Лабораторная работа 6

**Численное интегрирование функции.**

**Цель работы**. На примере разработки программы для численного интегрирования функции с заданной точностью методом прямоугольников и методом трапеций освоить следующие приемы программирования:

- передача в функцию параметров «по значению» и «по адресу»;

- передача в функцию имени функции;

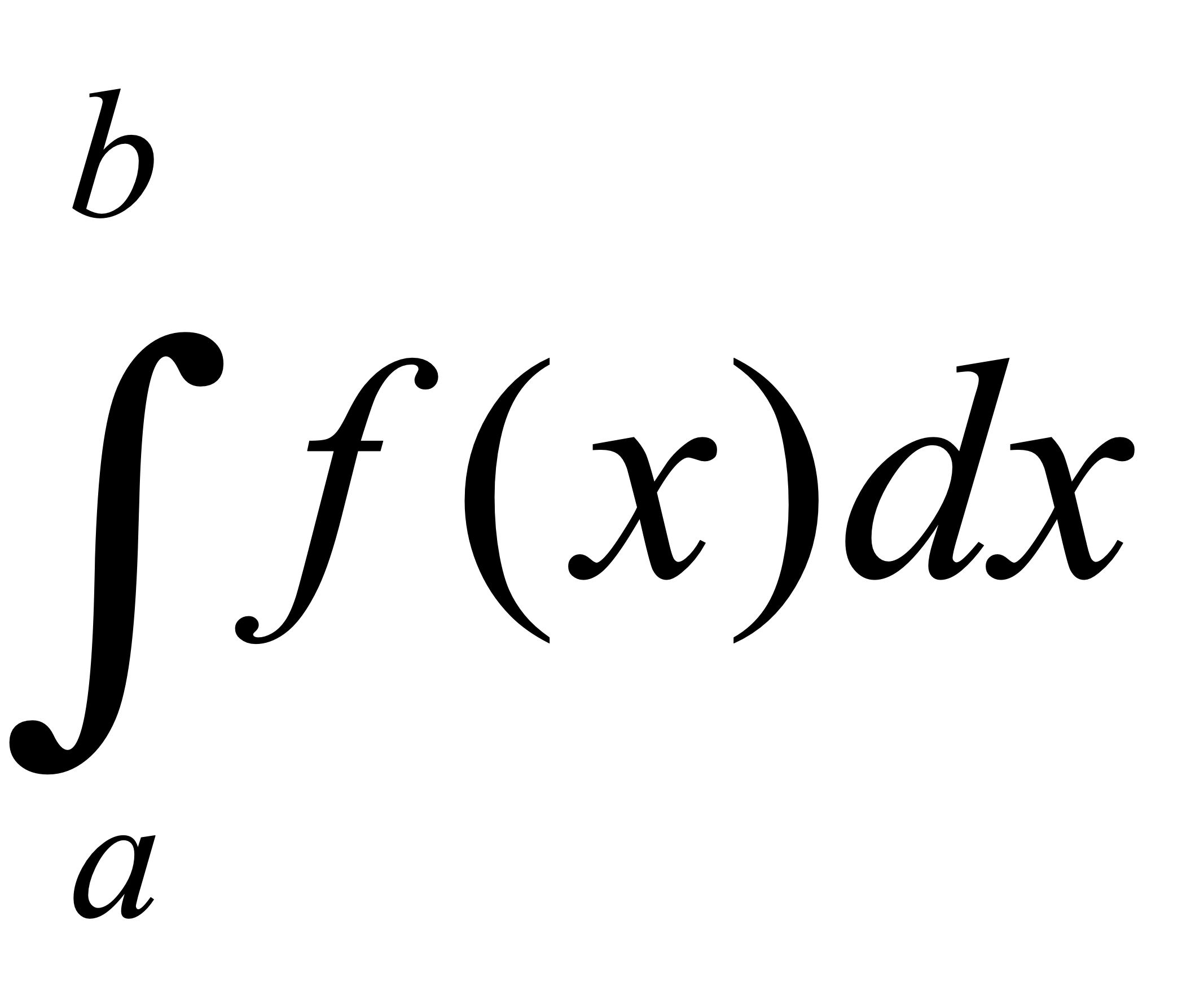
- передача одномерных массивов в функцию;

- объединение разнородных данных в структуру;

- использование массивов из элементов типа структура;

**Задание.**

1. Численное интегрирование функции с заданной точностью методом прямоугольников.

Вычислить определённый интеграл в пределах от ***a*** до ***b*** для четырех функций f1 = x, f2 = sin( 22 \* x ), f3 = x4 и f4 = arctg(x).

Вычисление интеграла оформить в виде функции IntRect.

Вычисления выполнить для пяти значений точности: 0.01, 0.001, 0.0001, 0.00001 и 0.000001.

Исследовать быстродействие алгоритма в зависимости от подынтегральной функции и требуемой точности (быстродействие алгоритма можно оценить числом элементарных прямоугольников ***n***).

Результаты представить в виде 5 таблиц, по одной таблице для каждого значения точности. В каждой таблице выводить данные для всех четырех функций.

Для печати таблицы результатов использовать функцию

void PrintTabl(resultToPrint\* i\_prn, int k), приведенную в приложении 2.

Здесь i\_prn[] – массив структур типа resultToPrint размерностью k.

Вид таблицы приведен в Приложении 1.

