Лабораторная работа №1

«Основные средства и технология разработки консольных программных проектов в интегрированной среде Visual Studio .NET» по Разделу Введение в дисциплину «Введение в информационные технологии»

1.1 Общее задание

1) Изучить основные средства и технологию разработки консольных программных проектов в интегрированной среде **Visual Studio .NET**.
2) Выполнить все шаги по созданию первого консольного проекта на VC++ из п. 2.3 «**Пример выполнения задания для изучения технологии работы в среде Microsoft Visual Studio .NET 2010**»
3) Оформить отчет по ГОСТУ и представить его преподавателю.
4) Ответить на контрольные вопросы, замечания преподавателя по работе и на заданные им вопросы по теме.
5) Получить отметку о выполнении и защите работы

1.2 Содержание отчета

Титульный лист с указанием номера, темы и названия работы, группы и Ф.И.О студента, Ф.И.О преподавателя.

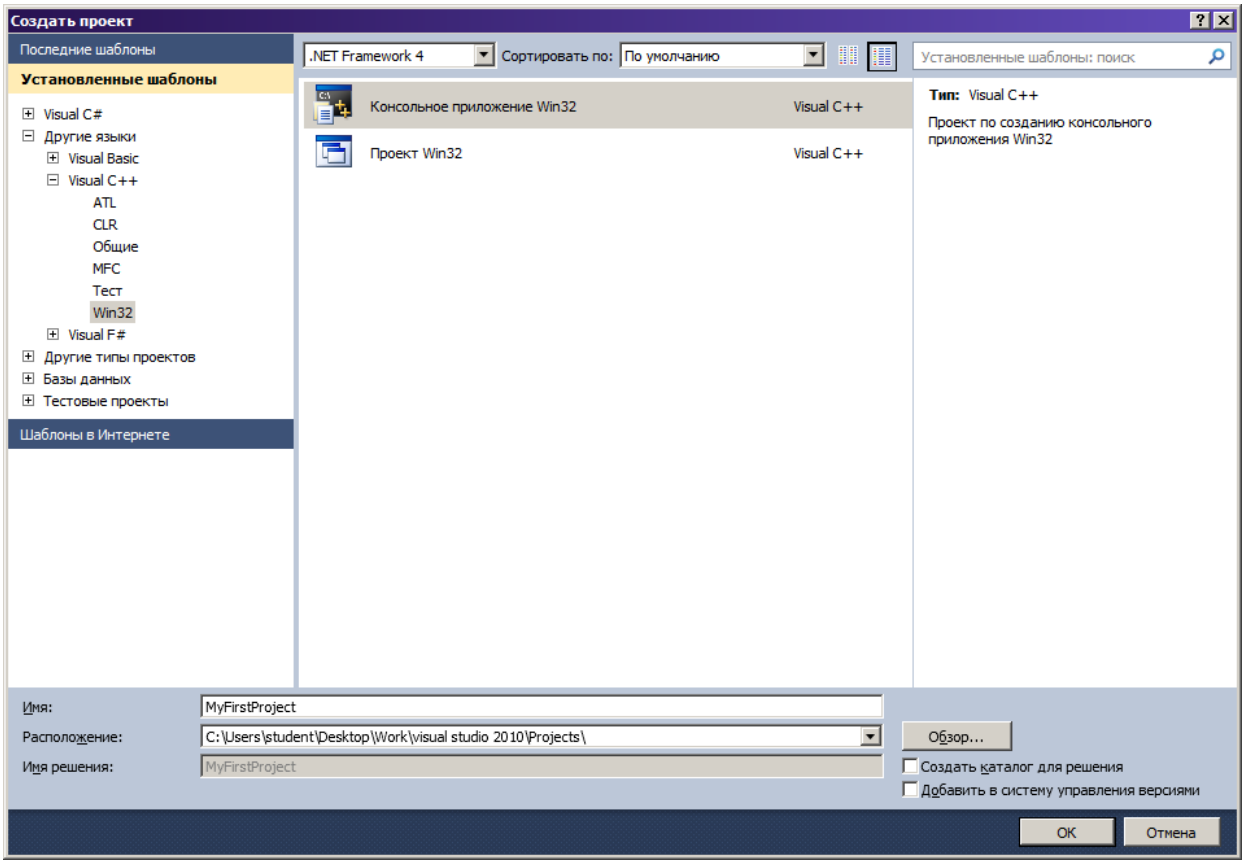
1) **Выполнение задания для изучения технологии работы в среде MS Visual Studio .NET 2010**
   (Рисунки и скриншоты этапов разработки приложения в соответствии с п. 1.3 с пояснениями к ним).

1.3 Пример выполнения задания для изучения технологии работы в среде MS Visual Studio .NET 2010

1) **Создание папки на магнитном носителе**
   Создайте на диске свою ***папку для размещения своих проектов***.
2) **Создание нового проекта**
   2.1) Найдите на рабочем столе компьютера пиктограмму ***Microsoft Visual Studio.Net*** и запустите программу.
   На экране появится окно ***Начальная страница Microsoft Visual Studio.***
   Вам предстоит создать новый проект, который объединяет файлы, необходимые для создания готовой к использованию версии

программы. Любая программа в среде VS, даже если она состоит из одной маленькой функции ***main***, создается как проект (***project***). Проект можно создать двумя способами. Воспользуемся одним из них

2.2) В окне ***Начальная страница*** щелкнем по кнопке ***Создать: Проект...*** На экране появится окно ***Создать проект***.



2.3) В окне ***Создать проект*** из списка установленных шаблонов проектов необходимо выбрать ***Visual C++***, а затем указать тип проекта, выделив шаблон ***Консольное приложение Win32***. В этом же окне надо указать имя и расположение вновь создаваемого проекта.

2.4) В поле ***Имя*** укажите имя проекта, например, ***MyFirstProject*** или любое другое допустимое имя.

2.5) В поле ***Расположение*** необходимо указать путь к папке, в которую вы хотите поместить ваш проект. Этот путь можно прописать вручную, но разумнее щелкнуть по кнопке ***Обзор…***, во вновь выпавшем окне ***Расположение проекта*** выбрать созданную ранее папку (для работы в лабораторном классе это должна быть папка

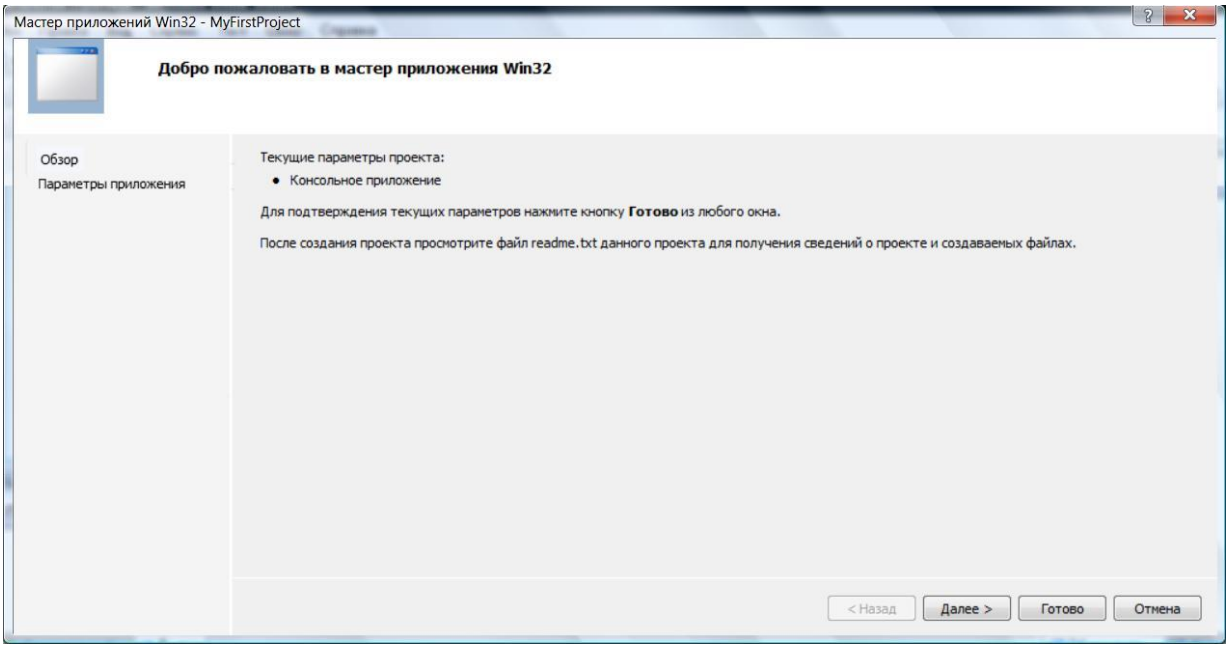
Общие документы), нажать кнопку ***Выбор папки***, и вернуться в окно **Создать проект**.

