**Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования**

**"Уфимский государственный авиационный технический университет"**

**Кафедра** Высокопроизводительных вычислительных технологий и систем

**Дисциплина:** Программирование

**Отчет по лабораторной работе № 2**

**Тема: «Функции»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Группа ПМИ-148 | Фамилия И.О. | Подпись | Дата | Оценка |
| Студент | Фаизова А.А. |  |  |  |
| Принял | Гайнетдинова А.А. |  |  |  |

**Уфа 2017**

**Требования к составу отчета:**

Отчет к лабораторной работе оформляется в текстовом процессоре Microsoft Word или OpenOffice (LibreOffice) Write в соответствии с требованиями стандарта СТО УГАТУ 016-2007 и содержать

• титульный лист,

• описание цели работы и краткую теоретическую справку по использованным операторам и функциям языка программирования,

• описание выполнения задания:

1. формулировка задания, как в методичке,
2. блок-схема каждой разработанной функции,
3. исходный текст разработанного приложения,
4. скриншоты примеров выполнения программы.

• выводы

• список использованной литературы,

**Требования к оформлению отчета**

* Текст отчета набирается шрифтом Times New Roman, размер шрифта 14pt или 12pt единый во всем документе. Исходный код должен быть набран любым моноширинным шрифтом (например, Courier New), размер символов можно уменьшить до 10 pt. Поля страницы: верхнее и нижнее – 2 см., левое – 2 см, правое – 1.5 см.
* Абзац должен начинаться с красной строки, за исключением тех случаев, когда абзац разорван каким-либо математическим выражением. Выравнивание внутри текстовых абзацев «по ширине», выравнивание в блоке с исходным кодом – «по левому краю».
* Все блок-схемы оформляются в соответствие с ГОСТ 19.701. Описание ГОСТ и пример блок-схемы приведены в разделе Справочник, методических указаний по выполнению лабораторных работ на сайте. Если блок-схемы составляются в сторонних приложениях, следует предусмотреть возможность их правки в учебном классе. В противном случае отчет с некорректно составленной блок-схемой и исправлениями вручную приниматься не будут.
* Описание выполнения каждого задания начинается с новой страницы. Следует использовать заголовок с текстом «**Индивидуальное задание №\_\_**», набранный полужирным шрифтом с выравниванием по левому краю, без красной строки. Текст задания, текст в блок-схеме и описание программы следует набирать шрифтом Times New Roman с прямым начертанием (не курсив!!!).
* Текст на скриншотах должен быть читаемым. Скриншоты должны содержать только окно с результатами выполнения программы, а не весь рабочий стол. Размер шрифта на скриншотах должен соответствовать по размеру окружающему его тексту.

Ниже и выше красным выделены места, которые необходимо изменить.

**Цель:** ознакомиться с принципами разбиения программ на блоки и правилами оформления функций на языке С

**Теоретический материал**

*Описание функции*

Подпрограмма это самостоятельный программный модуль (выделенная часть программы), принимающий данные, обрабатывающий их и возвращающий результат обработки. Подпрограммы делятся на два типа: функции и процедуры.

Функции обязательно принимают одно и более значение (входные параметры) и возвращают одно значение (возвращаемое значение). То есть результатом работы функции всегда является некое значение любого типа.

Процедуры могут иметь или не иметь входные параметры и ничего не возвращают явным образом. То есть результатом работы процедуры является некое действие.

В Си верно также и такое определение:

Функция - это группа операторов, у которой есть имя.

Обязательными для функции являются два компонента: определение и вызовы. То есть функцию обязательно необходимо определить – описать все ее операторы. Вызов – это использование функции, то есть передача ей данных и получение результата.

В Си нет процедур, есть только функции. Процедуры реализуются посредством функций, возвращающих пустое значение.

*Объявление функции*

До того, как функция будет вызвана, она должна быть объявлена.

Объявление функции, аналогично объявлению переменной, указываются имя функции, тип значения, которое она может возвращать, набор её параметров (для каждого параметра задаётся тип и, при желании, имя).

Объявление функции называют также её прототипом.

Схема:

Тип\_результата Имя\_функции (Тип\_пар1, Тип\_пар2, ...);

Тип\_результата — некоторый существующий (например, встроенный) тип данных или ключевое слово void, указывающее на то что функция никакого значения возвращать не будет.

Имя\_функции — уникальный для данного пространства имён идентификатор.

Тип\_парN — некоторый существующий (например, встроенный) тип данных для N-oro аргумента.

Примеры:

int max (int, int);

double cube (double);

float massa();

void printarr(\*int, int);

После объявления к функции можно обращаться в программе по имени, хотя пока и не понятно, какие действия она будет производить.

Если функция не возвращает никакого результата, т. е. объявлена как void, ее вызов не может быть использован как операнд более сложного выражения (например, значение такой функции нельзя чему-то присвоить).

*Определение (описание) функции*

Определение или описание функции содержит перечень тех операций, которые будут производится внутри функции.

Схема:

Тип\_результата Имя\_функции (Тип\_пар1 Имя\_пар1, Тип\_пар2 Имя\_пар2, ...) {

Оператор1;

Оператор2;

...

ОператорN;

return n;

};

Имя\_парN — уникальное внутри функции имя N-ro параметра. Имена параметров можно задавать и в прототипе функции, тогда в определении надо использовать те же имена.

ОператорN — некоторые операторы и выражения, содержащиеся внутри функции и выполняющиеся каждый раз при вызове функции. Внутри операторов мы можем обращаться к глобальным объектам программы; к локальным объектам, объявленным внутри функции; а также к аргументам функции.

return n — оператор, останавливающий работу функции и возвращающий n в качестве её значения (при этом тип n должен соответствовать типу результата в объявлении функции). Наличие этого оператора обязательно для функции возвращающей значение. Для функции объявленной как void можно вызывать оператор return без аргументов, это досрочно завершит функцию, иначе — будут выполнены все действия до конца блока описания функции.

Первая строка - это, по существу, заголовок функции, который отличается от ее прототипа только отсутствием точки с запятой в конце и обязательным присутствием имен формальных параметров.

Здесь тип либо void (для функций, не возвращающих значения), либо обозначение типа возвращаемого функцией значения.

Список параметров функции может заканчиваться запятой с последующим многоточием "...". Многоточие обозначает возможность обращаться к функции с большим количеством фактических параметров, чем явно указано в спецификации параметров. Такая возможность должна быть обеспечивается специальными средствами в теле функции.

Тело функции - это часть определения функции, ограниченная фигурными скобками и непосредственно размещенная вслед за заголовком функции. Тело функции может быть либо составным оператором, либо блоком. Особенность языка Си состоит в невозможности внутри тела функции определить другую функцию. Другими словами, определения функций не могут быть вложенными, однако объявлять функцию можно и в теле другой функции.

Вывод: в языке Си допустимы функции с параметрами и без параметров, функции, возвращающие значения указанного типа и ничего не возвращающие.

Параметры функции часто называют формальными параметрами (аргументами), т. к. они определяют тип данных, порядок и количество передаваемой в функцию информации, но не саму информацию. При вызове функции ей указываются фактические параметры (аргументы). Если параметры не используются, вместо них рекомендуют ставить ключевое слово void.

Блок определения функции называется также её телом.

! Одна функция не может объявляться или определяться внутри другой (т.е. нельзя объявлять и определять функции внутри main).

Пример объявления и описания функции:

int max (int, int);

int max (int n1,int n2) {

if(nl > n2) {

return n1;

} else {

return n2;

}

}

int main(void) {

int а = 100 - max(10,20);

printf("a = %d",a);

return 0;

}

Видно, что вся информация, имевшаяся в прототипе функции, повторяется в её определении, поэтому если функция определена до её первого вызова, то отдельно прототип указывать не обязательно.