2. Выполнить п.1, используя для интегрирования метод трапеций. Вычисление интеграла оформить в виде функции IntTrap.

Для печати таблиц результатов использовать ту же функцию, что и в методе прямоугольников.

**Указания по выполнению работы.**

Алгоритм метода Дарбу-Римана аналогичен алгоритму метода прямоугольников, только на каждом шаге вычисляются две суммы – верхняя (S2) и нижняя (S1):

f1 = f( x ); *// значение функции на левой границе отрезка*

f2 = f( x + dx ); *// значение функции на правой границе*

if(f1 <= f2 ) *// возрастающий участок*

{ S1 += f1 \* dx; *// нижняя сумма*

S2 += f2 \* dx; *// верхняя сумма*

*}*

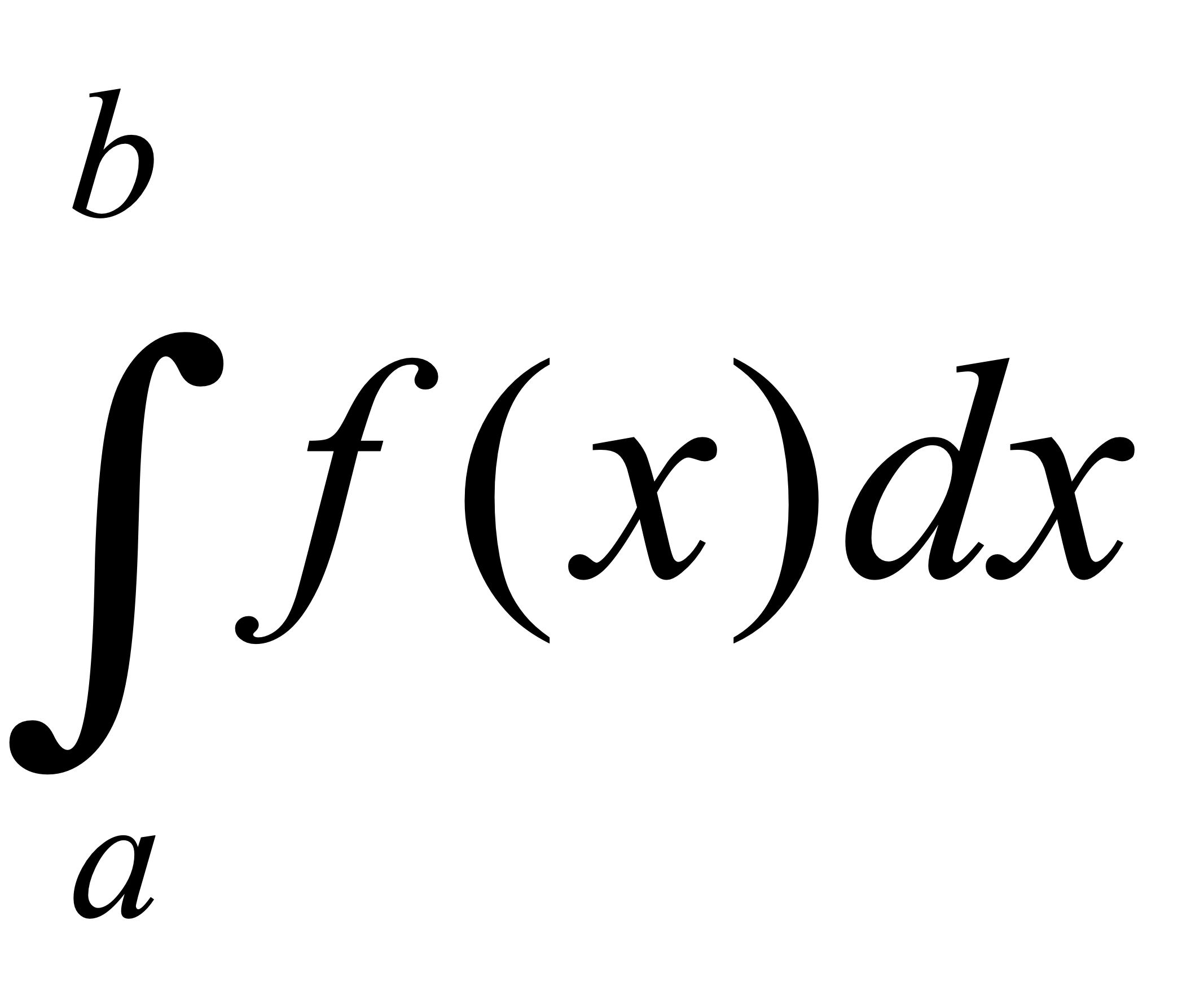
else *// убывающий участок*

{ S2 += f1 \* dx; *// верхняя сумма*

S1 += f2 \* dx; *// нижняя сумма*

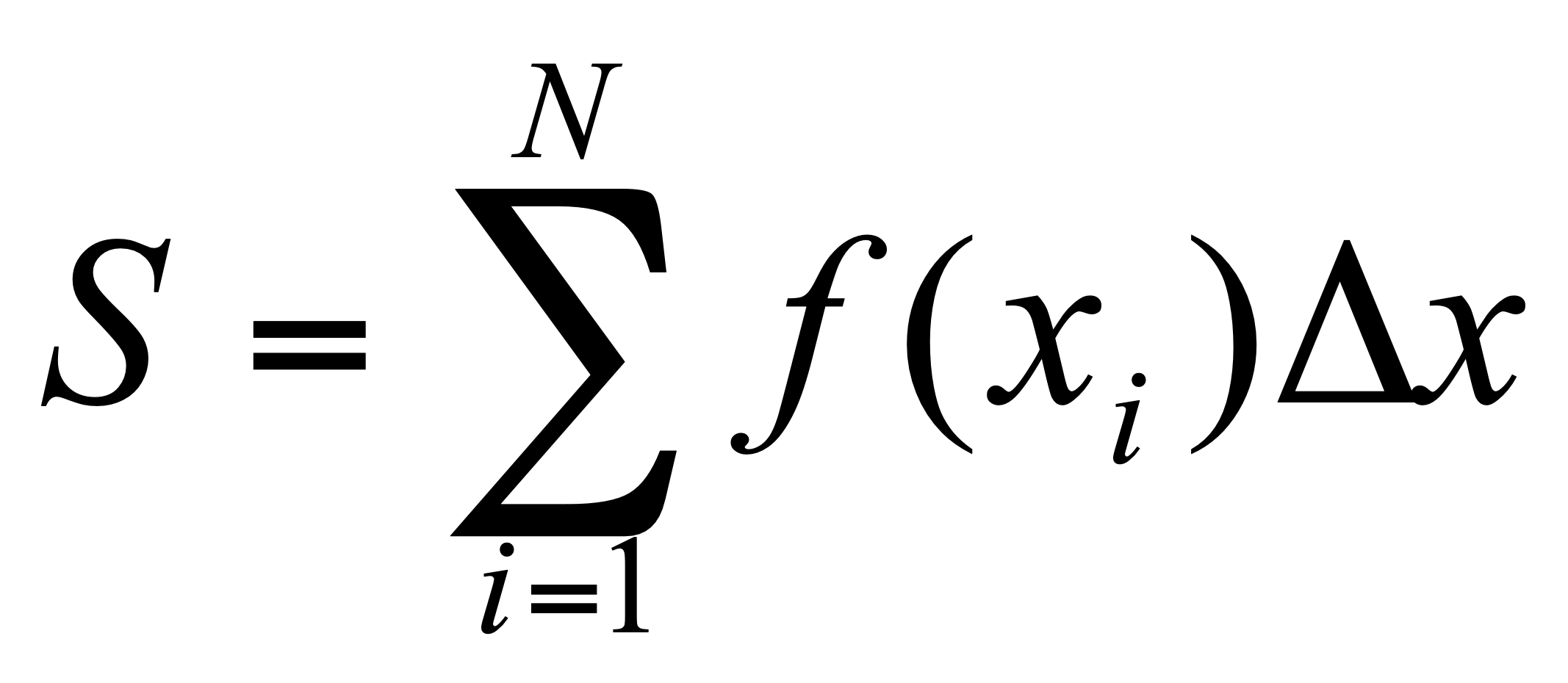
*}*

Вычисления прекращаются, если |S2-S1| < eps.

Задача вычисления определенного интеграла формулируется следующим образом: вычислить  для подынтегральный функции *f(x)* при заданных значениях пределов интегрирования ***a, b*** и требуемой точности ***eps***.

При численном интегрировании площадь под кривой заменяется суммой площадей «элементарных» прямоугольников с высотой, проведенной из середины основания.