2.6) Проверьте, у вас должно быть установлено:

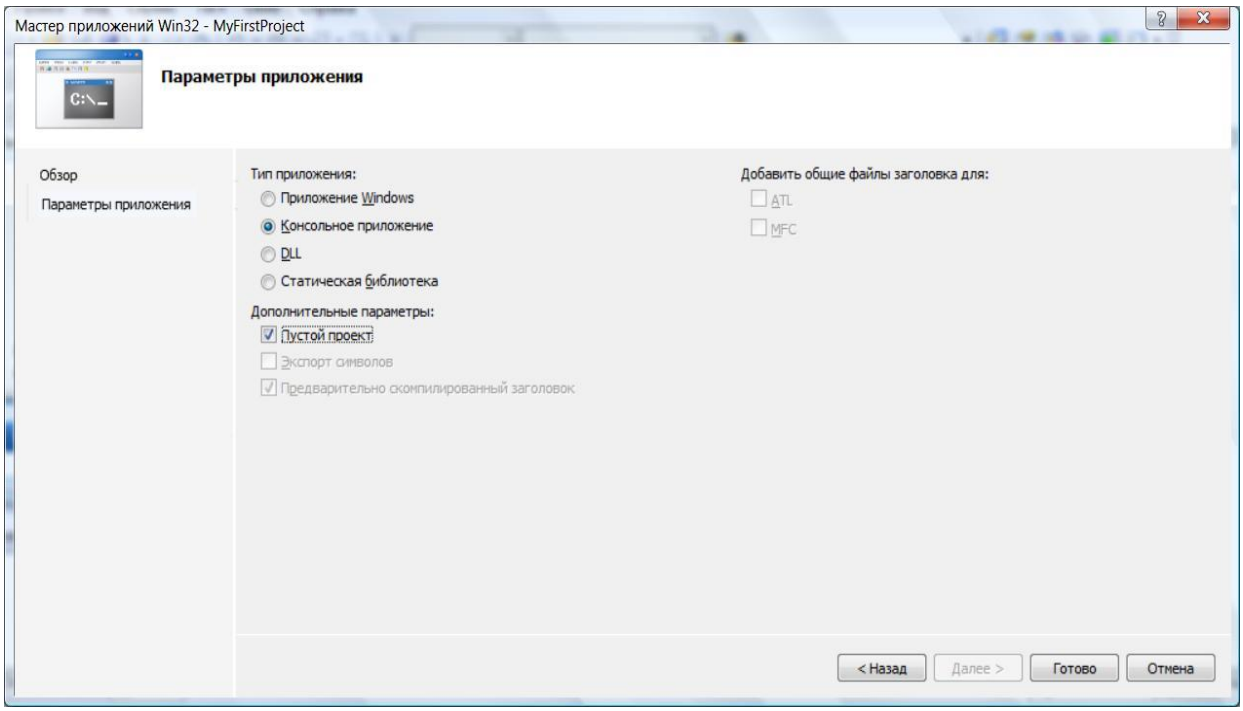
- в окне ***Шаблоны*** – шаблон ***Консольное приложение Win32***;
- в текстовом поле ***Имя*** указано имя создаваемого вами проекта;
- в текстовом поле ***Расположение*** указан путь к папке, в которой вы хотите создать ваш проект;
- переключатель ***Создать каталог для решения*** выключен (сброшена галочка);
- поле ***Имя решения*** недоступно.

2.7) Если все в порядке, нажмите кнопку ***OK***.

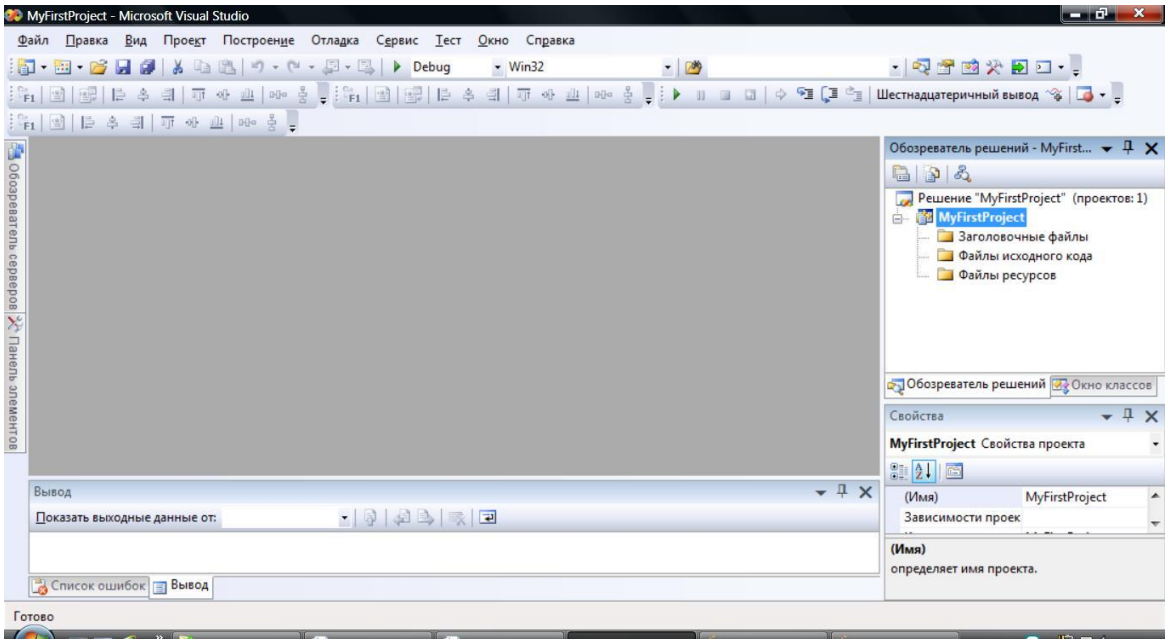
2.8) На экране появится окно ***Мастер приложений Win32 – MyFirstProject***.



2.9) Нажмите кнопку ***Далее >***.
В окне появятся установленные параметры приложения.

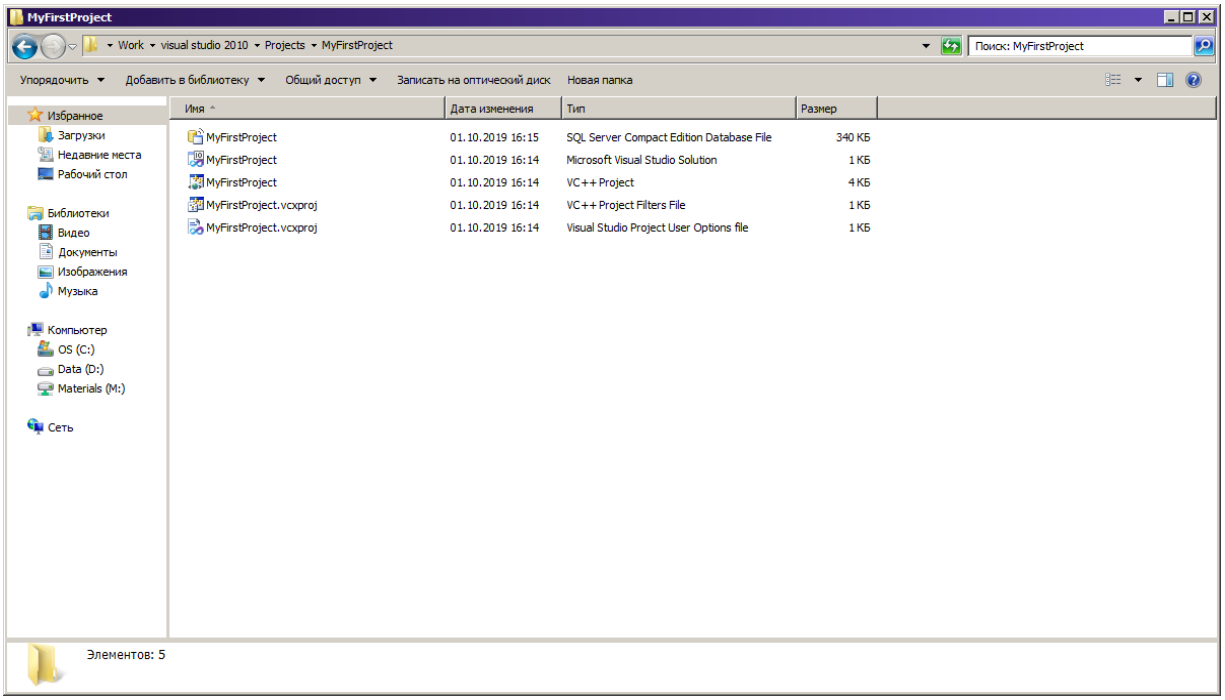


2.10) В группе флажков ***Дополнительные параметры*** установите флажок ***Пустой проект*** и нажмите кнопку ***Готово***.
2.11) На экране появится окно рабочей среды проекта ***MyFirstProject*** – ***Microsoft Visual Studio.Net.***
2.12) Если настройки среды таковы, что у вас на экране не появилось окно ***Обозреватель решений – MyFirstProject***, то необходимо выбрать в
2.13) меню среды элемент ***Вид*** и далее выполнить команду ***Обозреватель решений***. В рабочей среде появится окно ***Обозреватель решений***.