Пример:

double cube (double a) {

return a\*a\*a;

}

int main(void) {

double pi = 3.1415;

printf("Volume is %lf", cube(pi));

return 0;

}

Но отдельно прототип указывают в тех случаях, когда функция будет описываться позже своего использования. Например, можно было объявить функцию до main, вызвать её из main, но описать только после main.

Пример:

double cube (double);

int main(void) {

double pi = 3.1415;

printf("Volume is %lf", cube(pi));

return 0;

}

double cube (double a) {

return a\*a\*a;

}

Такая возможность позволяет создавать модульные программы, исходный код которых может храниться даже в разных файлах.

*Библиотеки функций*

В Cи используются различные виды функций. Прежде всего это стандартные и библиотечные функции. Стандартные функции входят в стандартную библиотеку, поставляемую вместе с компилятором. К библиотечным относятся любые функции, описание и реализация которых находятся в библиотеках не входящих в стандартный набор.

Библиотеки подключается при помощи директивы препроцессора #include. После директивы препроцессора указывается имя библиотеки или имя заголовочного файла библиотеки. Имя заключается в угловые скобки (для стандартных библиотек) или в двойные кавычки (для остальных библиотек).

В угловых скобках или в двойных кавычках указывается имя специального заголовочного файла (расширение .h как раз и образовано от английского слова header – заголовок). В заголовочных файлах приводятся прототипы функций. А реализация обычно размещается в файле с таким же именем, но расширение меняется на .c

*Макросы препроцессора*

Существуют два типа макросов: макрос-объект(object-like macro) и макрос-функция(function-like macro), оба типа объявляются с помощью директивы #define. Рассмотрим сначала макросы-объекты. Объявляются они как:

#define ИМЯ\_МАКРОСА [замещающий текст]

Всё, что идёт после имя макроса до конца строки является замещающим текстом. При помощи макросов-объектов часто определяются константы или часто встречающиеся последовательности переменных и аврифметических операторов.

Следует помнить, что макрос это просто подстановка а не переменная.

Второй вид макросов — это макро-функции (function-like macros). Определяются они с помощью той-же директивы #define, после которой (сразу без пробелов) в круглых скобках идёт список разделённых запятыми аргументов.

*Рекурсивные функции*

Рекурсивной называют функцию, которая прямо или косвенно сама вызывает себя. Именно возможность прямого или косвенного вызова позволяет различать прямую или косвенную рекурсию.

При каждом обращении к рекурсивной функции создается новый набор объектов автоматической памяти, локализованных в теле функции.

Рекурсивные алгоритмы эффективны, например, в тех задачах, где рекурсия использована в определении обрабатываемых данных. Поэтому серьезное изучение рекурсивных методов нужно проводить, вводя динамические структуры данных с рекурсивной структурой.

Функция называется *косвенно рекурсивной* в том случае, если она содержит обращение к другой функции, содержащей прямой или косвенный вызов определяемой (первой) функции. В этом случае по тексту определения функции её рекурсивность может быть и не видна. Если в функции используется вызов этой же функции, то имеет место *прямая рекурсия*, т.е. функция по определению рекурсивная.

*Итерационные функции*

Итерация в программировании — организация обработки данных, при которой действия повторяются многократно, не приводя при этом к вызовам самих себя.

**Ответы на контрольные вопросы**

1. Приведите структуру (формат) определения функции.

Определение или описание функции содержит перечень тех операций, которые будут производится внутри функции.

Схема:

Тип\_результата Имя\_функции (Тип\_пар1 Имя\_пар1, Тип\_пар2 Имя\_пар2, ...) {

Оператор1;

Оператор2;

...

ОператорN;

return n;

};

Имя\_парN — уникальное внутри функции имя N-ro параметра. Имена параметров можно задавать и в прототипе функции, тогда в определении надо использовать те же имена.

ОператорN — некоторые операторы и выражения, содержащиеся внутри функции и выполняющиеся каждый раз при вызове функции. Внутри операторов мы можем обращаться к глобальным объектам программы; к локальным объектам, объявленным внутри функции; а также к аргументам функции.

return n — оператор, останавливающий работу функции и возвращающий n в качестве её значения (при этом тип n должен соответствовать типу результата в объявлении функции). Наличие этого оператора обязательно для функции возвращающей значение. Для функции объявленной как void можно вызывать оператор return без аргументов, это досрочно завершит функцию, иначе — будут выполнены все действия до конца блока описания функции.

1. Приведите структуру (формат) вызова функции.

Для обращения к функции используется выражение с операцией «круглые скобки»:

обозначение\_функции (список\_фактических\_параметров)

Операндами операции ‘()’ служат обозначение\_функции и список\_фактических\_параметров. Наиболее естественное и понятное обозначение\_функции – это её имя. Кроме того, функцию можно обозначить, разыменовав указатель на нее.

Список фактических параметров, называемых аргументами, - это список выражений, количество которых равно числу формальных параметров функции (исключение составляют функции с переменных кол-вом параметров). Соответствие между формальными и фактическими параметрами устанавливается по их взаимному расположению в списках. Порядок вычисления значений фактических параметров стандарт языка Си не определяет.

Между формальными и фактическими параметрами должно быть соответствие по типам. Лучше всего, когда тип фактического параметра совпадает с типом формального параметра. В противном случае компилятор автоматически добавляет команду преобразования типов, что возможно только в том случае, когда такое приведение типов допустимо.

Особое внимание нужно уделить правилам передачи параметров при обращении к функциям. Синтаксис языка Си предусматривает только один способ передачи параметров – передачу по значениям. Это означает, что формальные параметры функции локализованы в ней, т.е. недоступны вне определения функции и никакие операции над формальными параметрами в теле функции не изменяют значений фактических параметров.

Вызов функции всегда является выражением, однако размещение такого выражения в тексте программы зависит от типа возвращаемого функцией значений.

1. С помощью какого оператора производится возврат из функции в точку ее вызова?

Обязательным, но не всегда явно используемым оператором функции является оператор возврата из функции в точку вызова, имеющий две формы:

return;

return выражение;

Первая форма соответствует завершению функции, не возвращающей никакого значения – для функции, перед именем которой в ее определении указан тип void. Выражение во второй форме оператора return должно иметь тип, указанный перед именем функции в ее определении, либо иметь тип, допускающий автоматическое преобразование к типу возвращаемого функцией значения.

1. Что такое прототип (объявление) функции?

Для вызова функции, компилятор организует передачу в нее исходной информации, которая указывается в операторе вызова. Этой информацией заполняются параметры функции. Если тип передаваемой информации не соответствует типу параметра функции, компилятор будет вынужден осуществить преобразование передаваемой информации к типу параметра. Преобразование информации может происходить и при приеме результата.

Если компилятору функция не известна, то передача информации осуществляется согласно типу этой информации, а в качестве типа результата принимается тип int. Такое поведение компилятора может быть неприемлемым при работе со многими функциями, а такая ситуация может наблюдаться, когда вызов функции будет расположен выше ее определения, или определение будет сделано в другом файле. Поэтому, в таких случаях,

компилятор должен быть информирован о типе результата и о типах параметров. Делается это с помощью прототипа – заголовка функции, скопированного в нужное место и закрытого точкой с запятой. Иногда прототип называют объявлением или описанием функции.

Пример прототипа функции, показанной выше:

int summa(int a, int b) ;

Названия параметров в прототипе можно пропустить, в этом случае прототип будет выглядеть следующим образом:

int summa(int, int) ;

1. Где размещают прототипы функций?

