Национальный Исследовательский Университет

«Московский Энергетический Институт»

Кафедра прикладной математики и искусственного интеллекта

Тема: Статистический анализ качества кода.

Студент: Ростовых Александра

Москва 2021

**Использование cppcheck**

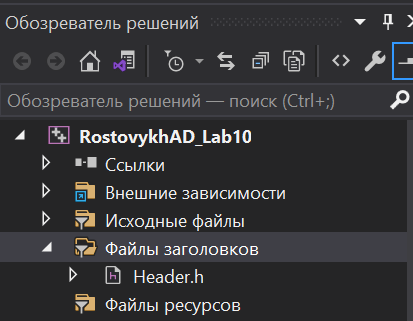
Cppcheck - это инструмент статического анализа кода C / C ++. Он обеспечивает уникальный анализ кода для обнаружения ошибок и фокусируется на обнаружении неопределенного поведения и опасных конструкций кода. Цель состоит в том, чтобы уменьшить количество ложных срабатываний. Cppcheck разработан, чтобы иметь возможность анализировать ваш код C / C ++, даже если он имеет нестандартный синтаксис (распространенный во встроенных проектах).

**Цель работы**

Научиться проверять качество кода С++ с помощью статического анализатора cppcheck.

**1. Создать тестовый проект на C++**

Создаем проект на С++. Добавим в проект заголовочный файл Header.h (чтобы понимать, что cppcheck будет просматривать полностью весь проект).



Добавим тестовые функции и тестируемый код в проект:

Файл Header.h:

#pragma once

int t() {

char a[10];

a[11] = 0;

return 0;

}

void f() {

int ct;

while (ct < 100)

{

std::cout << ct;

}

}

void g() {

for (int i; i < 1; i++) {

i++;

}

}

double Division(int n, int k) {

double res = 0;

return n/k;

}

Файл RostovykhAD\_Lab10.cpp:

#include <iostream>

#include "Header.h"

#include <math.h>

void f(int\* ptr) {

\*ptr = 0;

}

void func(int\* Arr) {

Arr = new int[10];

for (int i = 0; i < 10; i++) Arr[i] = i;

}

bool isNegative(int x)

{

if (x < 0)

return true;

}

int main()

{

struct type1 { int t1; };

int\* a = new int; // переменная

int\* b = new int[2]; // массив

delete b;

delete [] a;

double d = Division(5, 3);

double dd = Division(5, 0);

f();

g();

double d1 = 1.5;

int n = d1;

int\* Ptr = 0;

func(Ptr);

for (int i = 0; i < 10; i++) std::cout << Ptr[i] << " ";

type1 t;

std::cout << t.t1;

bool neg = isNegative(1);

std::cout << neg;

int\* Ptr1 = 0;

f(Ptr1);

double k = sqrt(-3);

return 0;