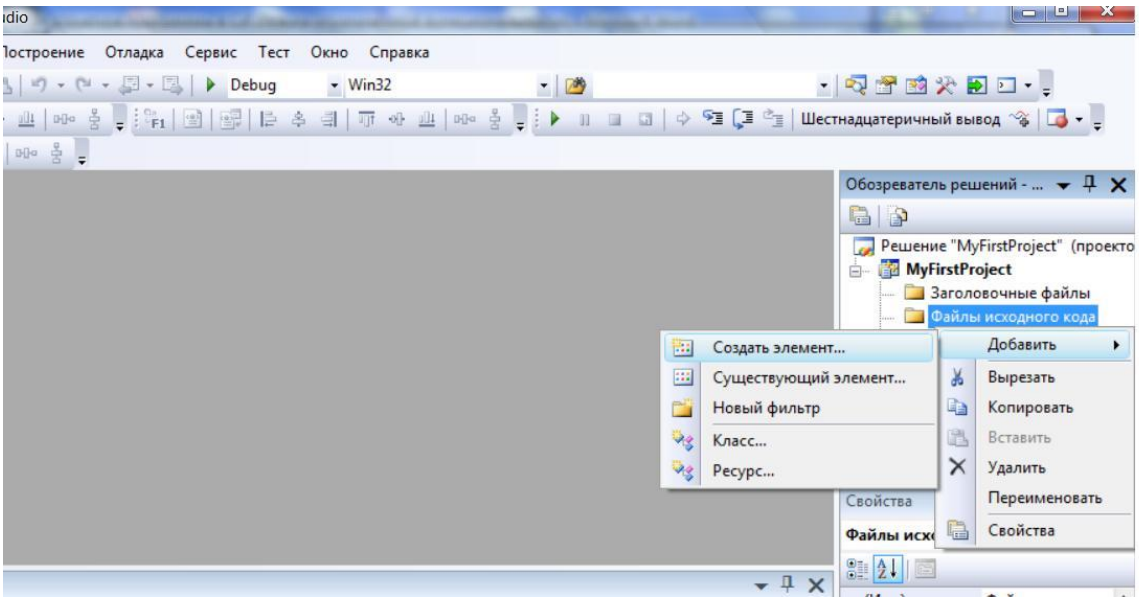
2.14) Решение содержит один проект ***MyFirstProject***, в который включены три папки: ***Заголовочные файлы***, ***Файлы исходного кода***, ***Файлы ресурсов*** (в некоторых версиях VS2010 также может быть четвертая папка ***Внешние зависимости***). Эти папки пусты, так как проект был создан с опцией ***Пустой проект***.

2.15) Прежде чем продолжить работу, сверните окно среды ***Microsoft Visual Studio.Net*** и загляните в папку ***MyFirstProject***, созданную мастером приложений для вашего проекта. Там будут расположены пока только служебные файлы, в которых система хранит свою служебную информацию о проекте. Папок с именами ***Заголовочные файлы***, ***Файлы исходного кода***, ***Файлы ресурсов*** вы не увидите. Эти папки виртуальные, их система заводит только в окне ***Обозреватель решений*** для удобства работы пользователя с его файлами.

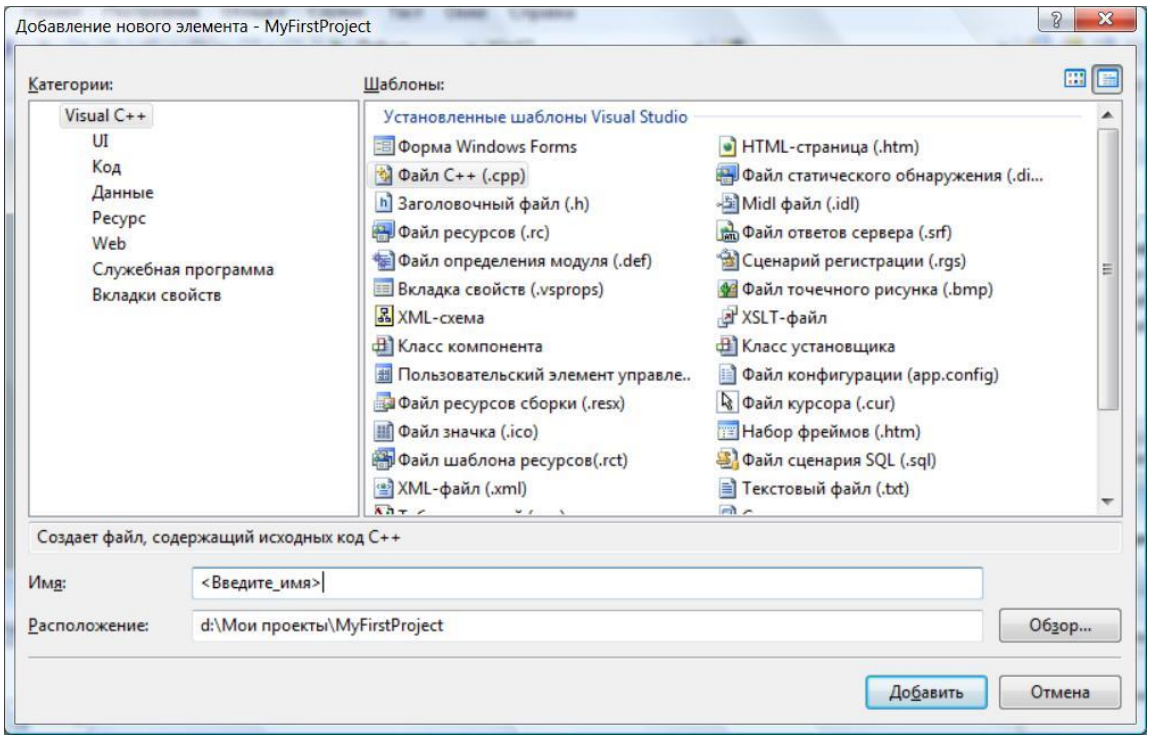


3) Добавление к проекту нового файла с исходным текстом

3.1) Разверните снова главное окно ***Microsoft Visual Studio.Net.*** Вам необходимо в папке ***Файлы исходного кода*** вашего решения создать файл с расширением ***.cpp***, в который вы запишете текст своей первой программы. Для этого щелкните правой кнопкой мыши по папке ***Файлы исходного кода*** и в появившемся контекстном меню выполните команду ***Добавить,*** а затем команду ***Создать элемент….***



В результате на экране появится окно ***Добавление нового элемента – MyFirstProject***.



3.2) Из списка допустимых **Шаблонов** выделите шаблон ***Файл C++ (.cpp)***, введите в текстовое поле ***Имя*** – имя для вашего файла, например, ***MyFirst*** и нажмите кнопку ***Добавить.***

После этого в списке файлов папки ***Файлы исходного кода*** окна ***Обозреватель решений – MyFirstProject*** появится новый файл ***MyFirst.cpp*** и откроется окно редактора исходного кода C++ с этим же именем ***MyFirst.cpp***.