Прототипы функций можно располагать только в местах декларации переменных. Типичное расположение прототипов – в начале файла программы или в собственном заголовочном файле.

1. Чем прототип функции отличается от ее заголовка?

В отличие от заголовка функции в её прототипе могут не указываться имена формальных параметров , и прототип завершается точкой с запятой.

1. Каким образом включают в программу прототипы библиотечных функций?

Библиотеки подключается при помощи директивы препроцессора #include. После директивы препроцессора указывается имя библиотеки или имя заголовочного файла библиотеки. Имя заключается в угловые скобки (для стандартных библиотек) или в двойные кавычки (для остальных библиотек).

В угловых скобках или в двойных кавычках указывается имя специального заголовочного файла (расширение .h как раз и образовано от английского слова header – заголовок). В заголовочных файлах приводятся прототипы функций. А реализация обычно размещается в файле с таким же именем, но расширение меняется на .c

1. Опишите передачу параметров в функцию по значению.

Передача параметров по значению предусматривает следующие шаги:

1. При компиляции функции (точнее, при подготовке к ее выполнению) выделяются участки памяти для формальных параметров, т.е. формальные параметры оказываются внутренними объектами функции. При этом для параметров типа float формируются объекты типа double, а для параметров типов char и short int создаются объекты типа int. Если параметром является массив, то формируется указатель на начало этого массива и он служит представлением массива параметра в теле функции.
2. Вычисляются значения выражений, использованных в качестве фактических параметров при вызове функции.
3. Значения выражений - фактических параметров заносятся в участки памяти, выделенные для формальных параметров функции. При этом float преобразуется в double, a char и short int в тип int.
4. В теле функции выполняется обработка с использованием значений внутренних объектов параметров, и результат передается в точку вызова функции как возвращаемое ею значение.
5. Никакого влияния на фактические параметры (на их значения) функция не оказывает.
6. После выхода из функции освобождается память, выделенная для ее формальных параметров.

Вызов функции всегда является выражением, однако размещение такого выражения в тексте программы зависит от типа возвращаемого функцией значения. Если в качестве типа возвращаемого значения указан тип void, то функция является функцией без возвращаемого результата. Такая функция не может входить ни в какие выражения, требующие значения, а должна вызываться в виде отдельного выражения-оператора

1. Приведите определение встраиваемой функции. Какие требования предъявляются к такой функции?

Функцию можно определить как встроенную с помощью модификатора inline, который рекомендует компилятору вместо обращения к функции помещать ее код непосредственно в каждую точку вызова. Модификатор inline ставится перед типом функции. Он применяется для коротких функций, чтобы снизить накладные расходы на вызов (сохранение и восстановление регистров, передача управления). Директива inline носит рекомендательный характер и выполняется компилятором по мере возможности. Использование inline-функций может увеличить объем исполняемой программы. Определение функции должно предшествовать ее вызовам, иначе вместо inline-расширения компилятор сгенерирует обычный вызов.

1. Что обрабатывает модификатор inline?

Ключевое слово inline перед функцией позволяет вставить тело функции прямо в тот код, из которой она вызывается (т.е. как макроопределении, но более безопасное). Модификатор inline анализирует код соответствующей функции, и если этот код простой (без циклов, рекурсий), то будет произведена простая замена.

1. Приведите определение рекурсивной функции.

Рекурсия – это вызов функции самой себя. Ниже показан пример прямой рекурсии:

int func()

{

…

func() ; // Вызов самой себя

…

}

Рекурсия образует цикл, который начнет завершаться, когда очередной вызов функции пропустит вызов самой себя. Рекурсивный цикл бесконечно выполняться не может, поскольку на каждый вызов функции тратиться память в специальной структуре программы, называемой программным стеком.

Поскольку рекурсия образует цикл, то ее можно применять для всех задач, связанных с повторениями.

1. Какие элементы обязательно должен содержать рекурсивный алгоритм (рекурсивная функция)?

Рекурсивный алгоритм должен включать рекурсивную формулу, условие прекращения выполнения алгоритма, чтобы рекурсия не продолжалась до бесконечности.

1. Дайте определение шаблона функции.

Шаблоны (англ. template) — средство языка C++, предназначенное для кодирования обобщённых алгоритмов, без привязки к некоторым параметрам (например, типам данных, размерам буферов, значениям по умолчанию).

Итак, описание шаблона начинается с ключевого слова template за которым в угловых скобках («<» и «>») следует список параметров шаблона. Далее, собственно идет объявление шаблонной сущности (например функция или класс), т. е. имеет вид: template < template-parameter-list > declaration.

Шаблон — это лишь макет, по которому компилятор самостоятельно будет генерировать код.

**Индивидуальное задание №1**

Задание: Написать программу, содержащую две функции. Первая функция, вычисляющая значение ay, располагается после функции main(). Вторая, функция печати результата, располагается до функции main().

Блок-схема

Начало

Ввод

a

Ввод

y

x:=1

m:=|y|

**WL**

Пока m>0

x:=x\*a

**A**

Начало

алгоритма

Ввод основания степени

Ввод степени

Переменной, отвечающей за значение получаемого числа, присваиваем начальное значение

Вспомогательную переменную m приравниваем к модулю значения степени

Начало цикла, включающего в себя функцию по вычислению значения

ay

**A**

m:=m-1

**WL**

y < 0

x:=

Да

Нет

Вывод

x

Конец

Соединитель

Конец цикла

Проверка условия «Отрицательна ли степень?»

Присвоение x обратного значения

при отриц.степени

Ввод значения ay

Конец алгоритма

Описание программы**:**

a – значение основания степени;

y – значение степени;

x – значение ay;

m – вспомогательная переменная, с использование которой происходит выполнение цикла в функции по возведению числа а в степень у.

Исходный код программы

Пример программы, в которой функция располагается после функции main().

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

double Pow(double, int);

int main()

{

double a=0, x=0;

int y=0;

printf ("Enter the base of power:\n");

scanf ("%lf", &a);

printf ("Enter the exponent:\n");

scanf ("%d", &y);

x=Pow(a,y);

printf ("Value: %lf\n",x);

return 0;

}

double Pow(double a, int y)

{

double x = 1;

int m = abs(y);

while (m > 0)

{

x \*= a;

m--;

}

if (y < 0)

{x = 1/x;}

return (x);

}

Пример программы, в которой функция располагается до функции main().

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

double Pow(double a, int y)

{

double x = 1;

int m = abs(y);

while (m > 0)

{

x \*= a;

m--;

}

if (y < 0)

{x = 1/x;}

return (x);

}

int main()

{

double a=0, x=0;

int y=0;

printf ("Enter the base of power:\n");

scanf ("%lf", &a);

printf ("Enter the exponent:\n");

scanf ("%d", &y);