Формула приближенного значения определенного интеграла представляется в виде

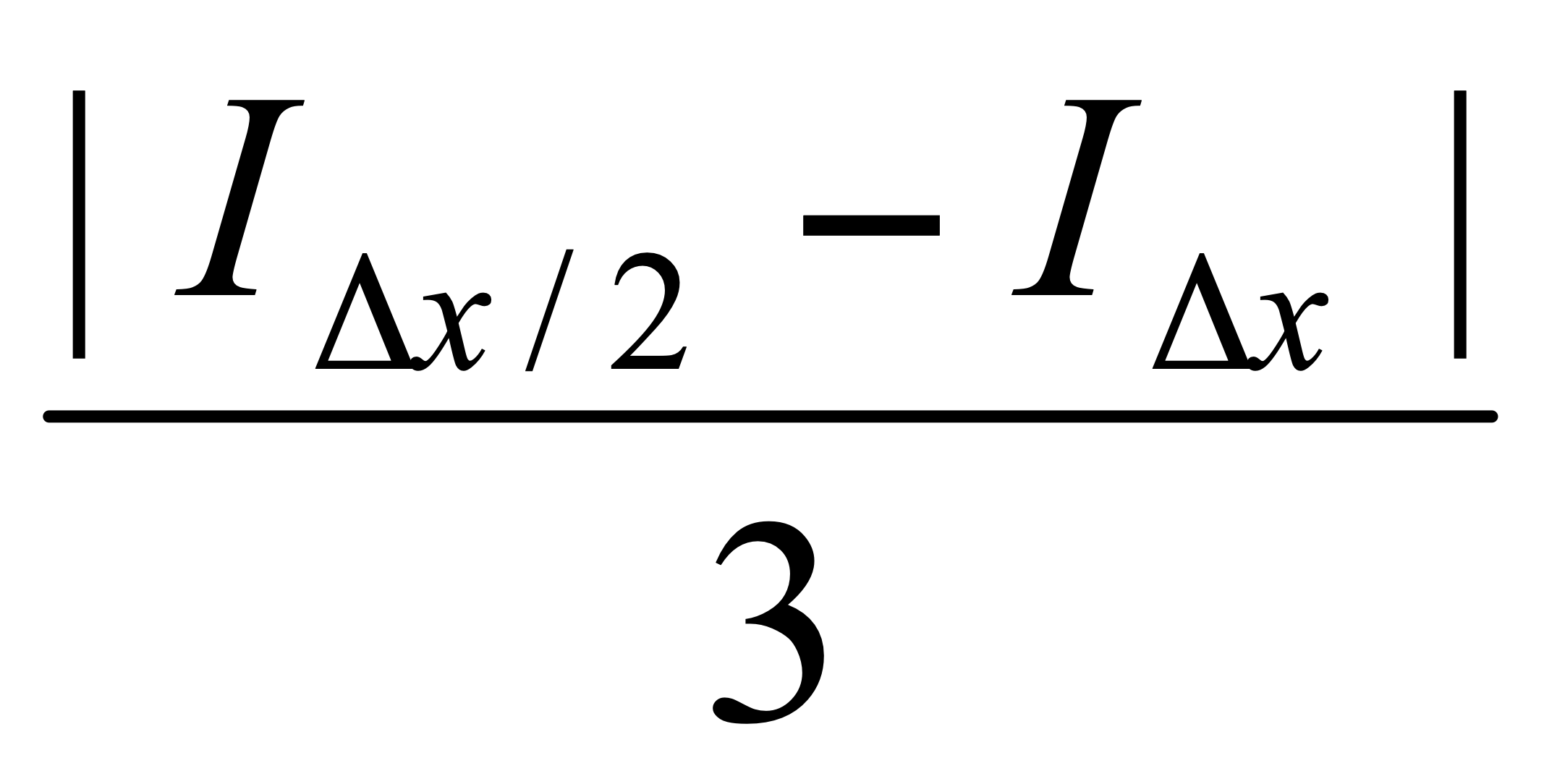
 ,

где: *xi = a +* D*x/2 + (i-1)*D*x; N -* число элементарных прямоугольников*.*

Для уменьшения объема вычислений множитель Dx следует вынести за знак суммы. Тогда в цикле нужно выполнять только суммирование, а затем полученную сумму один раз умножить на D*x*.

Для оценки погрешности вычисления интеграла на практике используют правило Рунге. Суть правила состоит в том, что выполняют вычисление интеграла с двумя разными шагами изменения переменной *х*, а затем сравнивают результаты и получают оценку точности. Наиболее часто используемое правило связано с вычислением интеграла дважды: с шагом D*x* и шагом D*x/2*.

Для методов прямоугольников и трапеций погрешность RD*x/2* вычисления интеграла с шагом D*x/2* оценивается следующей формулой:

|RD*x/2*| =, (1)

где ID*x/2* – значение интеграла, вычисленное с шагом D*x/2*; ID*x* – значение интеграла, вычисленное с шагом D*x.*

В программе вычисления интеграла с точностью ***eps***во внутреннем цикле находят значение определенного интеграла с шагом D*x/2*. Во внешнем цикле производится сравнение значений интегралов, вычисленных с шагами D*x* и D*x/2* соответственно. Если требуемая точность не достигнута, то число разбиений удваивается, а в качестве предыдущего значения интеграла берут текущее и вычисление интеграла выполняется при новом числе разбиений.

Вычисление интеграла оформить в виде функции IntRect, формальными параметрами которой являются:

*f* – имя интегрируемой функции,

*a, b* – границы интервала интегрирования,

*eps* – требуемая точность,

*n* – число прямоугольников, при котором достигнута требуемая точность (выходной).

Функция возвращает значение интеграла.

Прототип функции:

double IntRect(TPF f, double a, double b, double eps, int& n);

Здесь:

TPF – тип указателя на подынтегральную функцию:

typedef double (\*TPF)(double);

Для хранения и печати результатов вычислений используйте структуру, элементами которой являются наименование функции, значения интеграла (точное и вычисленное в виде суммы) и число «элементарных» прямоугольников ***n***, при котором достигнута требуемая точность. Точные значения, полученные аналитически, нужны для оценки правильности результатов численного интегрирования.

Так как в лабораторной работе требуется выполнять вычисление интеграла для четырех функций, для пяти значений точности для каждой функции и двумя методами, то для сокращения объема программы следует использовать циклы, а для обеспечения возможности реализации циклов обрабатываемые данные нужно хранить в массивах (массив указателей на функции, массив значений точности, массив структур для хранения и печати результатов вычислений).