3.3) В окне редактора можно вводить текст вашей программы. В этом задании вы должны просто скопировать в окно текст файла **MyFirst.cpp**.

```
/*      файл MyFirst.cpp содержит программу,   1
 состоящую из одной функции main                2
 единственное действие этой программы -        3
 ввести два числа, вычислить их сумму,         4
 вывести на экран эту сумму и                  5
 приветствие "ПОКА !"                          6
                                               7 */
// ======================================     8
#include <iostream>                          //  9
using namespace std;                         // 10

int main ()                                  // 11
{                                            // 12
  setlocale(LC_ALL,"rus");                   // 13
  int A;                                     // 14
  float B;                                   // 15
  cout <<"Введите 2 числа:\t"                // 16
       <<"\nпервое - целое\t";               // 17
  cin>>A;                                    // 18
  cout<<"второе- любое\t";                   // 19
  cin>>B;                                    // 20
  float Summa;                               // 21
  Summa     = A + B;                         // 22
  cout<<"Сумма   =    " <<  Summa;           // 23
  cout<<endl<<"\tПОКА !";                    // 24
  system("PAUSE");                           // 25
  return 0;                                  // 26
}                                            // 27
```

4) Компиляция, компоновка и выполнение проекта

Компиляция (compilation) – это преобразование программы или ее отдельного модуля, текст которых составлен на языке *программирования высокого уровня (исходная программа, исходный модуль – это файл с расширением **.cpp**)* в программу или модуль на *машинном языке* или на языке, близком к машинному (получают *объектный модуль – файл с расширением **.obj**).* Компиляцию осуществляет специальная программа – **компилятор (compiler)**, которая является неотъемлемой частью системы программирования. На вход компилятора поступает *исходный модуль (файл **.cpp**)*, который после компиляции преобразуется в объектный модуль (*файл **.obj**)*

Объектный модуль не может быть исполнен, его местоположение в оперативной памяти еще не известно (не определено). Компилятор вырабатывает только *относительные адреса* связи с другими модулями. В дальнейшем их предстоит заменить конкретными адресами (*абсолютными адресами*) той части оперативной памяти, в которой этот модуль будет выполняться.

Результат компиляции – это промежуточная форма программных модулей, к которым впоследствии необходимо присоединить библиотечные модули, содержащие *стандартные подпрограммы и процедуры*, а если нужно, то можно добавить любые другие модули, написанные самим пользователем, и скомпилированные в объектные модули, возможно даже с других языков высокого уровня.

Существуют различные виды компиляторов:

- *интерпретирующие* (пошаговые), осуществляющие последовательную независимую компиляцию каждой отдельной инструкции *исходной программы;*
- *оптимизирующие,* осуществляющие повышение эффективности объектных модулей, например, за счет вынесения из циклов последовательности команд, результаты действий которых не меняются при повторении циклов;
- *отладочные,* облегчающие пользователю отладку программ.

Компоновка, редактирование связей (linking, linking editing) – это процесс сборки *загрузочного модуля (исполняемого файла)* из полученных в результате раздельной компиляции *объектных модулей* с одновременным автоматическим поиском и присоединением *библиотечных подпрограмм и процедур.* В процессе компоновки программа собирается в единое целое непосредственно в оперативной памяти в файл, готовый к работе (*загрузочный модуль - файл с расширением* ***.exe****).*

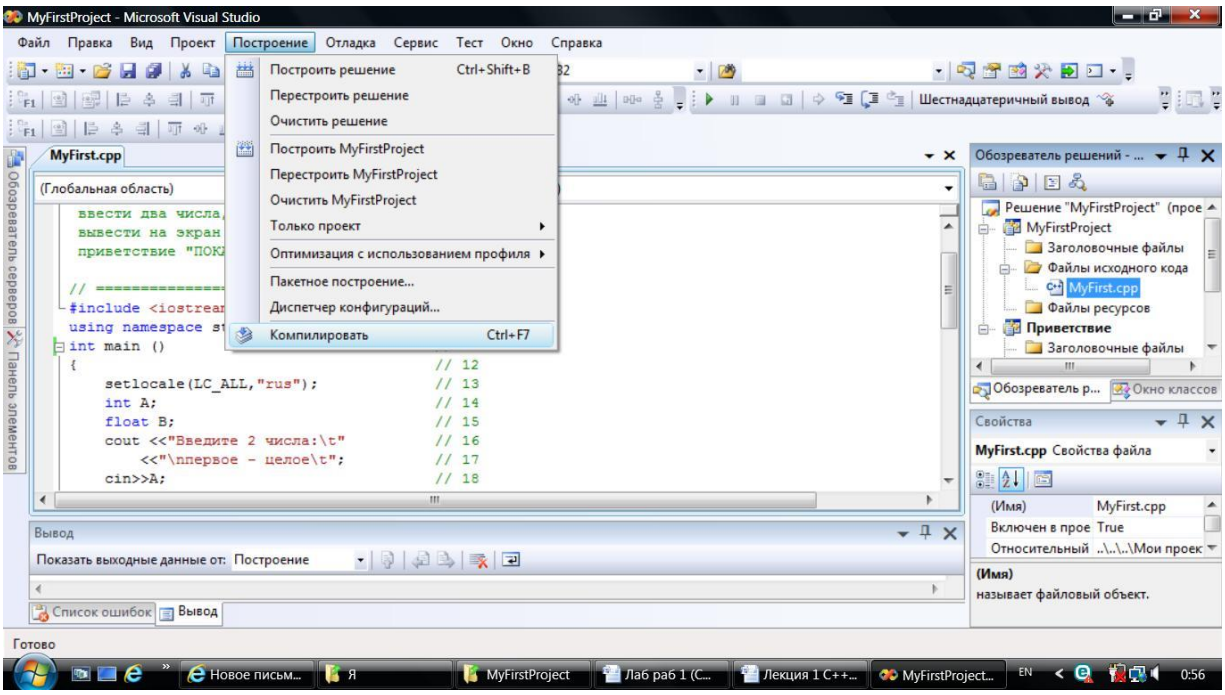
Работу по компоновке программы выполняет программа **компоновщик (linker).** Эта программа выполняет следующие основные функции:

- распределяет пространство оперативной памяти для программы;
- связывает вместе части программы, представленные отдельными объектными модулями (*файлами с расширением* ***.obj****);*
- настраивает адреса подготовленной программы, заменяя все *относительные адреса,* выработанные компилятором, соответствующими *абсолютными адресами* фактически распределенной памяти.

Компоновщики бывают в двух реализациях:

- компоновщики, которые готовят загрузочный файл; этот файл при необходимости может быть загружен в оперативную память для исполнения;
- компоновщики, которые готовят загрузочный файл, сразу физически размещают подготовленную версию машинного кода программы в памяти и передают управление на первую команду программы для непосредственного исполнения.

Эти операции могут быть выполнены с помощью команд меню ***Построение***.

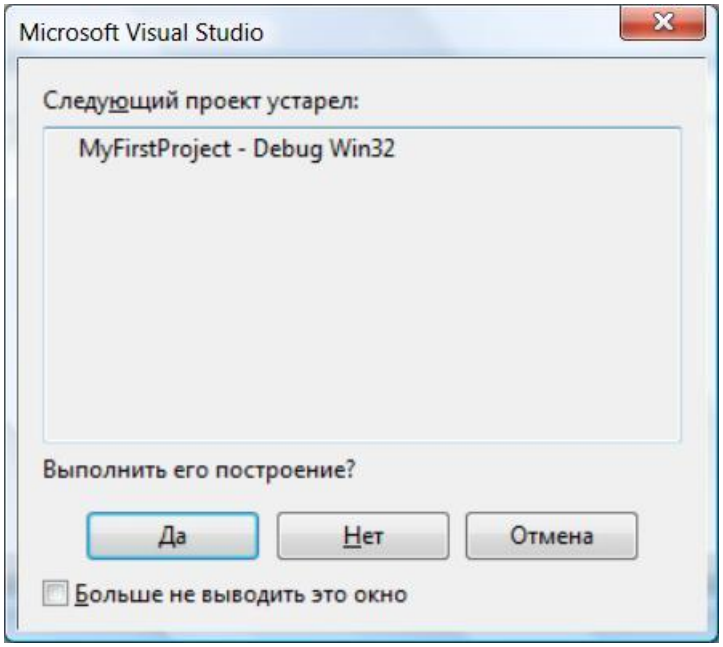


Краткое описание основных команд этого меню:
• ***Компилировать*** – компиляция выбранного файла, результаты компиляции отображаются в окнах ***Список ошибок*** и ***Вывод***. • ***Построить*** - компоновка проекта. Компилируются все файлы, в которых произошли изменения с момента последней компоновки. После компиляции происходит сборка всех объектных модулей, включая библиотечные, в результирующий исполняемый файл. Сообщения об ошибках компоновки выводятся в окна ***Список ошибок*** и ***Вывод***. Если обе фазы компоновки завершились без ошибок, то созданный исполняемый файл с расширением ***.exe*** может быть запущен, однако автоматически запуск этого файла не осуществляется. • ***Перестроить*** – делается то же, что и в команде ***Построить***, но при выполнении этой команды компилируются все файлы проекта независимо от того, были ли в них изменения. Эти операции могут быть выполнены и с помощью меню ***Отладка***, его команд ***Начать отладку*** или ***Запуск без отладки.*** Эти две команды делают все то же, что и команда ***Построить***, сразу же запуская файл с расширением ***.exe***.