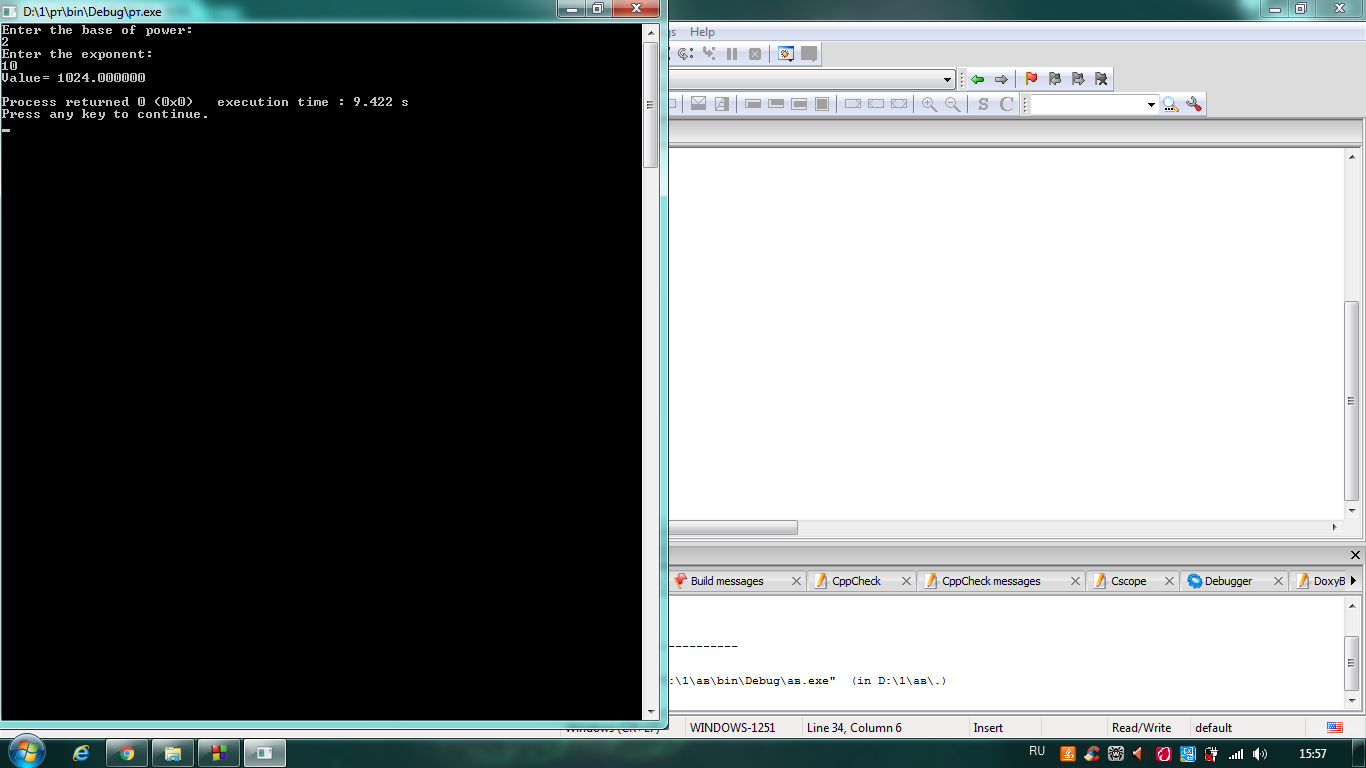
x=Pow(a,y);

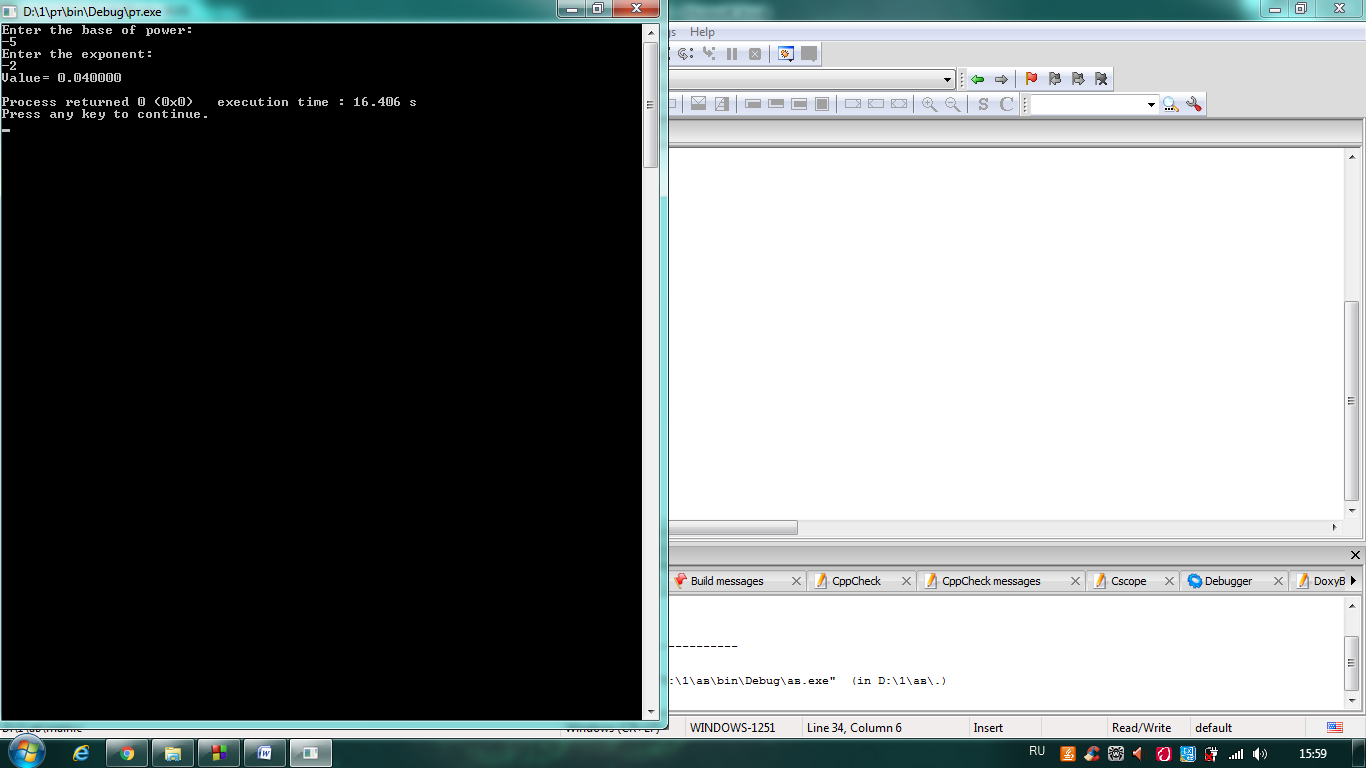
printf ("Value: %lf\n",x);

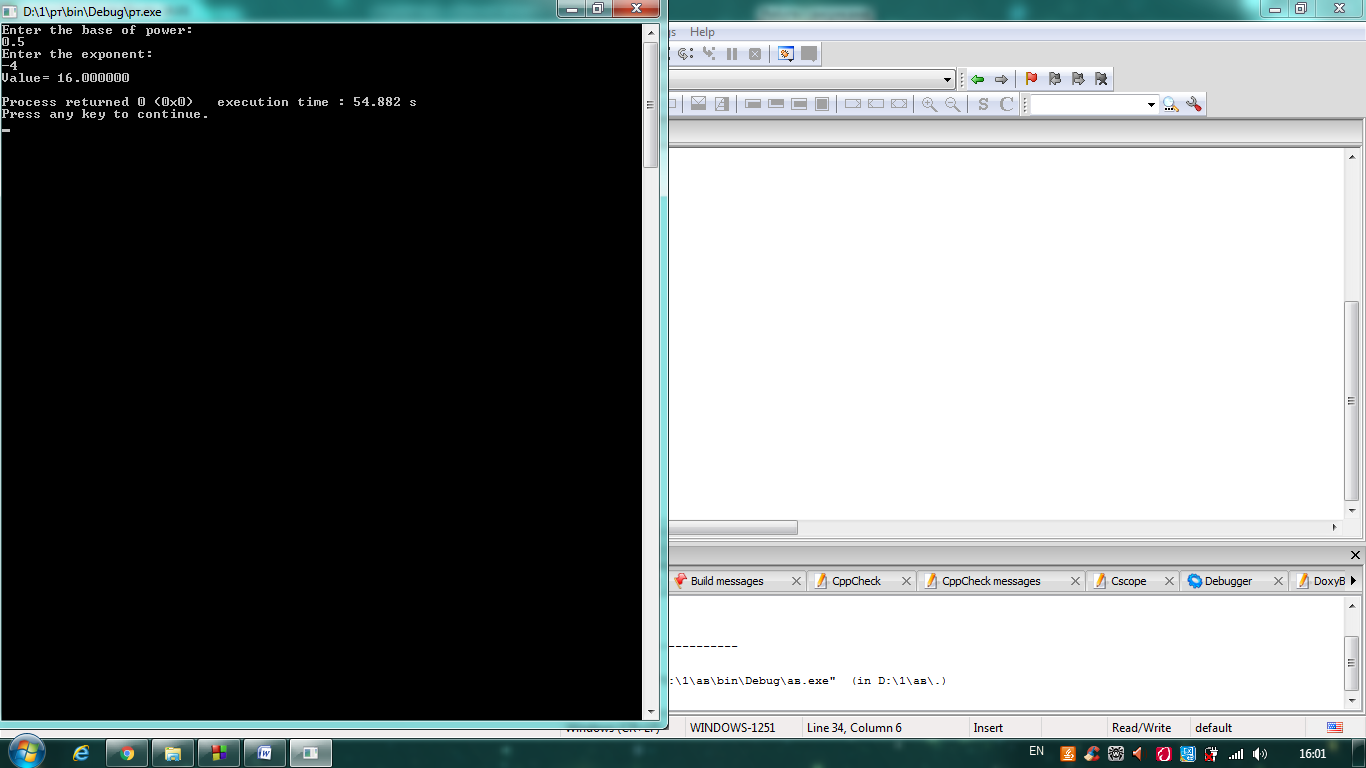
return 0;