}

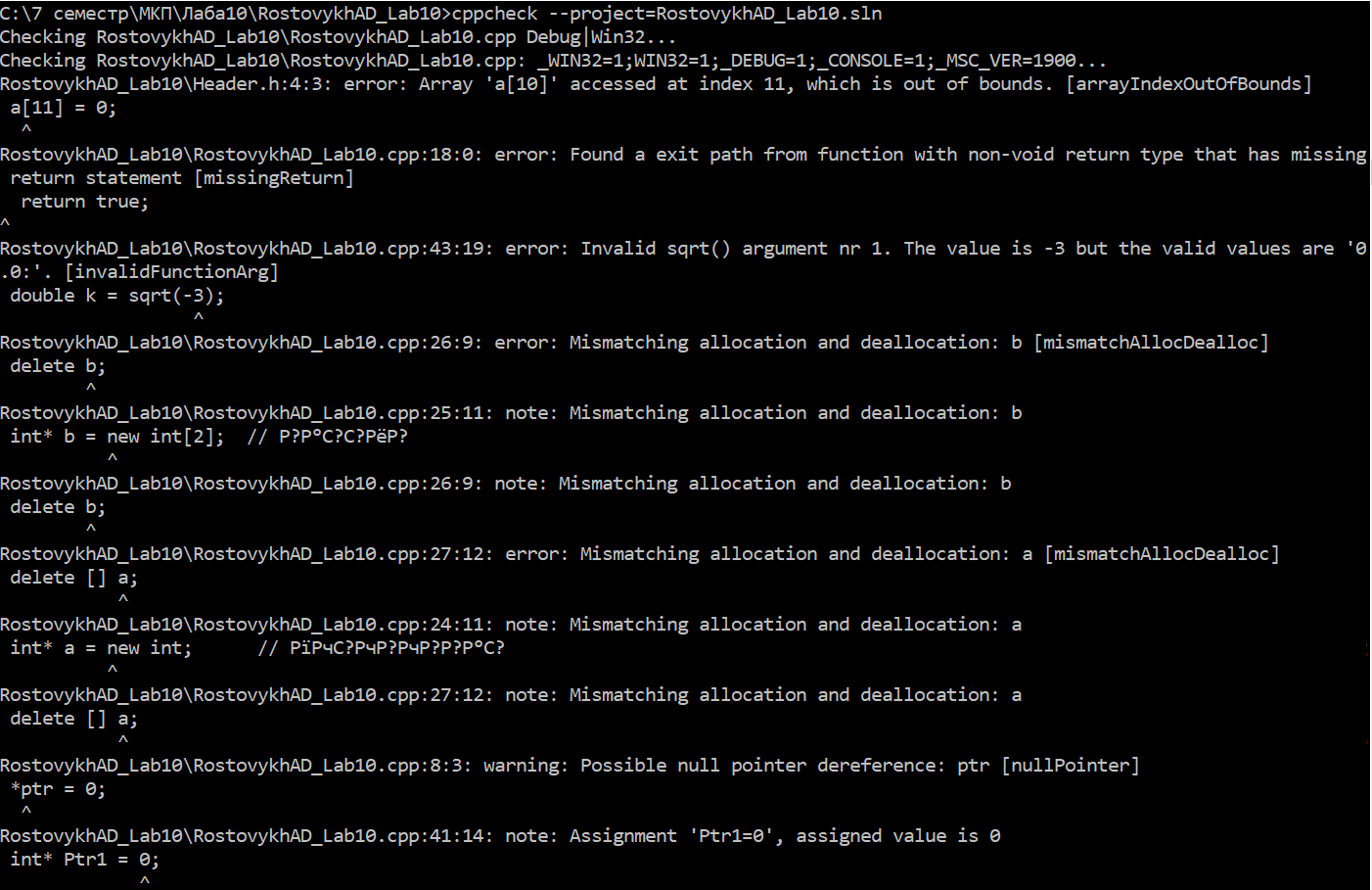
Из приведенного выше кода видно, что он имеет ошибки, такие как, например, выход за границы массива, взятие корня из отрицательного числа, разыменование пустого указателя и так далее.