5) Выполнение программы

5.1) Запустите ваш проект, выполнив команду ***Начать отладку*** из элемента меню ***Отладка***.

5.2) Если вы не выполняли отдельно процедуру компоновки программы (команда ***Построить***), то появляется выпадающее диалоговое окно с предложением выполнения построения проекта.



Согласитесь, на выполнение построения, нажав кнопку ***Да***.

Так как программа, которую вы скопировали, не имеет ошибок, то появляется черное консольное окно ***[Выполнение]***, в котором начинает работать ваша программа.

5.3) Введите два числа – первое обязательно целое, второе любое. Если второе число будет вещественным, то разделителем целой и дробной части должен быть символ точка (например, 5.8). Получите результат.

5.4) Сверните окно среды ***Microsoft Visual Studio.Net*** и разверните окно папки с проектом. В вашей папке появилась новая папка ***Debug***. В этой папке появились новые файлы, среди них файл типа ***Приложение MyFirstProject.exe*** (исполняемый файл). Запустите

его, щелкнув по его пиктограмме. При этом снова запускается на выполнение ваша программа.

5.5) Разверните снова окно среды ***Microsoft Visual Studio.Net.***
Закройте свой проект, выбрав в элементе меню ***Файл*** команду ***Закрыть решение***. Закрыть проект можно также, просто закрыв окно среды.

6) Открытие существующего проекта

6.1) Существующий проект можно открыть из окна ***Начальная страница***.

Воспользуемся другим способом.

В элементе меню ***Файл*** выполните команду ***Открыть***, а затем ***Решение*** или ***Проект….***

В открывшемся окне ***Открыть проект*** найдите свою папку. Откройте ее и щелкните по одному из файлов ***MyFirstProject*** - либо типа ***Microsoft Visual Studio Solution***, либо ***VC++ Project.***

Ваш проект вновь откроется.

Научитесь общаться с компилятором при появлении ошибок в тексте вашей программы

Если в программе допущено прямое нарушение синтаксических правил языка C++, то на этапе компиляции возникает диалоговое окно с вопросом: “В ходе построения произошли ошибки. Продолжить?”. Так как продолжение бессмысленно, то следует ответить «Нет». Вы, конечно, можете ответить «Да», компилятор по своему разумению попытается исправить ошибку, но к чему это приведет, вот в чем вопрос.

Компилятор выдает сообщение об ошибках (*error*) в окне ***Список ошибок***. Такие ошибки помечаются в окне ***Список ошибок*** восклицательным знаком красного цвета. Иногда в этом окне компилятор выводит предупреждающее сообщение (w*arning*). Такое сообщение является признаком наличия в тексте места, которое, строго говоря, не является нарушением синтаксических правил языка, но достаточно необычно, что само по себе может свидетельствовать об ошибке. Своими предупреждениями компилятор как бы говорит: “Вы уверены, что действительно имели в виду именно это?”. На эти ошибки надо обращать внимание и думать над ними! Они могут привести к неправильному решению на этапе выполнения.

Если вы работаете над новой программой, то компиляцию и компоновку следует делать раздельно, так как при наличии ошибок- w*arning* компоновка при запуске командами ***Построить*** или ***Начать***

отладку не будет приостановлена, а это может привести к неожиданным результатам на этапе выполнения.

Целью дальнейшей работы является изучение реакции компилятора на ошибки в тексте программы. Для этого внесите ряд указанных ошибок в текст программы и прочитайте сообщения, выдаваемые компьютером.

Удаление той или иной строки осуществляйте с помощью комментария //. Такой метод позволит вам быстро восстановить текст программы после анализа сообщений.

После выполнения каждого пункта возвращайте текст программы к исходному состоянию!

6.2) Добавьте в оператор определения объекта в строке **[15]** через запятую еще одно имя, например, объекта **Z**. Откомпилируйте проект командой ***Компилировать*** из меню ***Построение***. Прочитайте сообщение ***warning*** в окне ***Список ошибок***. Локализуйте место обнаруженной ошибки двойным щелчком по строке сообщения. Слева в окне редактора появится черная стрелка-указатель на строку предполагаемой ошибки. Не реагируйте на это предупреждение и выполните команду ***Начать отладку*** из меню ***Отладка*** (Полезно запомнить горячую клавишу ***F5***, соответствующую команде ***Начать отладку.*** Также можно щелкнуть на кнопке ***Начать отладку***, расположенной на Главной панели инструментов).

Ваша программа правильно работает?

6.3) В операторе присваивания в строке **[22]** замените операцию присвоить = на операцию равно **==**. Выполните сразу команду ***Начать отладку***. Как отработала ваша программа? Сообщение ***warning*** не остановило компоновку программы, хотя ошибка, на которую указывало сообщение, существенно повлияла на работу программы. Выполнение команды ***Начать отладку*** в такой ситуации было недопустимо. Выполните команду ***Компилировать*** и прочитайте сообщение ***warning*** в окне ***Список ошибок.***

Исправьте ошибку.

6.4) Уберите из текста программы строку **[25]**. Сделайте это, не стирая ее, а поставив два символа // комментария в начале этой строки**. Компилятор «не видит» комментарии!** Выполните свою программу. Введите запрашиваемые вашей программой исходные данные. Где ответ? Вы не видите его. Его закрыло окно редактора. Таким образом, оператор ***System("PAUSE")*** используется для того, чтобы с его помощью организовать задержку, и программа не будет выполнять следующие действия, пока пользователем не будет введен любой символ. Программа не будет заканчивать свою работу по закрывающей тело функции **main** скобке **}**, пока пользователь не

рассмотрит результат работы программы на экране, а после этого не введет символ (в этой программе – неважно какой).
Раскомментируйте строку **[25]**.

6.5) Закомментируйте оператор определения объекта на строке **[15]**. Откомпилируйте проект, прочитайте сообщения в окне ***Список ошибок*** и разберитесь, почему последовали именно такие сообщения. Обратите внимание, что на одну ошибку последовало несколько сообщений.
Раскомментируйте строку **[15]**.

6.6) Закомментируйте строку **[9]**. Откомпилируйте проект, прочитайте сообщения в окне ***Список ошибок*** и разберитесь, почему последовали именно такие сообщения. Обратите внимание, что на одну ошибку последовало несколько сообщений.
Раскомментируйте строку **[9]**.

6.7) Закомментируйте инструкцию на строке **[10]**. Откомпилируйте проект, прочитайте сообщения в окне ***Список ошибок*** и разберитесь, почему последовали именно такие сообщения. Обратите внимание, что на одну ошибку последовало несколько сообщений. В чем и почему совпадают эти сообщения с сообщениями при удалении строки **[9]**?
Раскомментируйте строку **[10]**.

6.8) Подведите курсор мыши к именам объектов ***cout*** и ***cin***. Вам показаны типы этих объектов. Измените операцию >> в инструкции на строке **[20]** на операцию <<. Откомпилируйте проект, прочитайте сообщения в окне ***Список ошибок*** и разберитесь, почему последовали именно такие сообщения.
Верните строку **[20]** в исходное состояние.

6.9) Переставьте оператор определения объекта со строки **[15]** на любую строку после строки **[20]**. Откомпилируйте проект, прочитайте сообщения в окне ***Список ошибок*** и разберитесь, почему последовало именно такое сообщение.
Верните оператор определения объекта в исходное место.

6.10) Закройте ваш проект, выбрав в меню ***Файл*** команду ***Закрыть решение***.

7) Добавление к проекту существующего файла с исходным текстом

7.1) Создайте новый проект с именем ***MyFirstSumma***. Для этого в меню ***Файл*** выполните команду ***Создать***, затем команду ***Проект…***, а далее повторите пункты 3 и 4 этой лабораторной работы.

7.2) Если настройки таковы, что у вас на рабочем столе ***Microsoft Visual Studio.Net*** не появилось окно ***Обозреватель решений*** –

MyFirstSumma, то выберите в меню ***Вид*** пункт ***Обозреватель решений*** и раскройте его**.**
Окно появится.