}

Пример выполнения программы

**

**

**

**Индивидуальное задание №2**

Задание: Написать программу для вычисления объема цилиндра с кубической полостью. Расчет объема организовать с помощью макроса препроцессора.

Блок-схема

Начало

Ввод

R, H

Ввод

S

R>0 И H>0 И S>0 И S<=R И S<=H

Вывод

сообщения об ошибке

Вывод

V

Конец

V1:=πR2H

V2:=S3

V:=V1-V2

Нет

Да

Начало цикла

Конец цикла

Ввод значений радиуса основания и высоты цилиндра

Ввод значения стороны кубической полости

**Проверка** корректности исходных данных и существования заданной полости в цилиндре

Расчёт значения объёма цилиндра

Расчёт значения объёма кубической полости

Вычисление объёма цилиндра с кубической полостью

Вывод значения объёма заданной фигуры

Описание программы**:**

R –радиус основания цилиндра;

H –высота цилиндра;

S –сторона кубической полости;

V1 –объём цилиндра без полости;

V2 –объём кубической полости;

V –объём цилиндра с кубической полостью

Исходный код программы

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#define PI 3.14159265

#define v1 (R\*R\*H\*PI)

#define v2 (S\*S\*S)

#define v (v1 - v2)

int main()

{ float R=0, H=0;

float S=0;

float V=0, V1=0, V2=0;

printf ("Cylinder's radius = ");

scanf ("%f",&R);

printf ("Cylinder's height = ");

scanf ("%f", &H);

printf ("Cube's side = ");

scanf ("%f", &S);

if ((R > 0) && (H > 0) && (S > 0) && (S <= (sqrt(2)\*R) && (S <= H))) {

V1 = v1;

V2 = v2;

V = v;