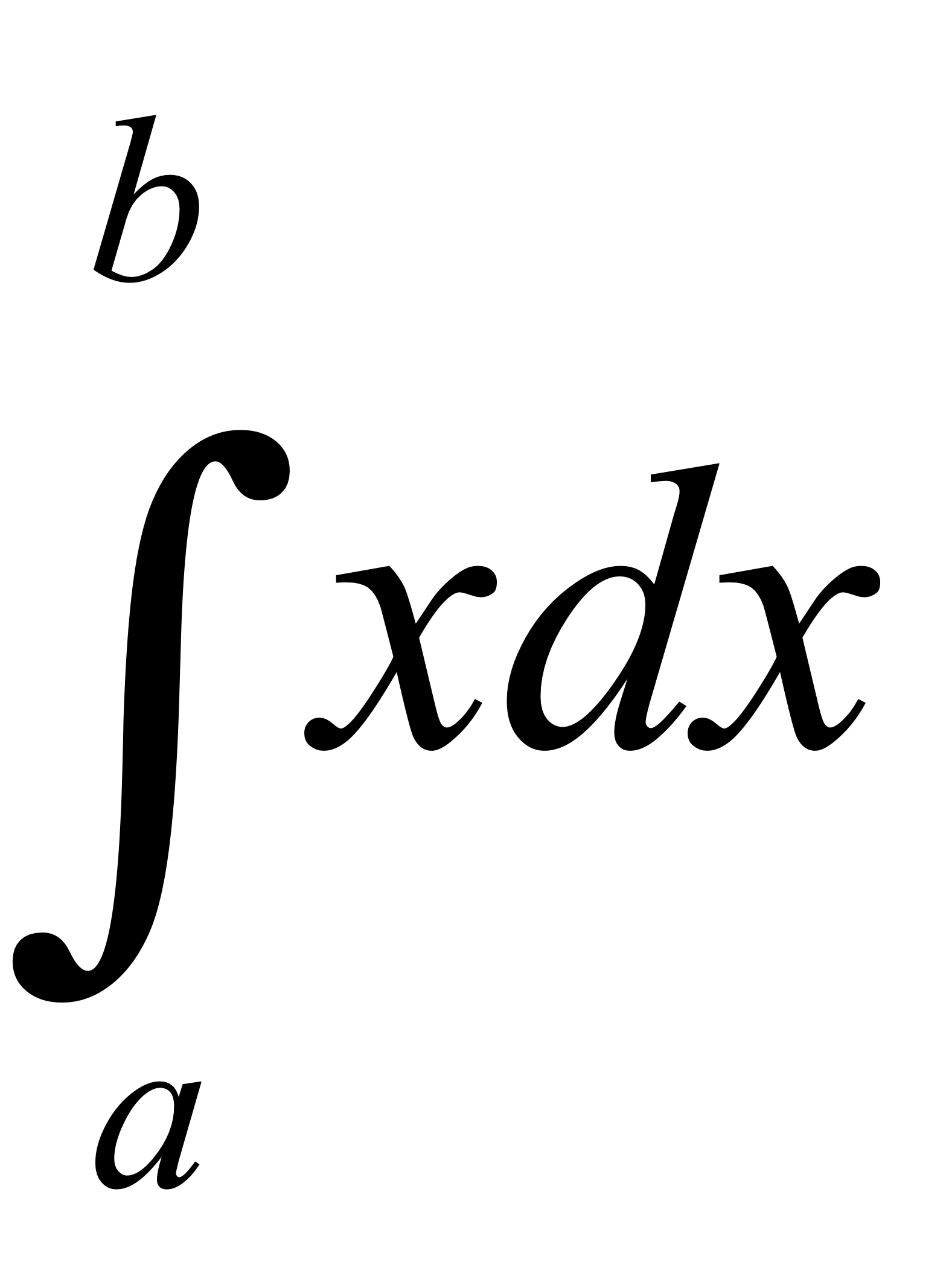
Алгоритм метода трапеций аналогичен алгоритму метода прямоугольников, только площадь элементарной трапеции вычисляется по формуле: Sт=dx\*(f(x)+f(x+dx))/2.

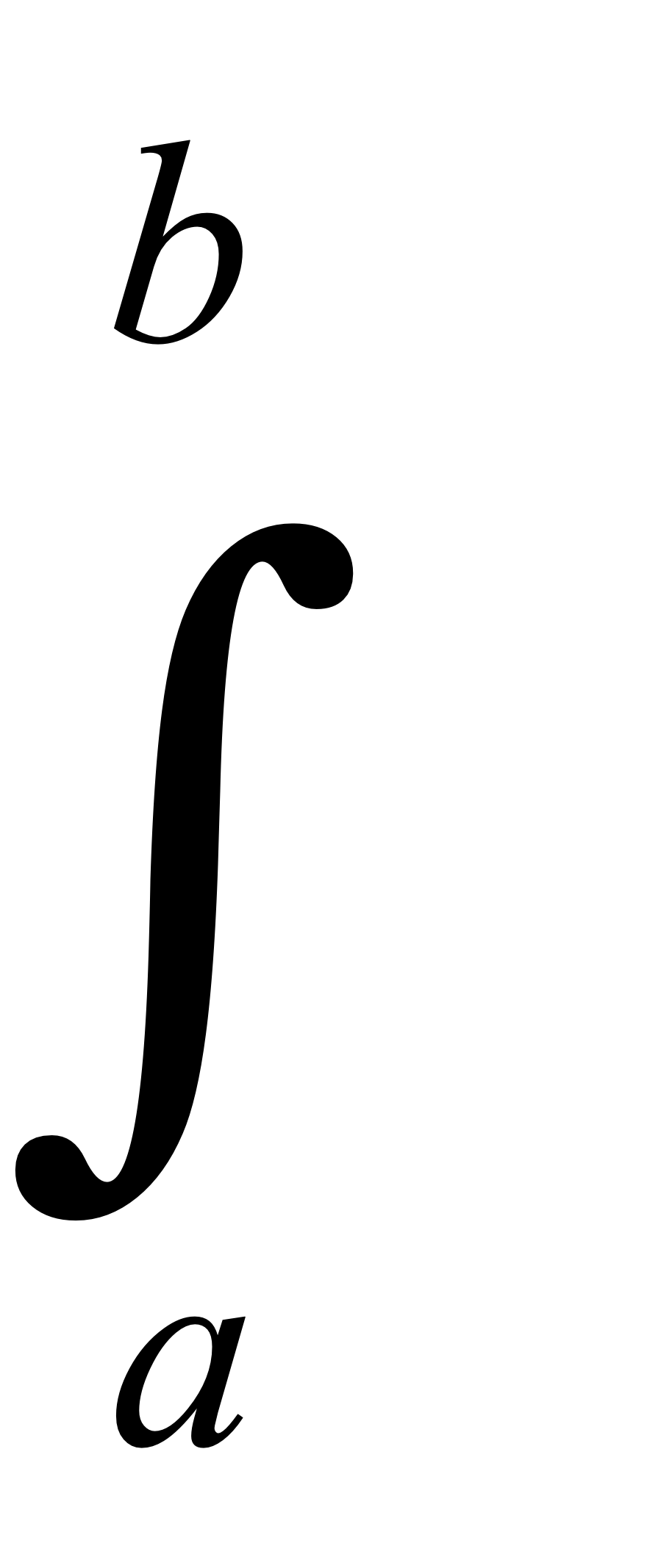
При этом значения функций на границах внутренних отрезков при вычислении интеграла используются дважды, а на границах интервала [a, b] - только один раз.