7.3) Добавим в наш новый проект уже существующий файл ***MyFirst.cpp*** из предыдущего проекта. Щелкните правой кнопкой мыши по папке ***Файлы исходного кода*** и в появившемся контекстном меню выберите команду ***Добавить,*** затем команду ***Существующий элемент….***
В результате будет отображено окно ***Добавление существующего элемента - MyFirstSumma***. В папке с файлами предыдущего проекта ***MyFirstProject*** найдите имя файла ***MyFirst.cpp*** (этот уже существующий файл мы хотим добавить во вновь созданный проект) и нажмите кнопку ***Добавить.*** Добавление существующего (или нового) элемента в проект можно также осуществить, выбрав соответствующий пункт в меню ***Проект.*** После этого в списке файлов папки ***Файлы исходного кода*** окна ***Обозреватель решений – MyFirstSumma*** появится обозначение нового файла ***MyFirst.cpp*** и откроется окно редактора с этим же именем ***MyFirst.cpp***.

7.4) Запустите этот проект, выполнив команду ***Начать отладку***.

7.5) Закончите работу, закрыв окно среды ***Microsoft Visual Studio.Net***.

Приложение 1.1 Отладка консольных проектов

На этапе отладки проверяется правильность работы программы. Ошибки, возникающие в процессе создания программы, могут быть вызваны и некорректностью метода или алгоритма, и неправильным применением самих средств языка программирования. В целом типы ошибок принято разделять на два неравнозначных класса.

Один из них – это класс синтаксических ошибок, то есть ошибок, связанных с неправильной записью или употреблением языковых конструкций. Эти ошибки легко исправимы, так как соответствующее программное обеспечение – компилятор – осуществляет автоматический контроль синтаксической правильности программ пользователя, а с помощью контекстно-зависимой помощи можно получить как разъяснения об ошибке, так и узнать правильный вид языковой конструкции.

Другой вид ошибок, действительно представляющий проблему программирования, – смысловые ошибки. Обнаружение и исправление их, что собственно и представляет собой процесс отладки, дело сложное, а порой, как это ни парадоксально звучит, и безнадёжное.

Как определить, что программа имеет смысловую ошибку? В лучшем случае программа не работает, то есть её работа прерывается в некоторый момент, и система выдаёт какое-нибудь туманное сообщение типа «исчезновение порядка числа с плавающей точкой». В худшем случае программа успешно завершает свою работу и выдаёт результаты, отвечающие интуитивным представлениям об их значениях, а о наличии ошибки в программе мы узнаём только после практического внедрения результатов, например, когда по нашим прочностным расчётам построили мост, а он тут же обвалился под собственной тяжестью.

Как обнаружить такие скрытые ошибки? Самый популярный метод – так называемое тестирование. Следует взять такие исходные данные, правильный результат расчёта для которых известен заранее, и выполнить программу с этими данными. Если полученный результат совпадает с известным результатом, то, как говорят, «тест прошёл». Беда в том, что это совсем не означает, что программа не содержит ошибок.

Среда Visual Studio 2010 располагает мощными средствами отладки. Они позволяют контролировать выполнение программы таким образом, что вы можете пошагово выполнять код, строку за строкой, или же запустить его до определенной точки в программе.

В каждой точке программы, где отладчик останавливается, можно посмотреть (и даже изменить) значения переменных, прежде чем продолжить выполнение. Можно изменить исходный код, перекомпилировать, а затем перезапустить программу сначала. Можно даже изменить исходный код посреди пошагового выполнения программы. При переходе к следующему оператору после модификации кода отладчик автоматически перекомпилирует программу перед выполнением следующего оператора.

Мы рассмотрим здесь лишь самые основные способы и средства отладки, использование которых производится с помощью команд пункта меню ***Отладка*** (рисунок П1.1.1).

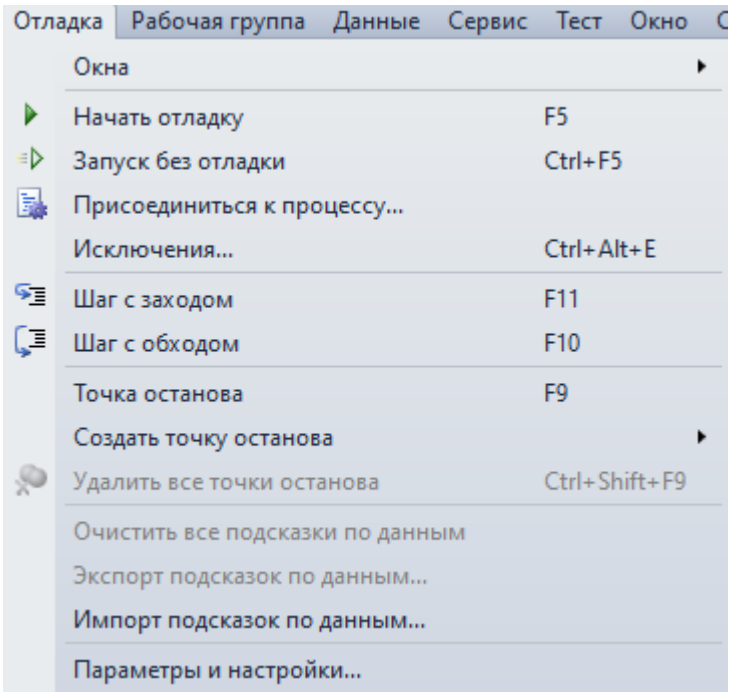


Рисунок П1.1.1 – Команды пункта меню **Отладка**

Отладчик имеет два главных режима выполнения — ***пошаговое выполнение*** кода (т.е. выполнение по одному оператору программы за раз) и ***выполнение до определенной точки***, указанной в исходном коде. Точка в исходном коде, где отладчик приостанавливает выполнение программы, называется ***точкой останова***.

Точка останова — это точка в программе, где отладчик автоматически приостанавливает выполнение в режиме отладки. Можно задать сразу несколько точек останова, чтобы запускать программу и приостанавливать ее выполнение в интересующих местах. В каждой точке останова можно посмотреть содержимое переменных и изменить их, если они имеют не те значения, которые нужно.

Любую программу вполне можно выполнить пошагово, однако для больших программ это непрактично. Обычно нужно лишь увидеть определенную область программы, в которой, как предполагается, может содержаться ошибка. Следовательно, необходимо устанавливать точки останова в местах предполагаемых ошибок и запускать программу так, чтобы она останавливалась на первой точке останова. Затем по желанию можно выполнить одиночные шаги, начиная с этой точки, при этом под одиночными шагами подразумевается выполнение отдельного оператора исходного кода.

Чтобы установить точку останова в начале строки исходного кода, надо щелкнуть на серой колонке слева от строки с оператором, перед которым требуется остановить выполнение. В ней появится красный кружок,

указывающий на точку останова в строке (рисунок П1.1.2). Удалить эту точку можно с помощью повторного щелчка.

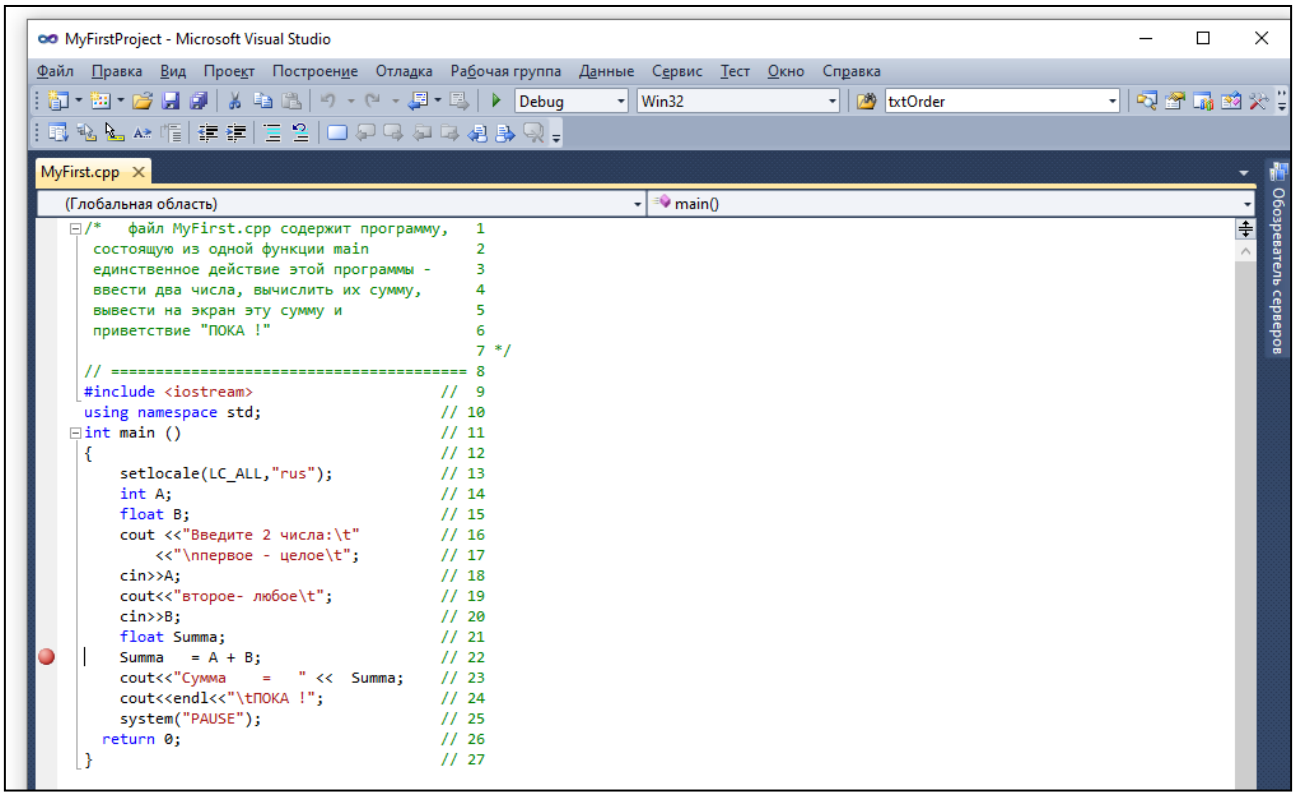


Рисунок П1.1.2 – Установка точки останова

После приостановки выполнения программы в точке останова можно начать пошаговое выполнение программы, используя команды пункта меню ***Отладка*** (рисунок П1.1.1) ***Шаг с заходом*** (клавиша F11) и ***Шаг с обходом*** (клавиша F10). Разница между этими командами состоит в том, что при шаге с заходом выполняется вход в вызываемую функцию (если такой вызов присутствует в операторе), а при шаге с обходом эта функция выполняется за один шаг, и управление передается на следующий оператор программы. При отсутствии вызова функции в точке останова, как на рисунок П1.1.2, обе команды действуют одинаково.

Находясь в точке останова или в любой другой точке в процессе пошагового выполнения программы, можно просматривать значения переменных в окнах ***Видимые*** и ***Локальные***, которые можно активировать из пункта меню ***Отладка\Окна*** (рисунок П1.1.3). В окне ***Видимые*** отображаются переменные, используемые вокруг текущей точки останова. В окне ***Локальные*** отображаются переменные, определенные в локальной области, которая обычно является текущей функцией или блоком. В простых случаях содержимое этих окон практически идентично.

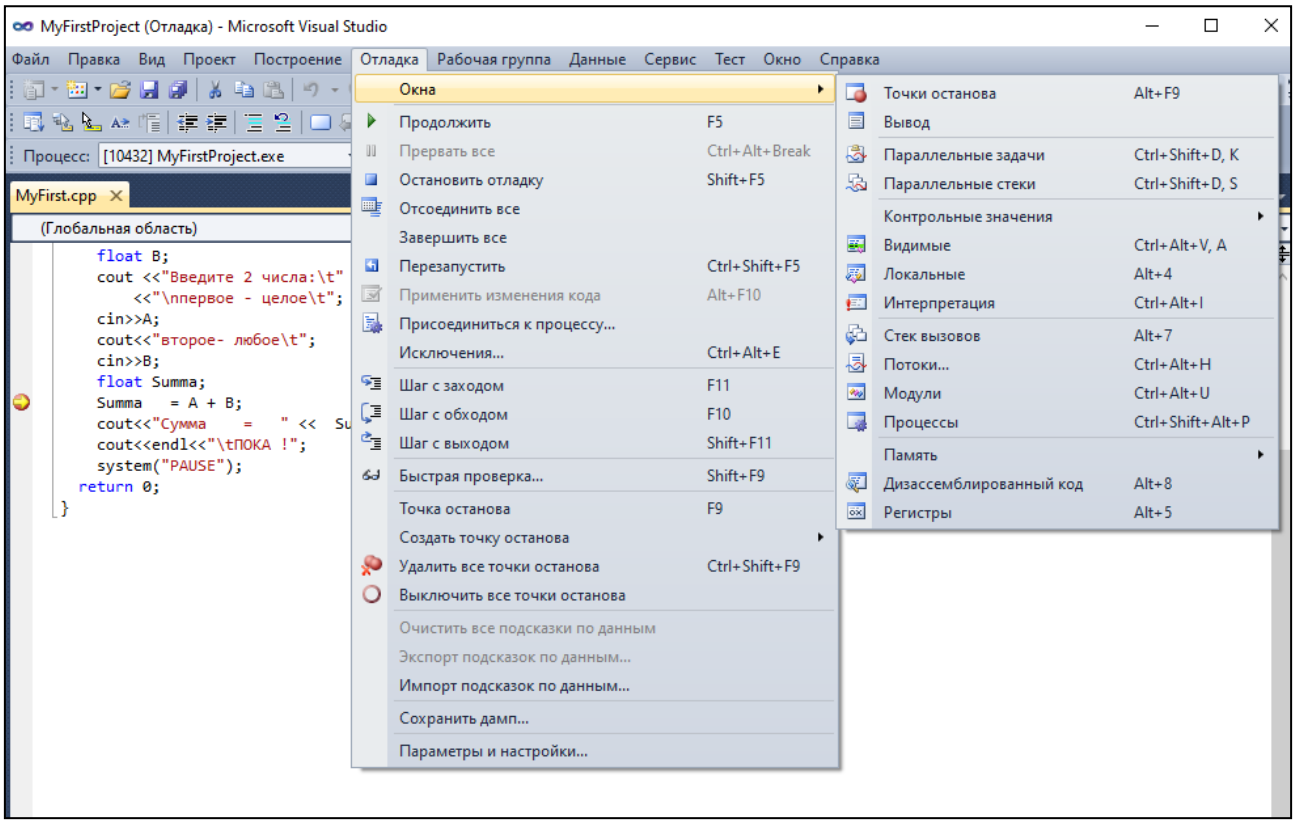


Рисунок П1.1.3 – Активизация окон ***Видимые*** и ***Локальные***

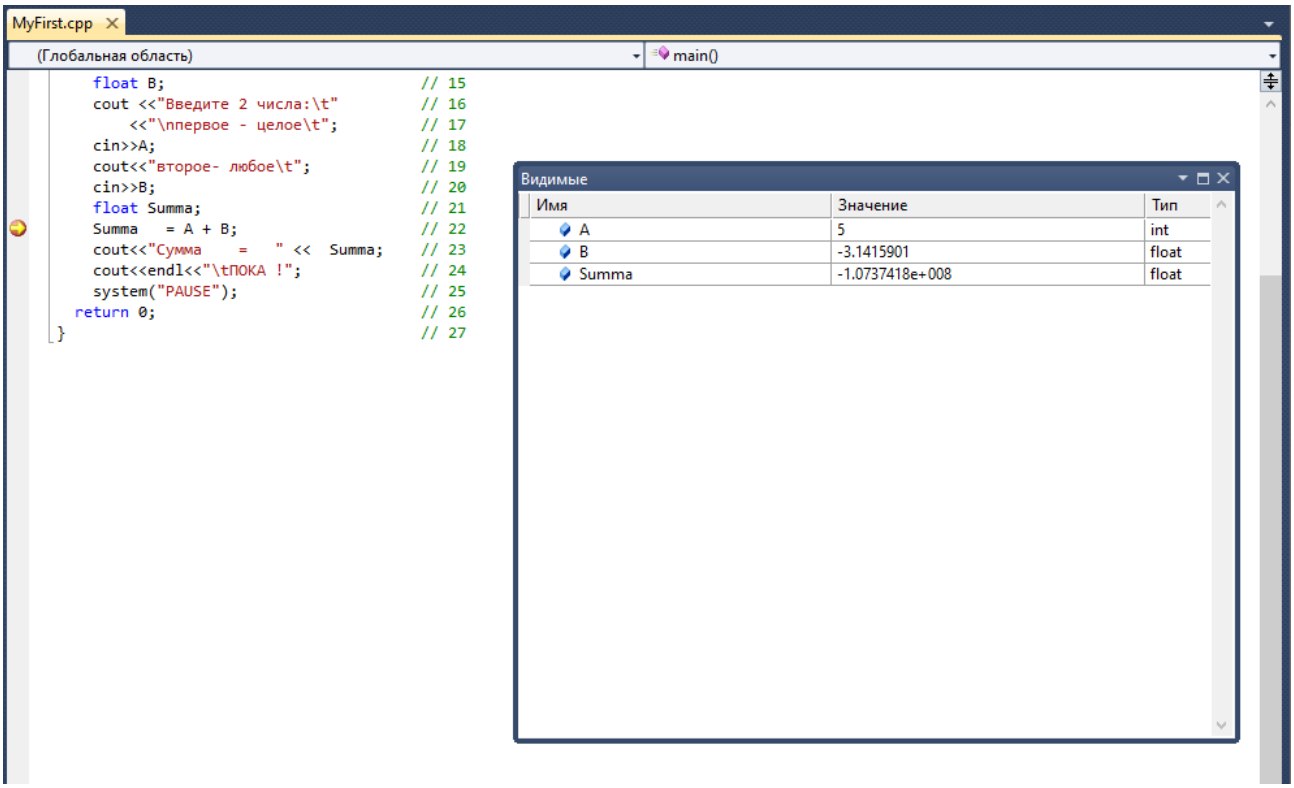


Рисунок П1.1.4 – Окно ***Видимые*** перед выполнением оператора в строке 22

На рисунке П1.1.4 показано содержимое окна ***Видимые*** при останове в строке 22 программного кода. Переменные ***A*** и ***B*** содержат введенные пользователем значения. В переменной ***Summa*** содержится некоторое

произвольное значение, так как оператор суммирования ***A*** и ***B*** еще не выполнялся.