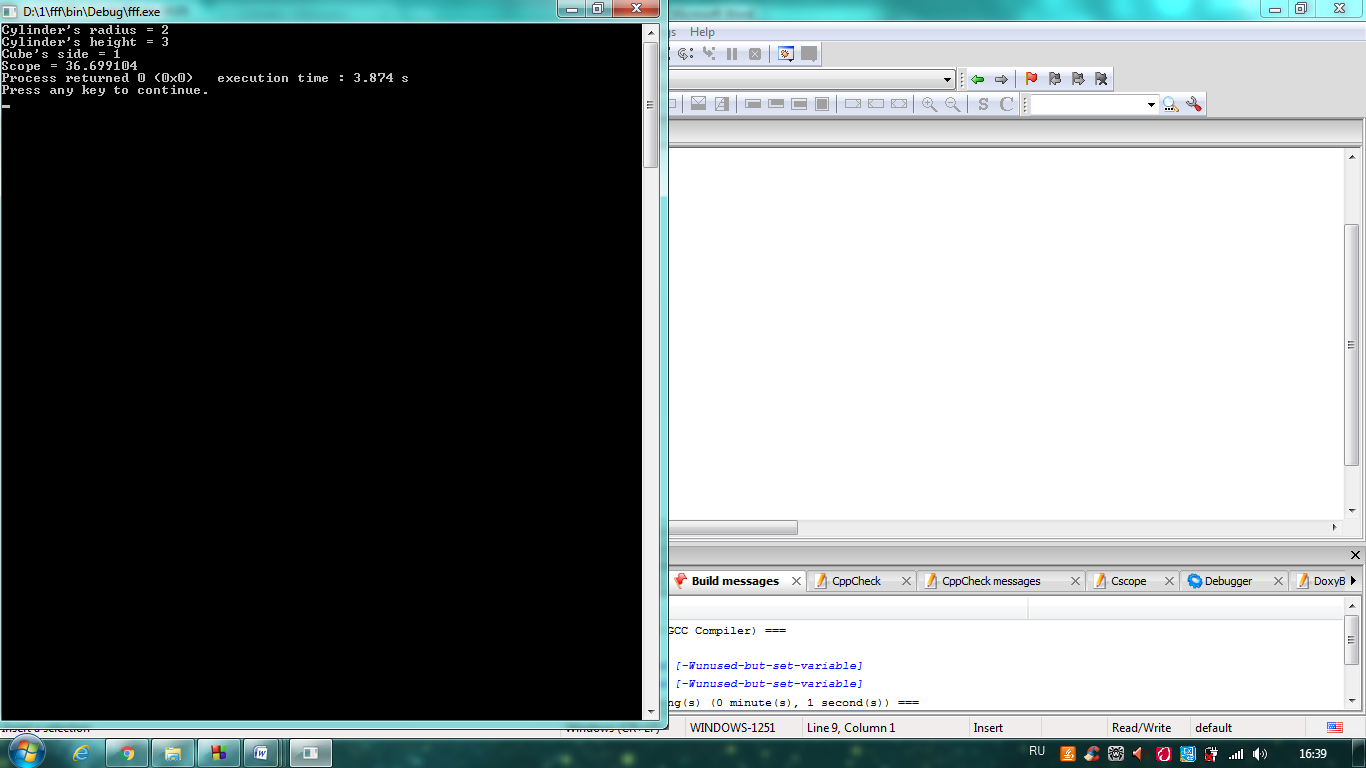
printf ("Scope = %f",V);}

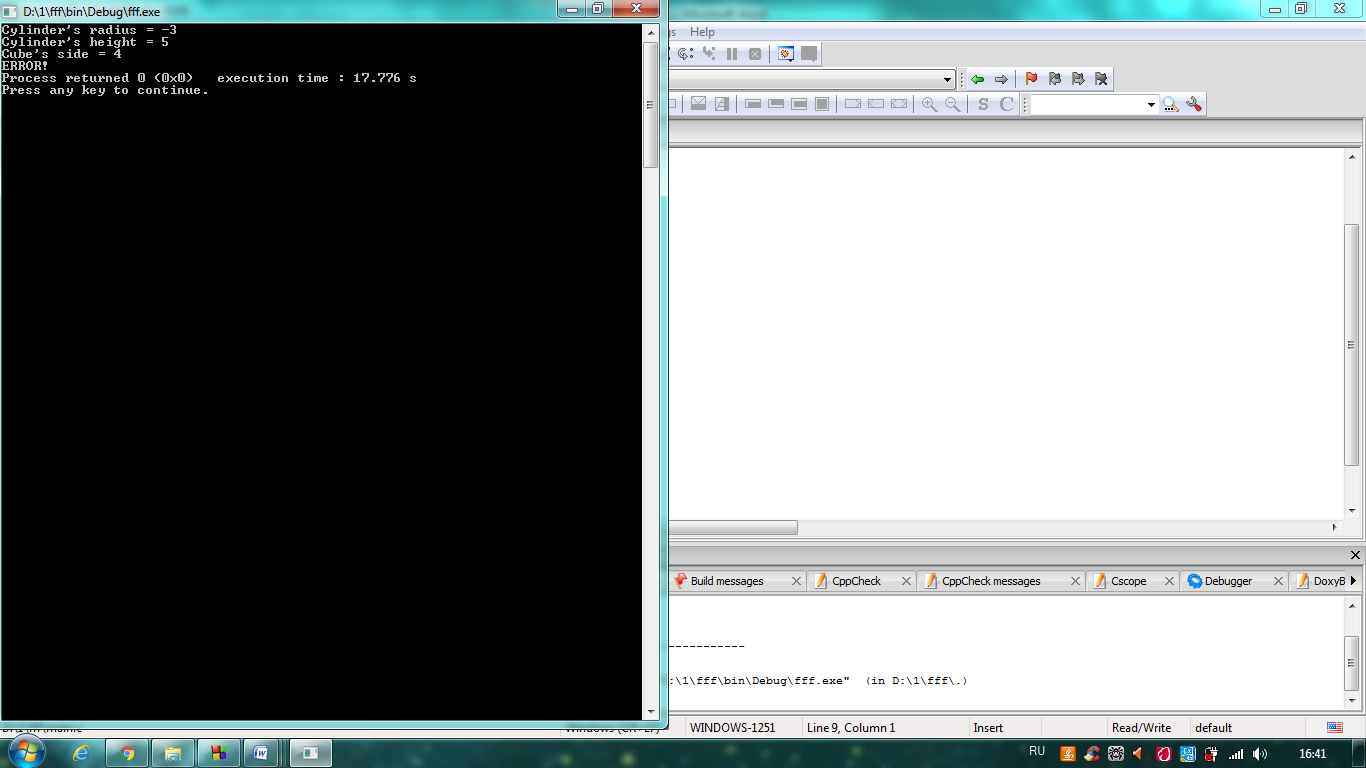
else {printf ("ERROR!");}

return 0;

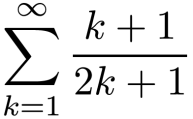
}

Пример выполнения программы





**Индивидуальное задание №3**

Задание: Написать программу, которая вычисляет частичную сумму ряда 

Расчет должен производиться двумя способами:

* при помощи итерационного цикла,
* при помощи рекуррентной формулы.

\*В третьем задании на каждый вариант создается отдельная функция. Функции помещаются в отдельный файл-библиотеку, для которой создается свой заголовочный файл. В основной программе этот заголовочный файл подключается. Результаты расчета по обоим методам сравниваются.

Блок-схема

Расчёт суммы при помощи итерационного цикла:

Начало

Конец

Ввод

N

sum:=0

k:=1,N;1

sum:=sum+

k

Вывод

sum

Ввод кол-ва слагаемых ряда

Переменной, отвечающей за сумму ряда, присваиваем начальное (нулевое) значение

Итерационный цикл

Вычисление частичной суммы ряда; переменная цикла i принимает начальное значение 1 до заданного числа N с шагом 1

Вывод значения частичной суммы ряда

Расчёт суммы при помощи рекурсивной формулы

Начало

Конец

Ввод

N

sum:=0

sum:= +

+sum recur(N-1)

Вывод

sum

N=1?

sum:=

Да

Нет

Ввод кол-ва слагаемых ряда

Переменной, отвечающей за сумму ряда, присваиваем начальное (нулевое) значение

Вывод значения частичной суммы ряда

Цикл-рекурсия

Описание программы**:**

N –количество слагаемых ряда;

sum –N-ая частичная сумма ряда;

k –переменная цикла for в итерационном цикле

Исходный код программы

#include <stdio.h>

#include <stdlib.h>

double sum\_iter(int N)

{

double k = 1 ;

double sum = 0;

for (k = 1; k <= N; k++)

{

sum = sum + (k + 1)/(2\*k + 1);

}

return (sum);

}

double sum\_rec(int N)

{

if (N == 1) return ((1.0+1.0)/(2.0+1.0));

return ((1 + (double)N)/(2\*(double)N + 1) + sum\_rec(N + -1));

}

int main()

{

int N = 0;

double sum=0;

printf (" Please enter boundary for sum: ");

scanf ("%d",&N);

sum = sum\_iter(N);

printf("\n Sum by iteration is %lf\n",sum);

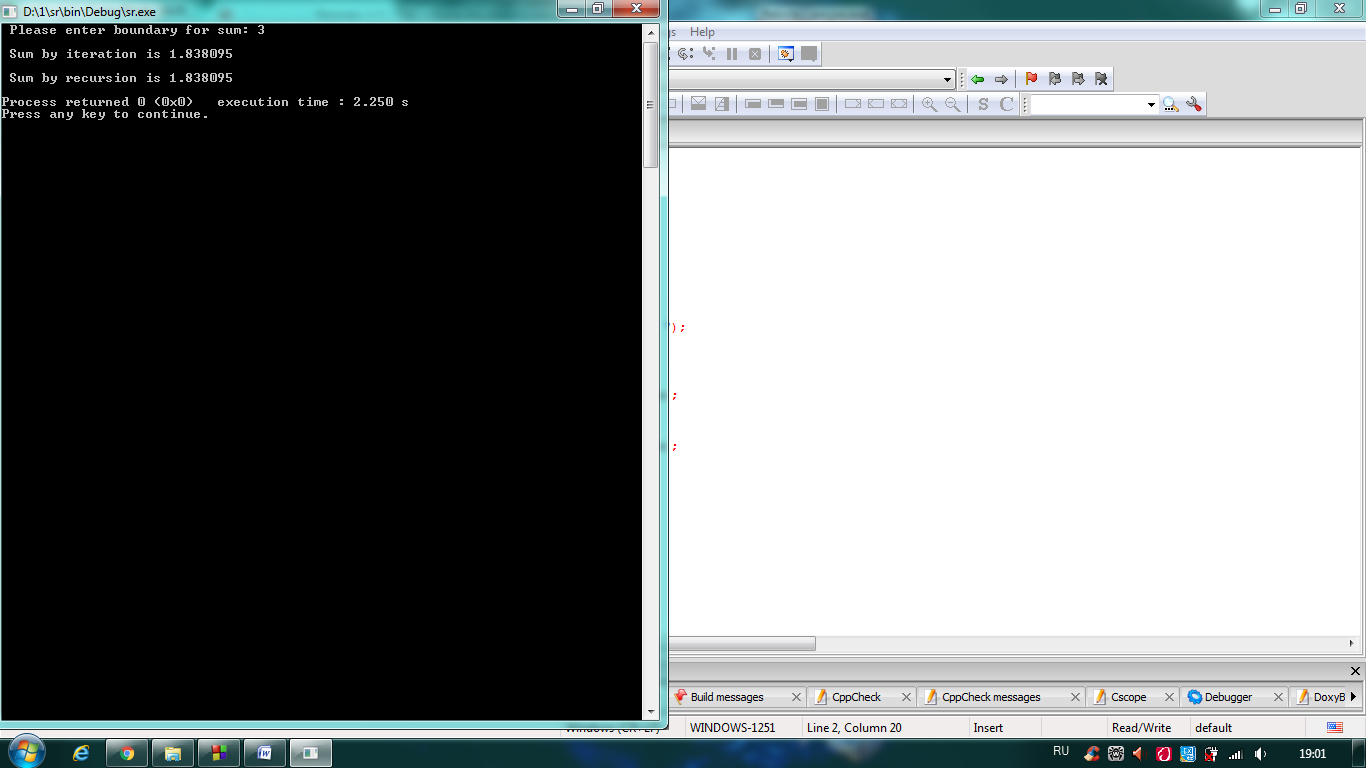
sum = sum\_rec(N);