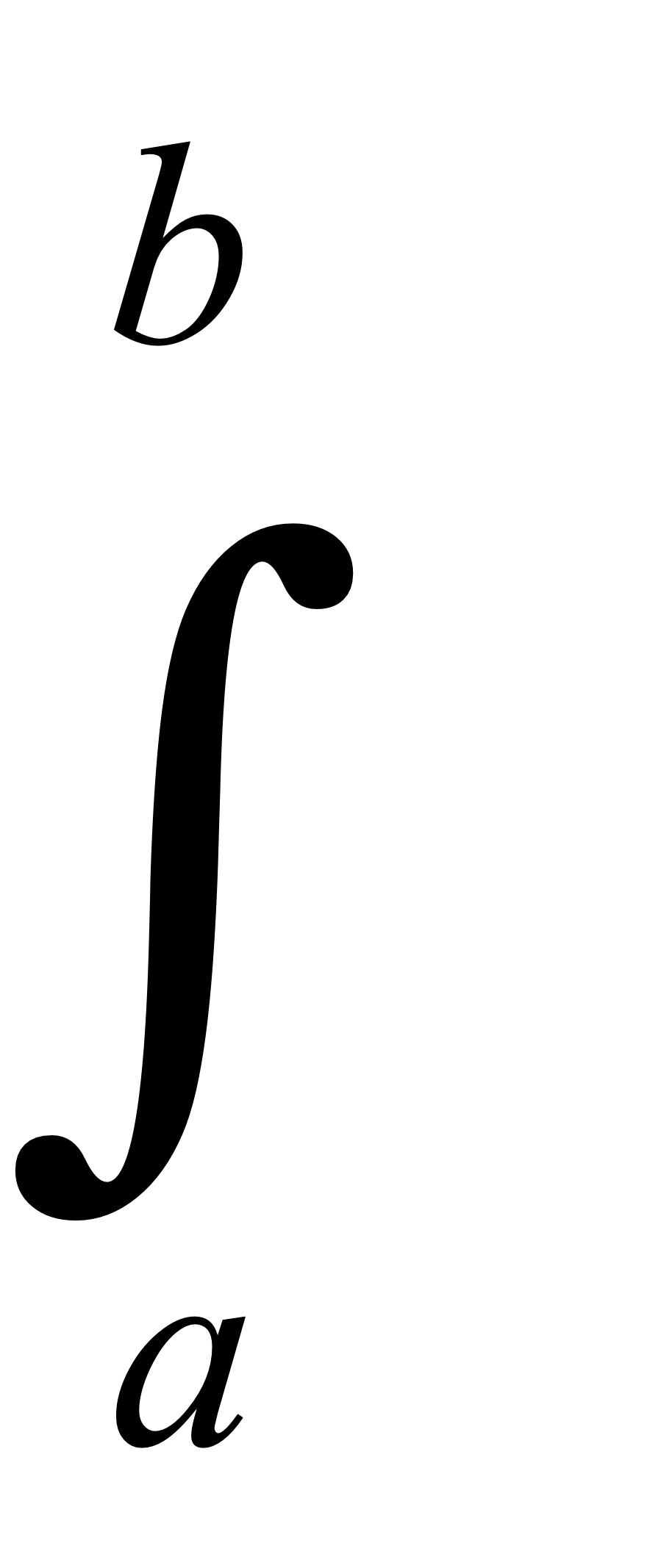
Прототип функции:

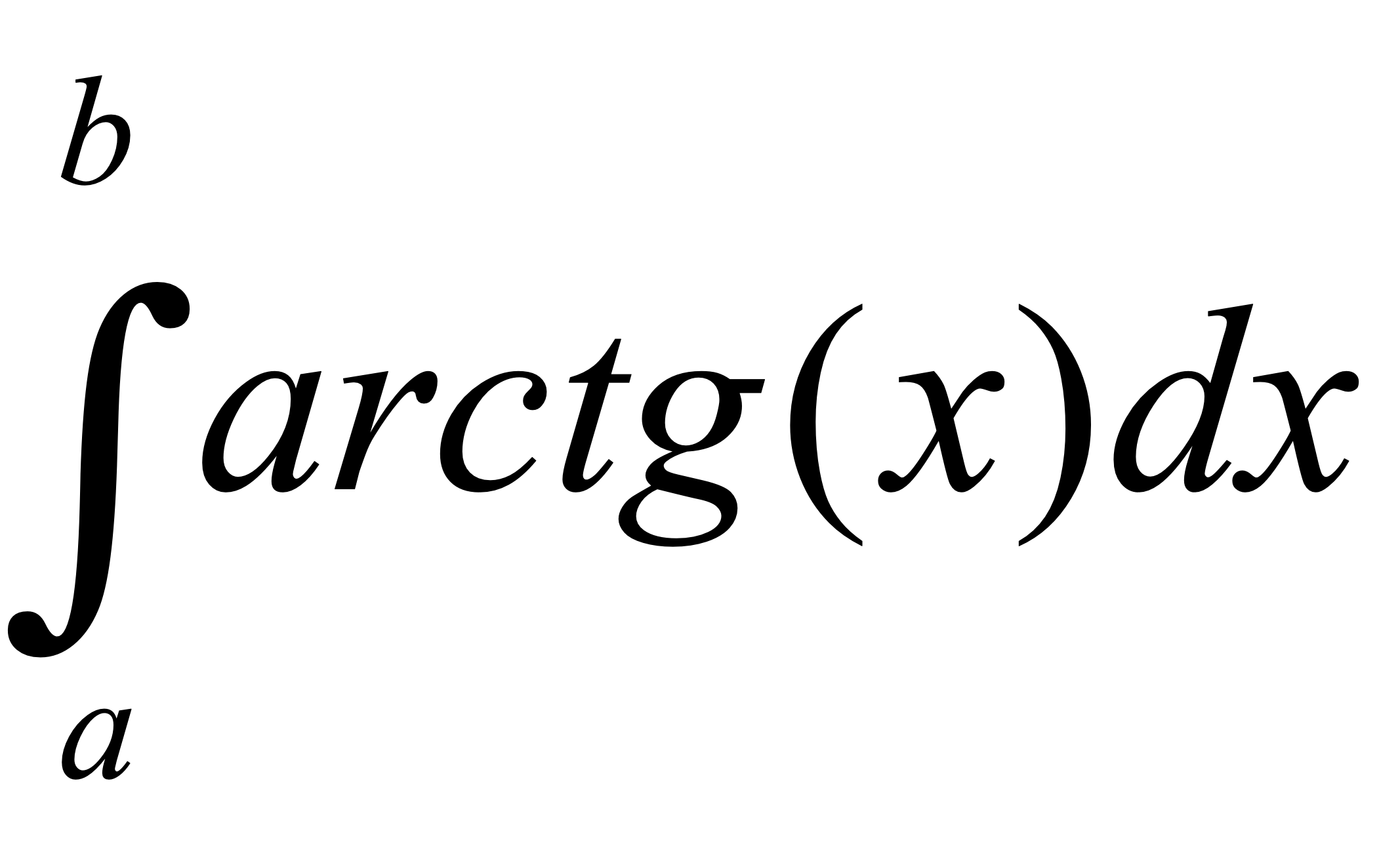
double IntTrap(TPF f, double a, double b, double eps, int& n);

**Формулы для вычисления точных значений интеграла:**

****=(b\*b - a\*a)/2.0;

*sin(22x)dx*=(cos(a\*22.0) - cos(b\*22.0))/22.0;

*x4dx*=(b\*b\*b\*b\*b - a\*a\*a\*a\*a)/5.0;

=b\*atan(b) - a\*atan(a) - (log(b\*b+1) - log(a\*a+1))/2.0;

**Примеры передачи в функцию в качестве параметров одномерных массивов и имен функций.**

Массивы и функции передаются в функцию через указатели.

Имя массива является указателем на его нулевой элемент. Указатель «ничего не знает» о длине массива и длина массива должна передаваться в функцию как параметр.

Имя функции указывает на первую команду кода функции.

*Передача одномерных массивов в функцию*

#include <iostream>

int sum(int \*a, int n);

int main() {

int n;

int a[]={1,2,3,4,5,6,7,8};

n=sizeof(a)/sizeof(int); *// Определение размерности*

*// инициализированного массива*

std::cout << " n = " << n <<std::endl;

cout << sum(a, n) << std::endl;

return 0;

}

int sum(int\* a, int n){ // В функцию передаются указатель на

//начало массив(имя массива a) и его

// размерность(n)

int s = 0;

int k = sizeof(a); //k – размер указателя (4 байта)

std::cout << " k = " << k << std::endl;

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

s += a[i];

return s;

}

*Передача имен функций в качестве параметров*

*/\*для удобочитаемости программы определяется новый тип*

*(тип пользователя)* ***PF*** *- указатель на функцию, которая имеет*

*один параметр типа int и не возвращает никакого значения\*/*

#include <iostream>

typedef void (\*PF)(int);

*//Определение функции f1*

void f1(PF pf) { *//функция получает в качестве параметра*

*// указатель типа PF*

pf(5); *//вызов функции через указатель*

*}*

void f(int i) {

std::cout << i <<std::endl;

}

int main() {

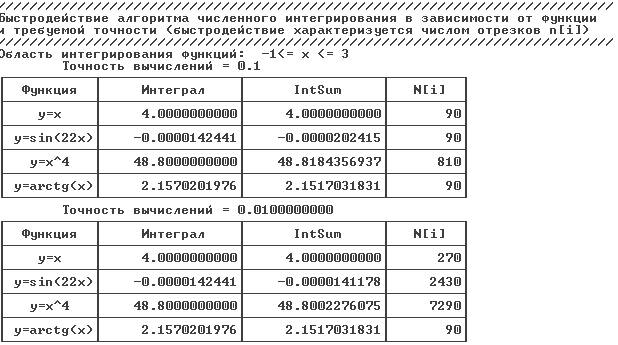
f1(f); *//Функция выведет на экран число 5*

return 0;

}

Приложение 1.

**Пример вывода таблицы результатов**



Приложение 2.

Функция для печати таблицы результатов

namespace {

const int numberOfTableColumns = 4; //число столбцов таблицы

const int maxWidthOfTableColumns = 18;

const int firstColumnWidth = 12; //ширина столбцов таблицы

const int secondColumnWidth = 18;

const int thirdColumnWidth = 18;

const int fourthColumnWidth = 10;

// Символы рамки в UTF-8

const char\* ul = "┌"; // верхний левый угол char(218)

const char\* ur = "┐"; // верхний правый угол char(191)

const char\* dl = "└"; // нижний левый угол char(192)

const char\* dr = "┘"; // нижний правый угол char(217)

//const std::string hz = u8"─"; // горизонтальная линия char(196)

const char\* vt = "│"; // вертикальная линия char(179)

const char\* cr = "┼"; // перекрестие char(194)

const char\* Td = "┬"; // Т-образный вниз char(197)

const char\* Tu = "┴"; // Т-образный вверх char(193)

const char\* Tr = "├"; // Т-образный вправо char(195)

const char\* Tl = "┤"; // Т-образный влево char(180)

}

struct resultToPrint { //данные для печати результатов интегрирования

char\* name; //название функции

double i\_sum; //значение интегральной суммы

double i\_toch; //точное значение интеграла

int n; //число разбиений области интегрирования при котором достигнута требуемая точность

};

void printTabl(resultToPrint\* i\_prn, int countRowOfTable)

{

int widthOfTableColumns[numberOfTableColumns]={firstColumnWidth, secondColumnWidth, thirdColumnWidth, fourthColumnWidth};

char\* title[numberOfTableColumns];

title[0] = new char [std::strlen(" Function ")+1];

std::strcpy(title[0], " Function ");

title[1] = new char [std::strlen(" Integral ")+1];

std::strcpy(title[1], " Integral ");

title[2] = new char [std::strlen(" IntSum ")+1];

std::strcpy(title[2], " IntSum ");

title[3] = new char [std::strlen(" N ")+1];

std::strcpy(title[3], " N ");

int size[numberOfTableColumns];

for(int i = 0; i < numberOfTableColumns; ++i){

size[i]=std::strlen(title[i]);

}

//шапка таблицы

std::cout << ul << std::setfill('-');

for(int j = 0; j < numberOfTableColumns - 1; ++j){

std::cout << std::setw(widthOfTableColumns[j] + 3) << Td;

}

std::cout << std::setw(widthOfTableColumns[numberOfTableColumns-1] + 3) << ur << std::endl;

std::cout << vt;

for(int j = 0; j < numberOfTableColumns; ++j){

int len = (widthOfTableColumns[j] - size[j]) / 2;

std::cout << title[j] << vt;

}

std::cout << std::endl;

//заполнение таблицы

for(int i = 0; i < countRowOfTable; ++i){

std::cout << Tr << std::fixed;

for(int j = 0; j < numberOfTableColumns - 1; ++j){

std::cout << std::setfill('-')

<< std::setw(widthOfTableColumns[j] + 3) << cr;

}

std::cout

<< std::setw(widthOfTableColumns[numberOfTableColumns - 1] + 3)

<< Tl << std::setfill(' ') << std::endl;

std::cout << vt << std::setw((widthOfTableColumns[0] - std::strlen(i\_prn[i].name))/2) << ' '

<< i\_prn[i].name << std::setw((widthOfTableColumns[0]-strlen(i\_prn[i].name))/2) << vt;

std::cout << std::setw(widthOfTableColumns[1])

<< std::setprecision(6) << i\_prn[i].i\_toch << vt

<< std::setw(widthOfTableColumns[2]) << i\_prn[i].i\_sum << std::setprecision(6) << vt

<< std::setw(widthOfTableColumns[3]) << i\_prn[i].n

<< vt << std::endl;

}

//низ таблицы

std::cout << dl << std::setfill('-');

for(int j = 0; j < numberOfTableColumns - 1; ++j){

std::cout << std::setw(widthOfTableColumns[j] + 3) << Tu;

}

std::cout << std::setw(widthOfTableColumns[numberOfTableColumns - 1] + 3)

<< dr << std::setfill(' ') << std::endl;

}