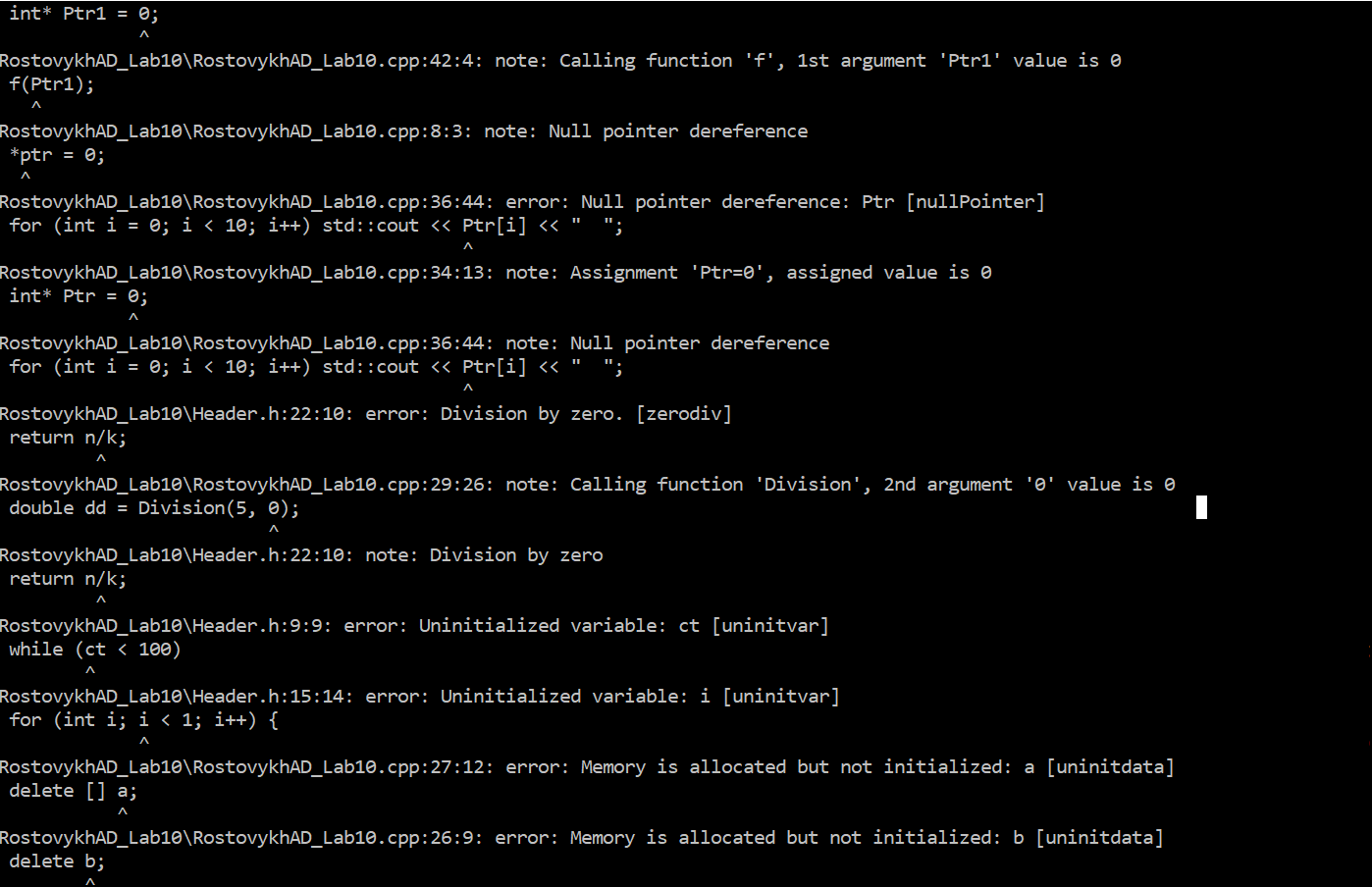
**2. Настроить проверку качества кода с помощью cppcheck после компиляции (через интерфейс командной строки)**

Наберем в командной строке следующую команду (предварительно добавив cppcheck после установки в переменную PATH):

C:\7 семестр\МКП\Лаба10\RostovykhAD\_Lab10>cppcheck --project=RostovykhAD\_Lab10.sln

Данной командой мы проведем анализ всех файлов проекта и увидим следующий результат:







Ошибка: Выход за границы массива:

*RostovykhAD\_Lab10\Header.h:4:3: error: Array 'a[10]' accessed at index 11, which is out of bounds. [arrayIndexOutOfBounds]*

*a[11] = 0;*

*^*

Ошибка: Функция, определенная как void возвращает значение:

*RostovykhAD\_Lab10\RostovykhAD\_Lab10.cpp:18:0: error: Found a exit path from function with non-void return type that has missing return statement [missingReturn]*

*return true;*

*^*

Ошибка: Взятие корня из отрицательного числа:

*RostovykhAD\_Lab10\RostovykhAD\_Lab10.cpp:43:19: error: Invalid sqrt() argument nr 1. The value is -3 but the valid values are '0.0:'. [invalidFunctionArg]*

*double k = sqrt(-3);*

*^*

Ошибка: Несоответствие выделения и удаления памяти:

*RostovykhAD\_Lab10\RostovykhAD\_Lab10.cpp:26:9: error: Mismatching allocation and deallocation: b [mismatchAllocDealloc]*

*delete b;*

*^*

Предупреждение: Возможное разыменование пустого указателя:

*RostovykhAD\_Lab10\RostovykhAD\_Lab10.cpp:8:3: warning: Possible null pointer dereference: ptr [nullPointer]*

*\*ptr = 0;*

*^*

Ошибка: Разыменование пустого указателя:

*RostovykhAD\_Lab10\RostovykhAD\_Lab10.cpp:36:44: error: Null pointer dereference: Ptr [nullPointer]*

*for (int i = 0; i < 10; i++) std::cout << Ptr[i] << " ";*

*^*

Ошибка: Деление на ноль:

*RostovykhAD\_Lab10\Header.h:22:10: error: Division by zero. [zerodiv]*

*return n/k;*

*^*

*RostovykhAD\_Lab10\RostovykhAD\_Lab10.cpp:29:26: note: Calling function 'Division', 2nd argument '0' value is 0*

*double dd = Division(5, 0);*

*^*

Ошибка: Неинициализированная переменная:

*RostovykhAD\_Lab10\Header.h:9:9: error: Uninitialized variable: ct [uninitvar]*

*while (ct < 100)*

*^*

Большинство проверок cppcheck по умолчанию не включает. Среди них следующие категории проверок, каждая из которых может включаться/выключаться независимо:

error — явные ошибки, которые анализатор считает критическими и обычно они приводят к багам (включено по умолчанию);

warning — предупреждения, здесь даются сообщения о небезопасном коде;

style — стилистические ошибки, сообщения появляются в случае неаккуратного кодирования (больше похоже на рекомендации);

performance — проблемы производительности, здесь cppcheck предлагает варианты, как сделать код быстрее (но это не всегда даёт прирост производительности);

portability — ошибки совместимости, обычно связано с различным поведением компиляторов или систем разной разрядности;

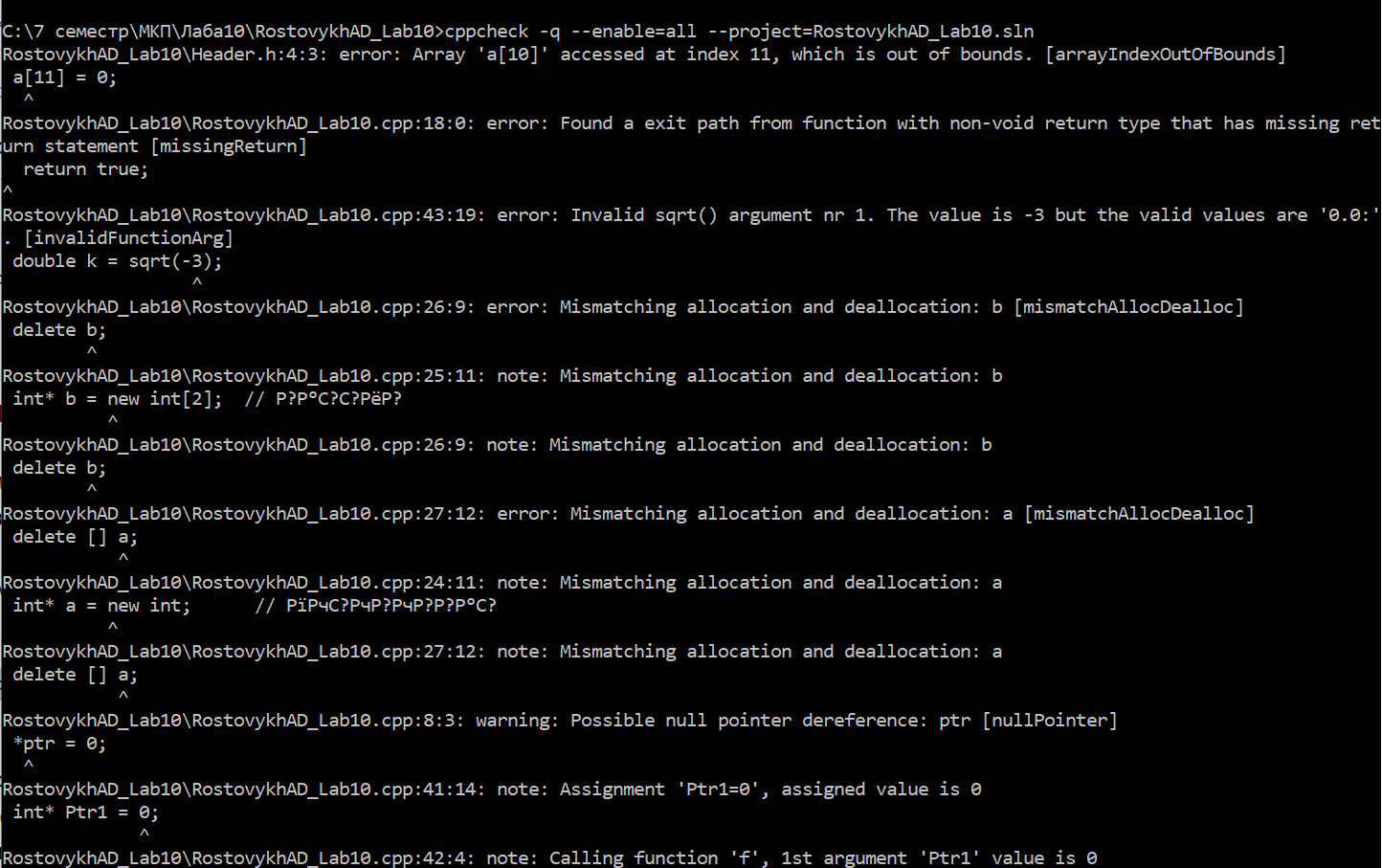
information — информационные сообщения, возникающие в ходе проверки (не связаны с ошибками в коде);

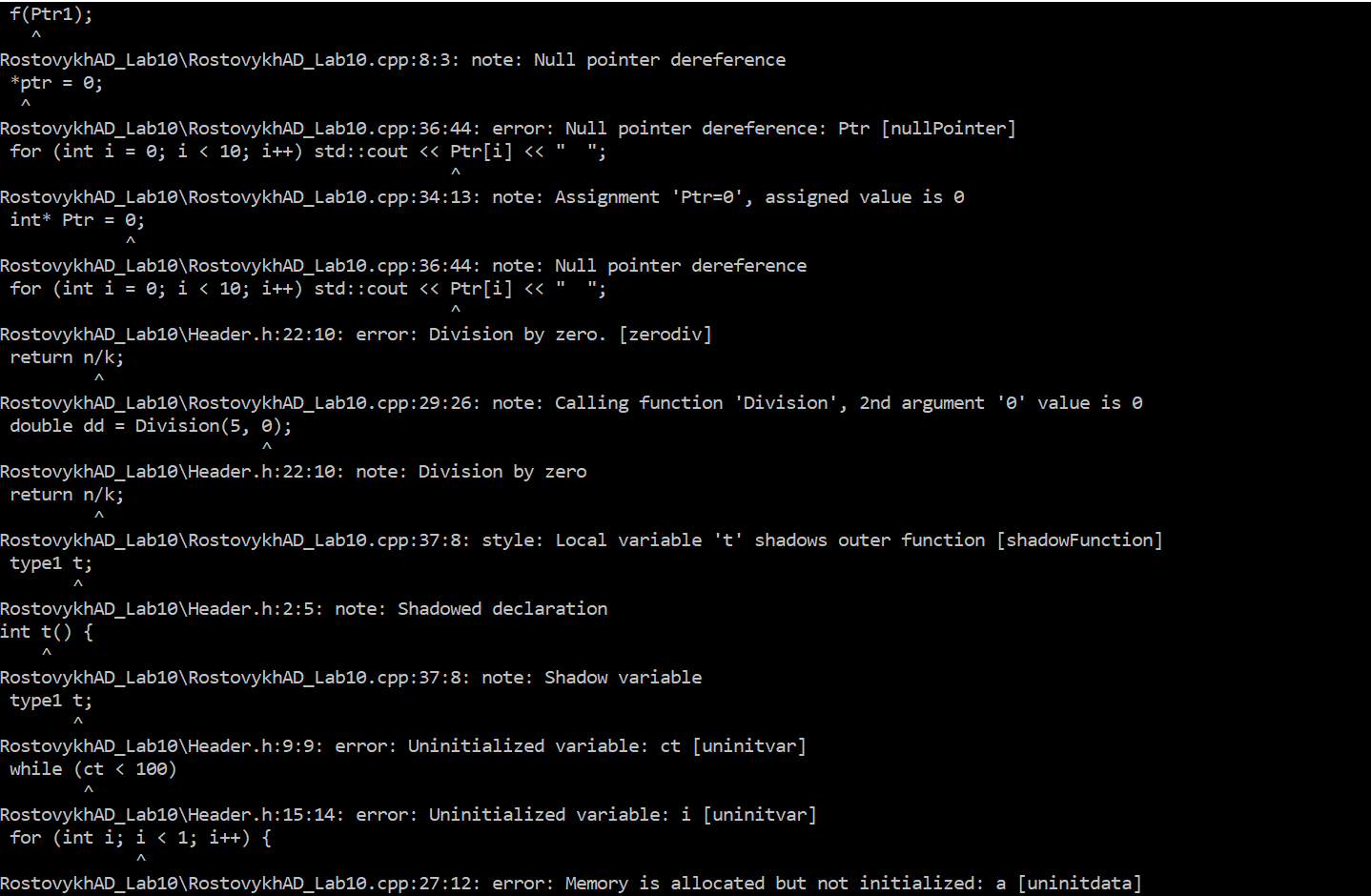
unusedFunction — попытка вычислить неиспользуемые функции (мёртвый код), не умеет работать в многопоточном режиме;

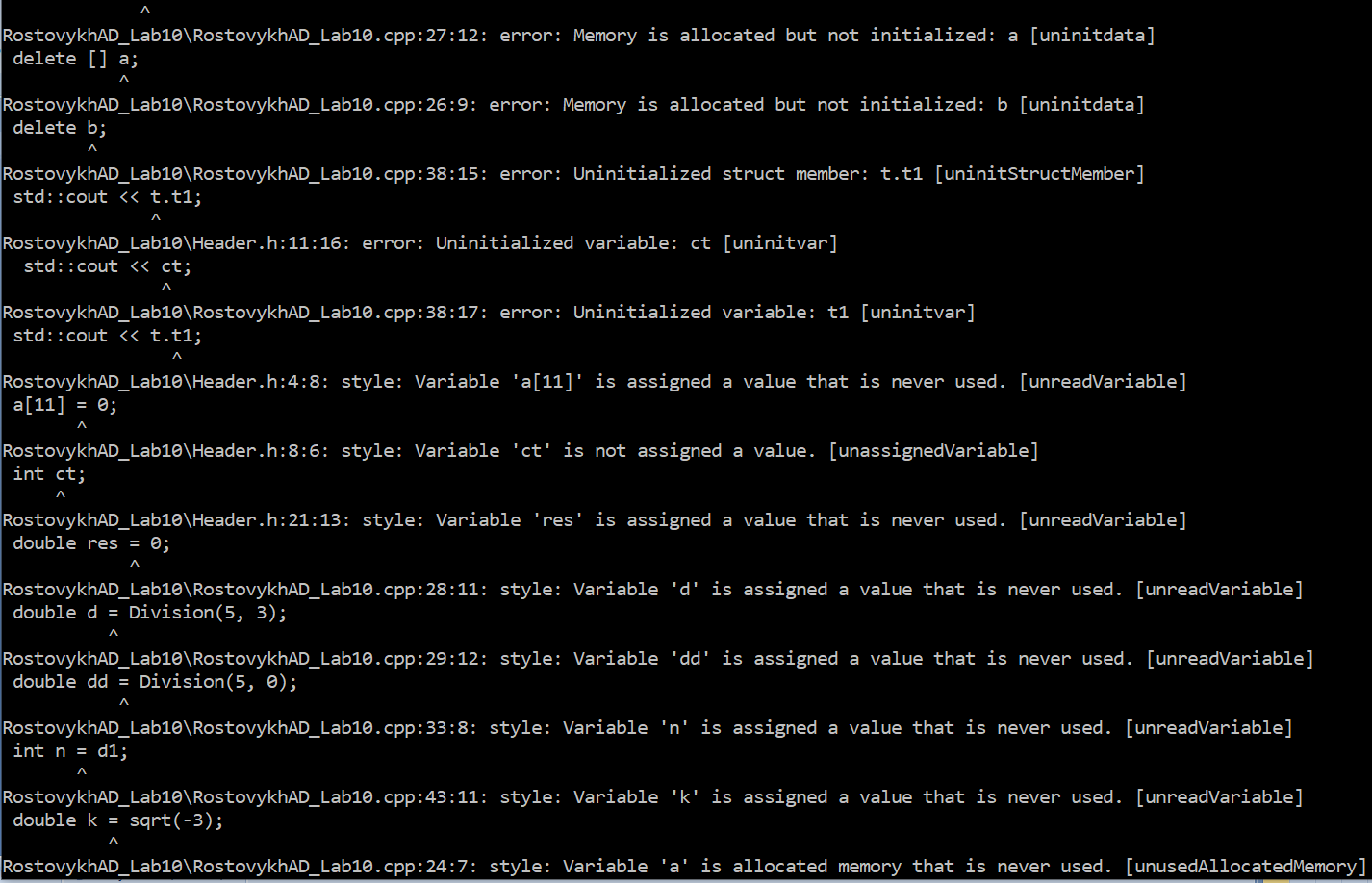
missingInclude — проверка на недостающий #include (например, используем random, а подключить stdlib.h забыли).

Включаются проверки параметром --enable, список категорий проверок перечисляется через запятую.

Добавим проверки:

**

**

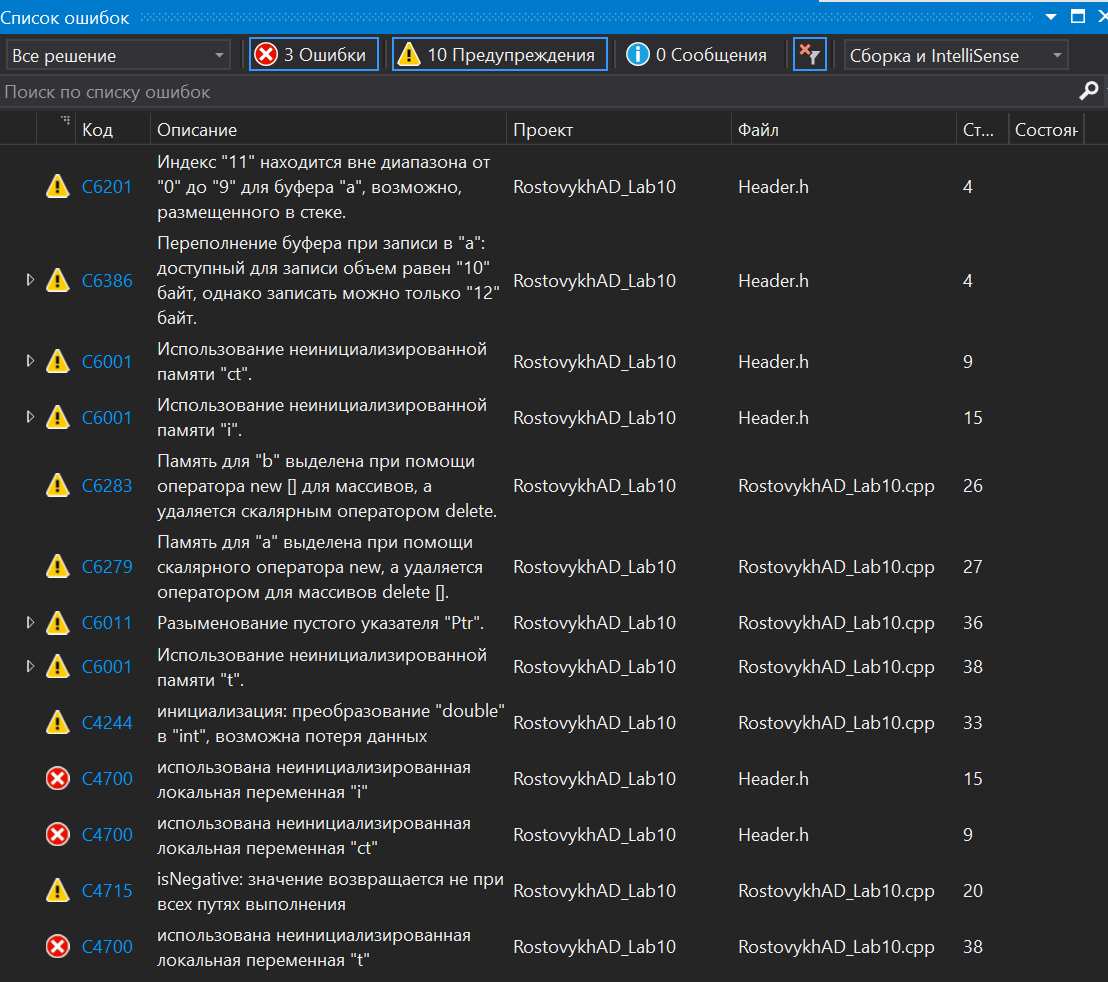
**

Появились проверки типа style, которые сообщают нам, например, о введенных, но неиспользованных переменных, а так же о неинициализированных.

**3. Собрать проект, проанализировать полученные предупреждения и ошибки**

Соберем проект и посмотрим, что нам выдаст окно ошибок и предупреждений:

Видим некоторые ошибки и предупреждения, но их намного меньше, чем выдал нам cppcheck, здесь, например, нет такой ошибки, как деление на ноль, выявленной cppcheck в ходе анализа.

****