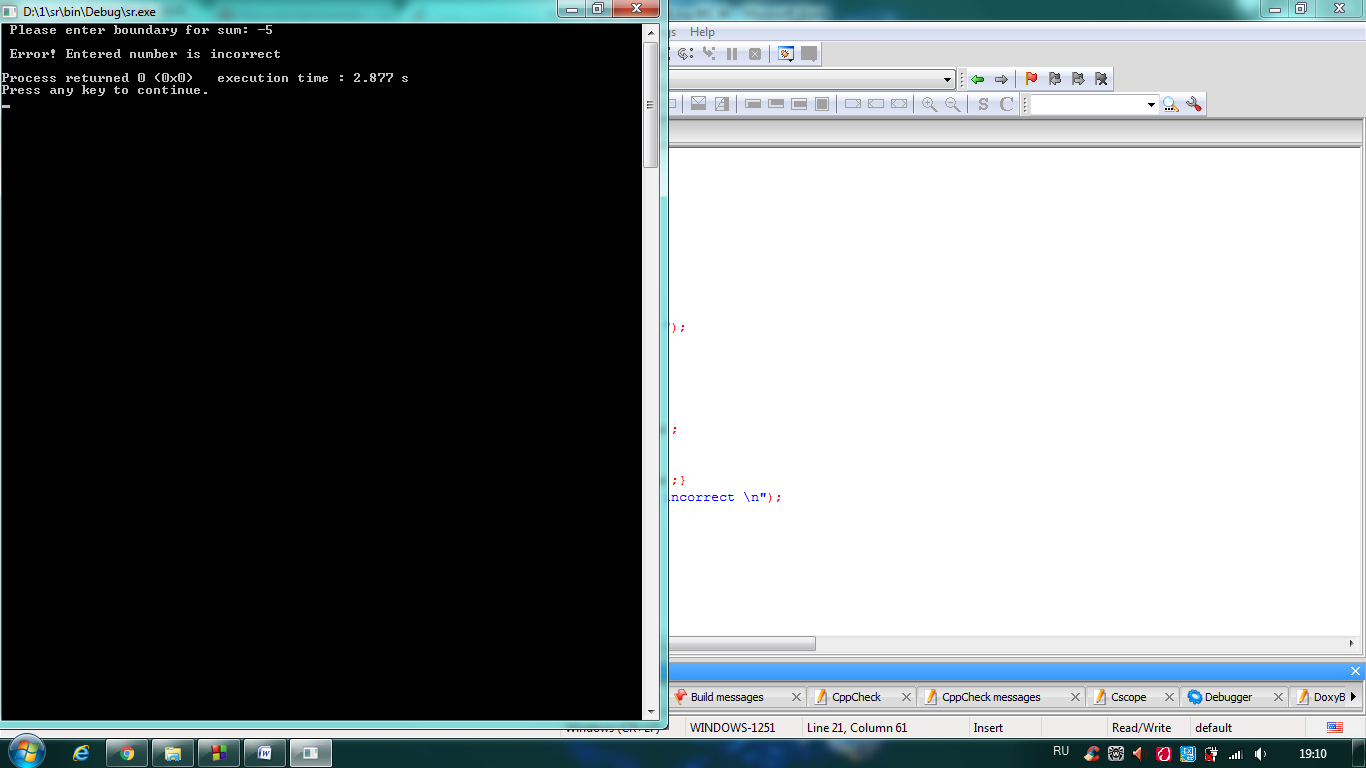
printf("\n Sum by recursion is %lf\n",sum);

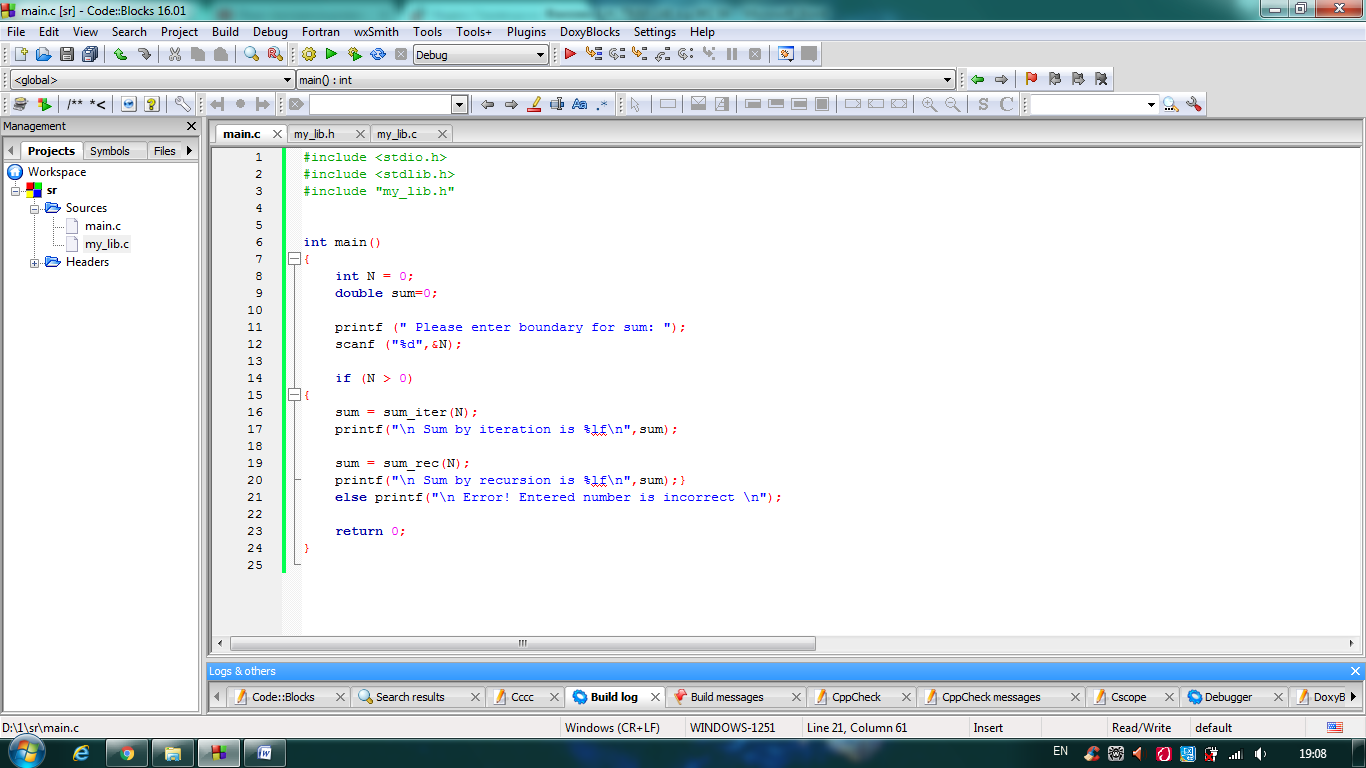
return 0;

