integral(10000);

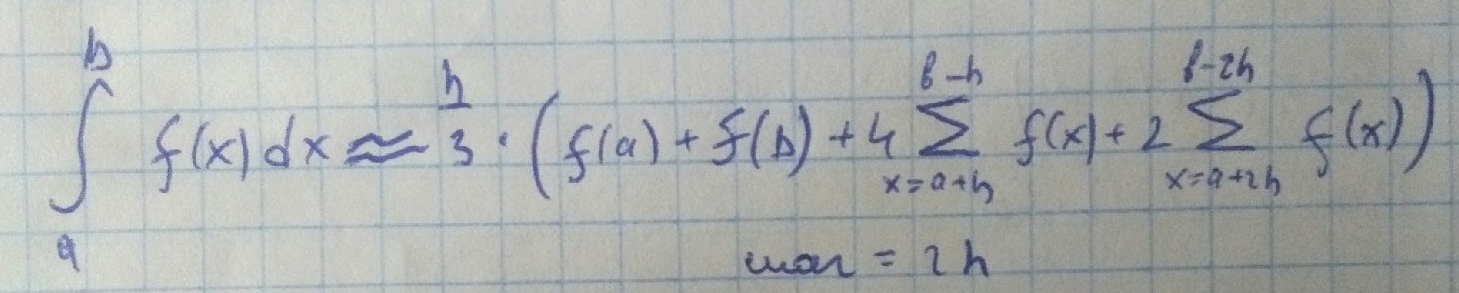
}

Результат:



**4. Методом Симпсона**

Математическая модель:



Список идентификаторов:

|  |  |  |
| --- | --- | --- |
| Имя | Тип | Смысл |
| a | double | Нижняя граница |
| b | double | Верхняя граница |
| x | double | Аргумент |
| sum\_1 | double | Сумма нечётных |
| sum\_2 | double | Сумма чётных |
| h | double | Шаг |
| res | double | Результат |
| n | int | Количество разбиений |

Код программы:

/\*

Интеграл:

sqrt(2x + 1.7) / (2.4 + sqrt(1.2x\*x + 0.5))

Интервал:

[1.9 ; 2.6]

Метод Симпсона

\*/

#include <stdio.h>

#include <math.h>

double function(double y) {

    return (sqrt(2 \* y + 1.7) / (2.4 + sqrt(1.2 \* y \* y + 0.5)));

}

void integral(int n) {

    double a = 1.9;

    double b = 2.6;

    double x, sum\_1, sum\_2, h, res;

    sum\_1 = 0; sum\_2 = 0;

    h = (b - a) / 2 / n;

    x = a + h;

    while (x <= b) {

        sum\_1 += function(x);

        x += 2 \* h;

    }

    x = a + 2 \* h;

    while (x <= b - h) {

        sum\_2 += function(x);

        x += 2 \* h;

    }

    res = h / 3 \* (function(a) + function(2 \* b) + 4 \* sum\_1 + 2 \* sum\_2);

    printf("n = %d, integral = %lf\n", n, res);

}

int main(void) {

    integral(10);

    integral(100);

    integral(1000);

    integral(10000);

}

Результат:

