**Задание 2. Создание программной системы с элементарным интерфейсом консольного режима на С++**

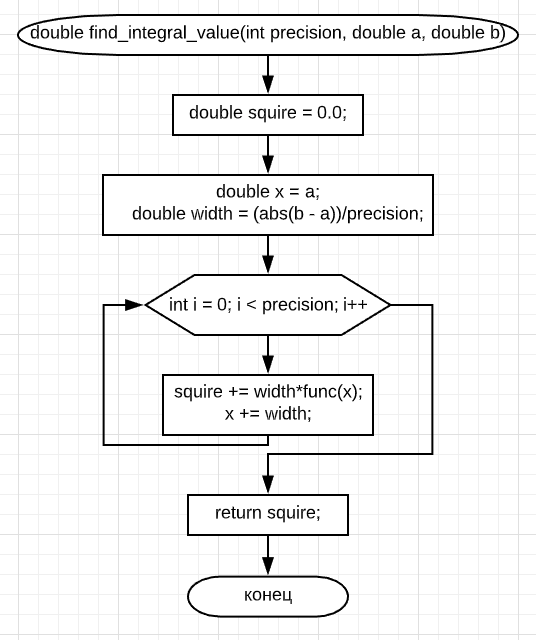
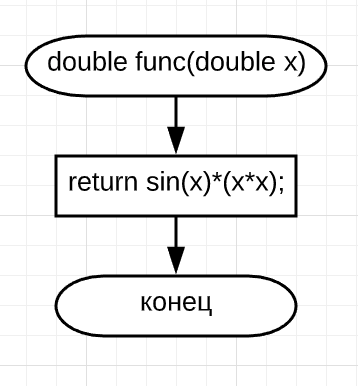
Выполнить структурную декомпозицию, разработать структурную схему, содержащую не менее 3 подпрограмм, и алгоритмы этих подпрограмм. Реализовать на С++ в консольном режиме. Предусмотреть примитивный интерфейс типа меню, позволяющий выбирать нужную подпрограмму.

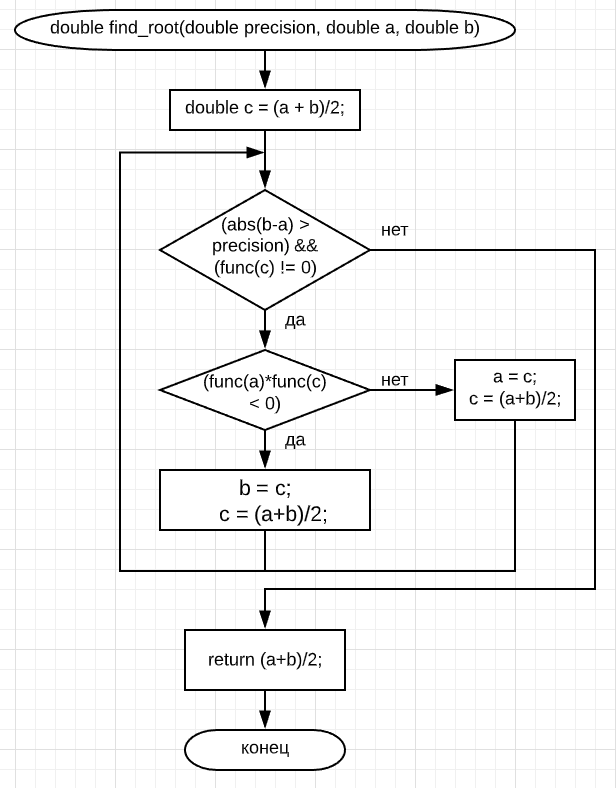
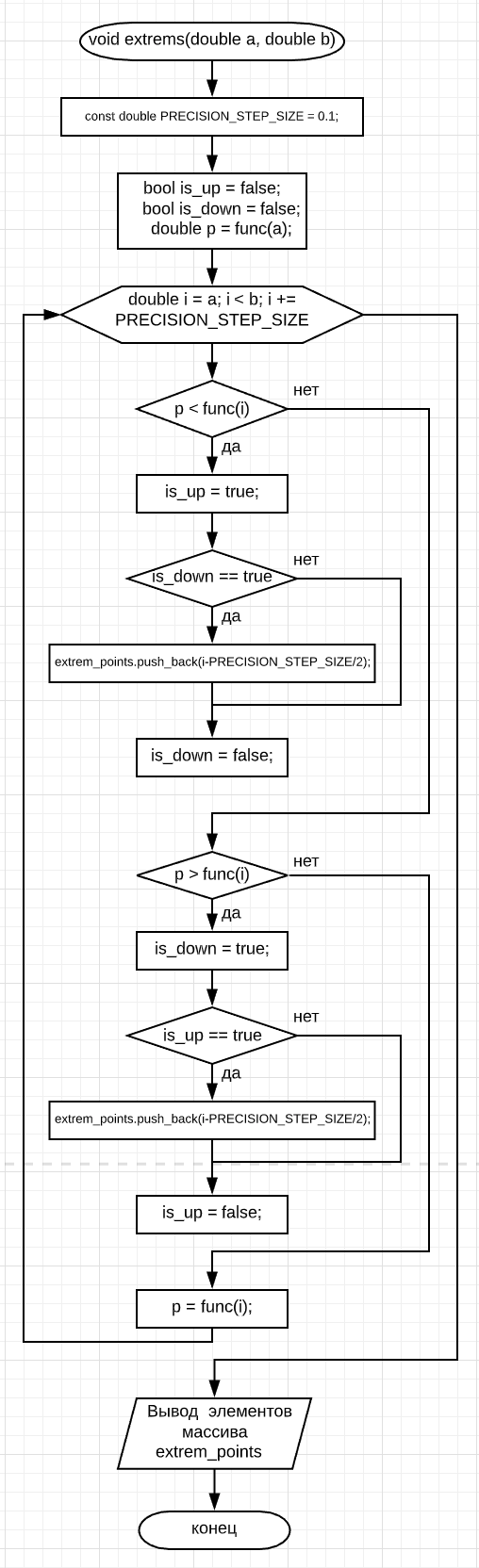
Написать программу исследования функции y=sin(x)\*x2 на заданном отрезке [a,b] . Исследование заключается в нахождении экстремумов функции, нахождении корня методом половинного деления и нахождении интеграла методом прямоугольников с заданной точностью ξ. Интервал пользователь должен иметь возможность задавать по запросу, а вид исследования выбирать с помощью меню.

**Код программы:**

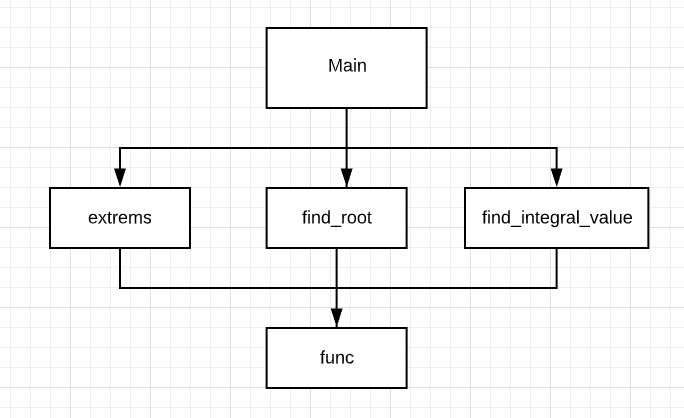
#include **<iostream>**#include **<cmath>**#include **<vector>  
  
double** func(**double** x) {  
 **return** sin(x)\*(x\*x);  
}  
  
**void** extrems(**double** a, **double** b) {  
 **const double** PRECISION\_STEP\_SIZE = 0.1;  
 *// const double INTERVAL = (b - a);* std::vector<**double**> extrem\_points;  
  
 **bool** is\_up = **false**;  
 **bool** is\_down = **false**;  
 **double** p = func(a);  
 **for** (**double** i = a; i < b; i += PRECISION\_STEP\_SIZE) {  
 **if** (p < func(i)) {  
 is\_up = **true**;  
 **if** (is\_down) extrem\_points.push\_back(i-PRECISION\_STEP\_SIZE/2);  
 is\_down = **false**;  
 }  
 **else if** (p > func(i)) {  
 is\_down = **true**;  
 **if** (is\_up) extrem\_points.push\_back(i-PRECISION\_STEP\_SIZE/2);  
 is\_up = **false**;  
 }  
 p = func(i);  
 }  
  
 *// для форматированного вывода* std::cout.setf(std::ios::fixed);  
 std::cout.precision(1);  
  
 std::cout << **"Точки экстремума функции sin(x)\*x^2 на интервале ["** << a << **", "** << b << **"]:"** << std::endl;  
 **for** (**int** i = 0; i < extrem\_points.size(); i++) std::cout << extrem\_points[i] << **" "** << std::endl;  
 std::cout << std::endl;  
  
 *// вернуть вывод как был* **return**;  
}  
  
**double** find\_integral\_value(**int** precision, **double** a, **double** b) {  
 **double** squire = 0.0;  
  
 **double** x = a;  
 **double** width = (abs(b - a))/precision;  
  
 **for** (**int** i = 0; i < precision; i++) {  
 squire += width\*func(x);  
 x += width;  
 }  
  
 **return** squire;  
}  
  
**double** find\_root(**double** precision, **double** a, **double** b) {  
 **double** c = (a + b)/2;  
 **while** ((abs(b-a) > precision) && (func(c) != 0)) {  
 **if** (func(a)\*func(c) < 0) {  
 b = c;  
 c = (a+b)/2;  
 } **else** {  
 a = c;  
 c = (a+b)/2;  
 }  
 }  
 **return** (a+b)/2;  
}  
  
  
**int** main() {  
 std::cout << find\_integral\_value(1024, 3.0, 15.0) << std::endl; *// precision - 1024+* std::cout << find\_root(0.001, 1.0, 5.0) << std::endl << std::endl; *// precision - 0.001* extrems(3.0, 15.0);  
  
 **return** 0;  
}

**Схема алгоритма:**

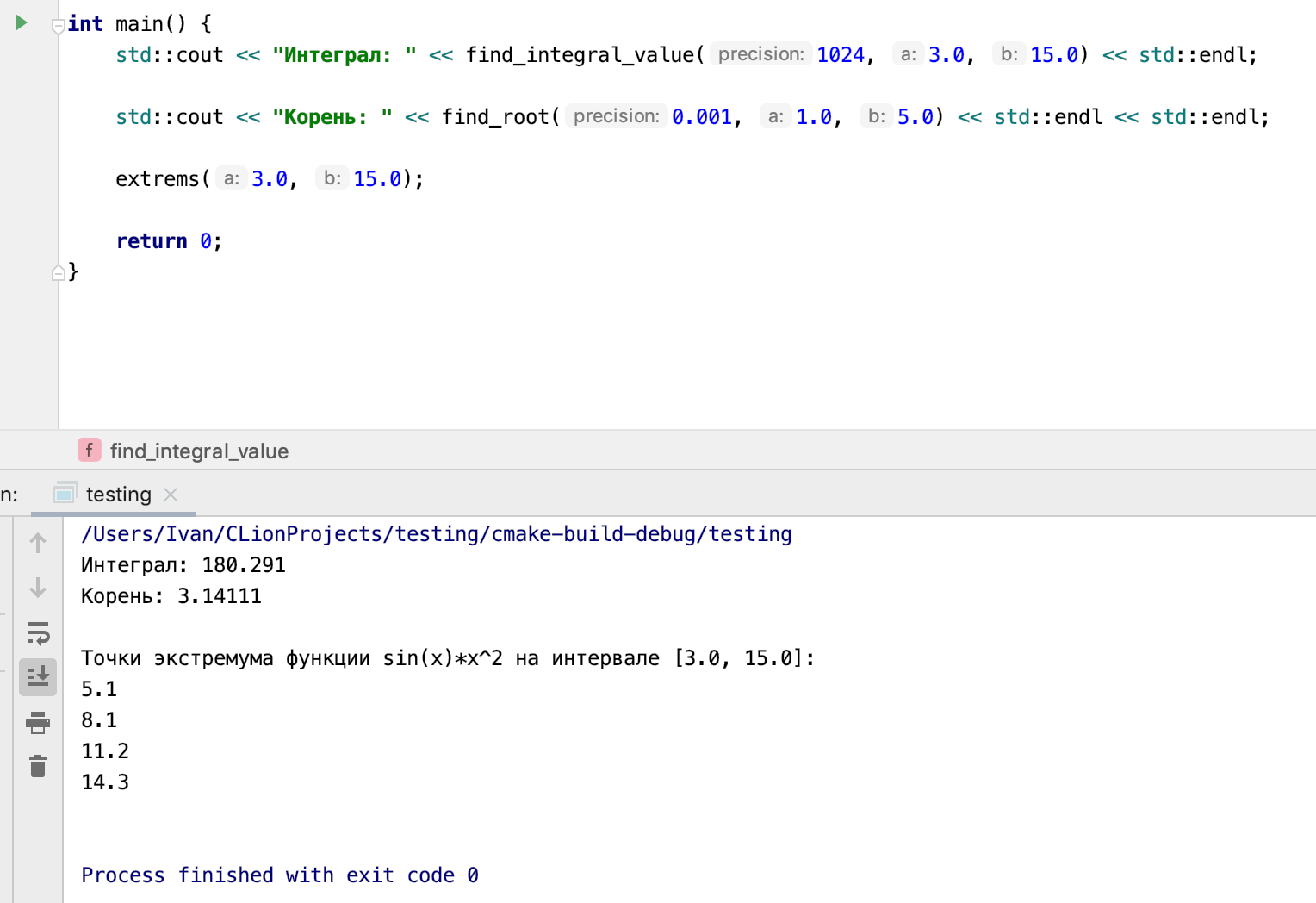
****

****

**Процедурная декомпозиция**

****

**Результат:**

****