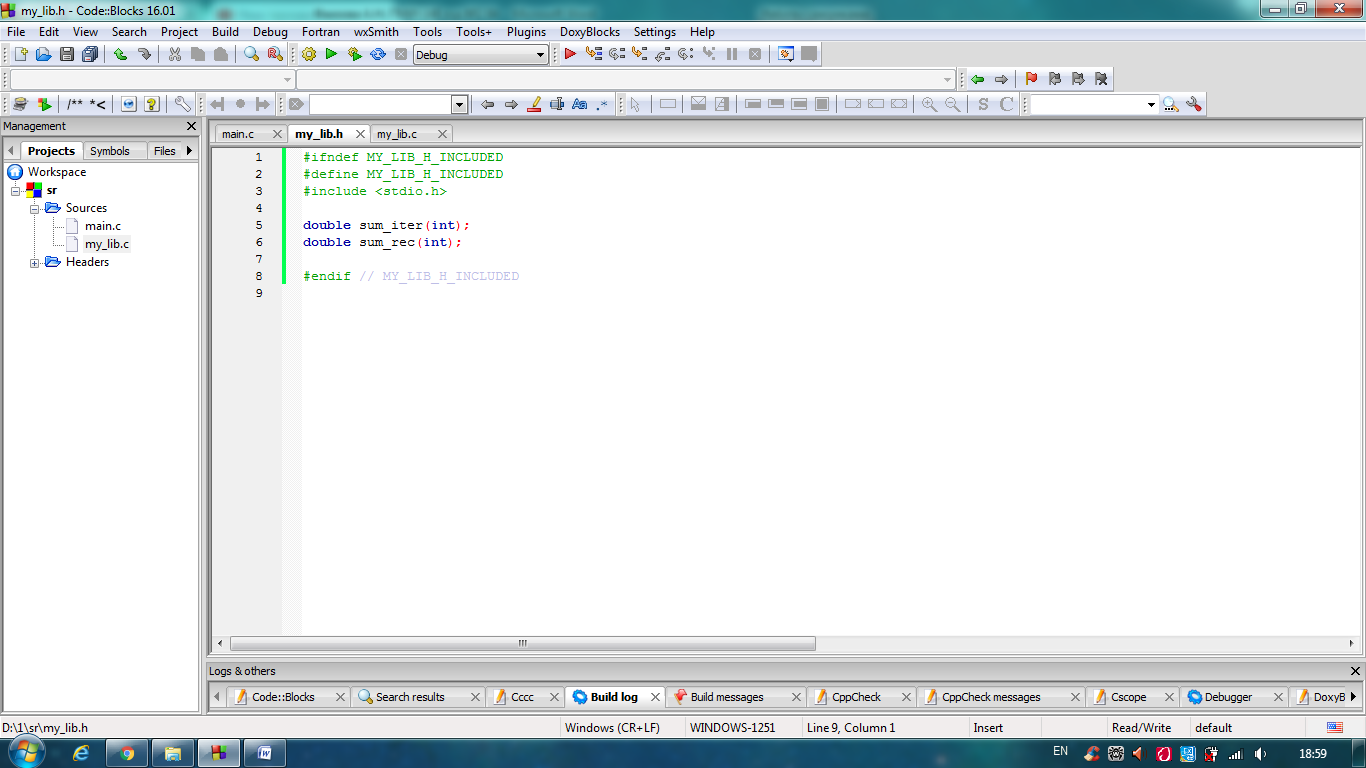
}

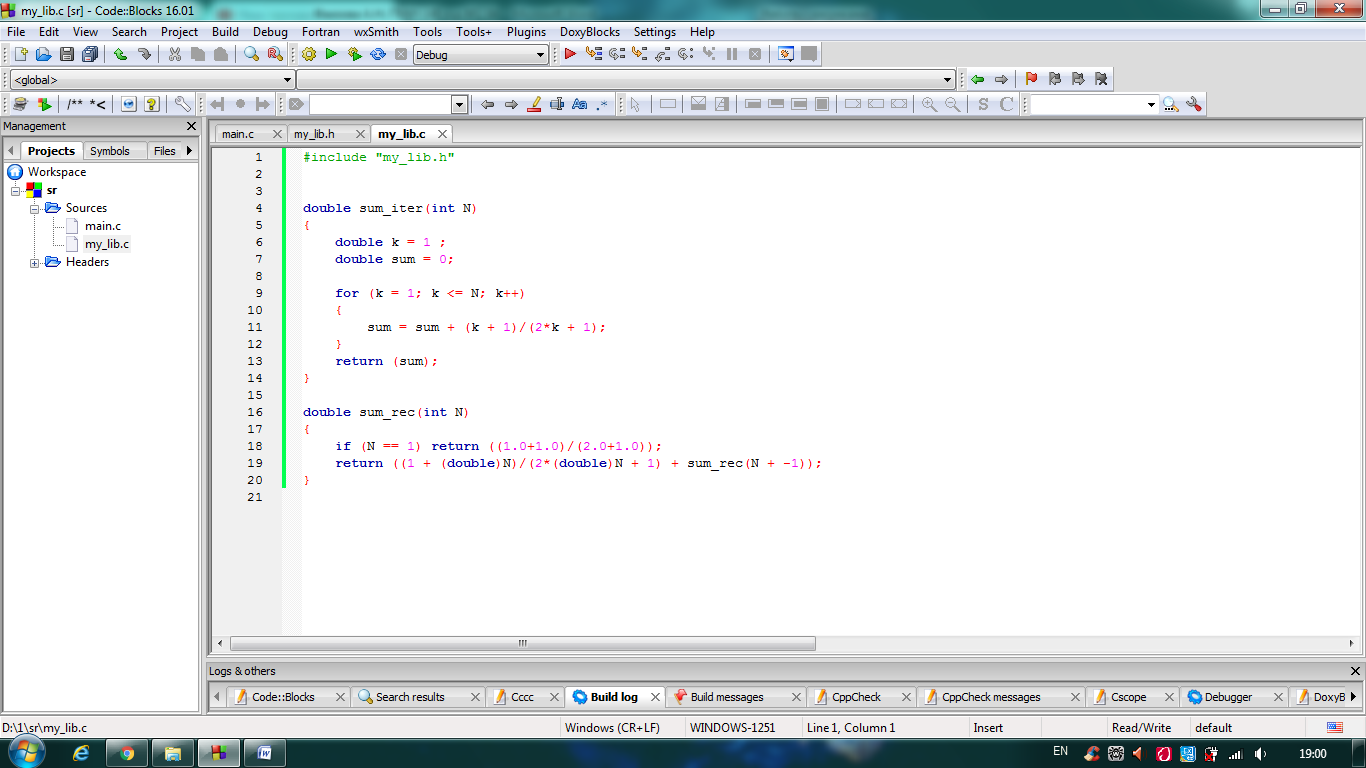
Пример выполнения программы











**Вывод**

На данной лабораторной работе мы ознакомились с принципами разбиения программ на блоки и правилами оформления функций на языке С.

**Список использованной литературы**

1. В.В. Подбельский, С.С.Фомин. Программирование на языке Си. М.: ФиС, 1999, 600 с.
2. Функции [Электронный ресурс] / Бикмеев А. Т. //

<http://bikmeyev-at.ugatu.su/students/CPP/Lab102/index02.html> (Дата обращения 16.10.2017).

1. Керниган Б., Ритчи Д. Язык программирования Си. Пер. с англ., 3-е изд., испр. — СПб.: "Невский Диалект", 2001. - 352 с.
2. Куцый О. Я. Программирование на языке Си. Москва.: МГТУ им. Э. Н. Баумана. 2010. 84 с.