Значение некоторой переменной можно также посмотреть, наведя указатель мыши на эту переменную в тексте программы (рисунке П2.1.5).

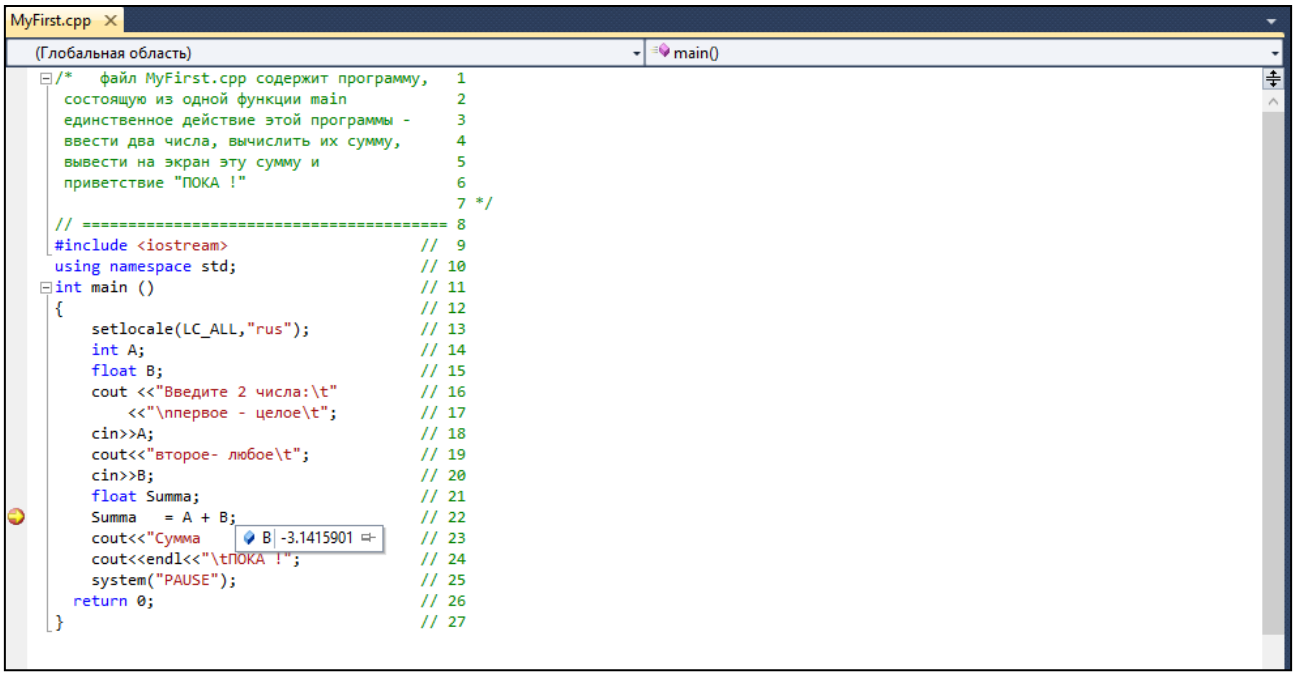


Рисунок П1.1.5 – Просмотр значения переменной наведением указателя мыши

Приложение 1.2 Тестирование программных проектов

Процесс разработки программных проектов можно выразить следующей формулой:

Разработка проектов = Изготовление + Доказательство правильности

Эта простая формула указывает на то, что после того, как программный код написан, необходимо убедиться в его правильности. Следует заметить, что наличие ошибок в только что разработанном программном коде – это вполне нормальное и закономерное явление. Практически невозможно разработать реальный, достаточно сложный программный проект без ошибок. Кроме того, нельзя делать вывод, что он работает правильно, лишь на том основании, что он выполняется, и выдает результаты, поскольку эти результаты еще не обязательно правильные. В программном проекте может оставаться большое количество логических и смысловых (семантических) ошибок. Ответственные участки (процедуры) проекта рекомендуется проверять отдельно при помощи тестов, ориентированных на проверку конкретной процедуры.

Проконтролировать программный код можно еще до ввода в компьютер, то есть за столом, с помощью ***просмотра, проверки*** и ***прокрутки***.

Просмотр текста процедур предусматривает ***обнаружение описок и расхождений с алгоритмом***.

При ***проверке программного кода*** разработчик по тексту мысленно воспроизводит тот вычислительный процесс, который определяет проект, после чего сверяет его с требуемым процессом. На время проверки нужно «забыть», что должна делать каждая процедура, и «узнавать» об этом по ходу её проверки. Только после окончания проверки можно «вспомнить» о том, что она должна делать и сравнить реальные действия каждой процедуры с требуемыми.

Основой ***прокрутки*** является ***имитация выполнения программного проекта***. Для выполнения прокрутки используют простейшие исходные данные и над ними производят все необходимые вычисления, следуя тексту программного кода. Прокрутка — это трудоемкий процесс, поэтому ее следует применять лишь для контроля логически сложных участков.

Следующим этапом контроля правильности программного проекта является его ***тестирование*** на компьютере. Тестирование – это испытание, проверка правильности работы программы в целом либо её составных частей. Отладка и тестирование – это два четко различимых и непохожих друг на друга этапа, поскольку при отладке происходит локализация и устранение синтаксических ошибок, явных ошибок кодирования и «лежащих на поверхности» семантических ошибок, а в процессе тестирования проверяется работоспособность проекта, не содержащего явных ошибок. Таким образом,

тестирование устанавливает факт наличия ошибок, а отладка выясняет ее причину.

Как бы ни был тщательно отлажен программный проект, решающим этапом, устанавливающим его пригодность для работы, является контроль результатов его выполнения на системе заранее разработанных ***тестов***.

Под ***тестом*** понимается некоторая совокупность исходных данных и точное описание результатов, которые должен выработать проект при этих данных. Если для системы тестовых исходных данных проект выдает правильные результаты, то работу проекта условно можно считать правильной, поскольку тестирование может показать лишь наличие ошибок, но не их отсутствие. Нередки случаи, когда новые входные данные вызывают «ошибку» или получение неверных результатов работы проекта, который уже считался полностью отлаженным. Для реализации метода тестов должны быть изготовлены или заранее известны эталонные результаты.

Тестовые данные должны обеспечить проверку всех возможных условий возникновения ошибок. При проведении тестирования руководствуются следующим:

- первый тест должен быть ***максимально прост***, чтобы проверить, работает ли программный проект вообще;
- должна быть испытана ***каждая ветвь*** алгоритма;
- очередной тестовый прогон должен контролировать нечто такое, что еще ***не было проверено*** на предыдущих прогонах;
- арифметические операции в тестах должны ***предельно упрощаться*** для уменьшения объема вычислений;
- в тестовых примерах количество элементов последовательностей, точность для итерационных вычислений, количество проходов цикла должны задаваться из соображений ***сокращения объема вычислений;***
- тестирование должно быть ***целенаправленным и систематизированным***, поскольку при случайном выборе тестовых данных многие ситуации могут оказаться непроверенными;
- усложнение тестовых данных должно происходить ***постепенно***.

Пример. Разработаем систему тестов для задачи нахождения действительных корней квадратного уравнения $ax^2 + bx + c = 0$. При решении этой задачи возможны случаи, приведенные на рисунке П1.2.1.

Номер теста	Проверяемый случай	Коэффициенты			Результаты
		a	b	c	
1	d >0	1	1	-2	x1 = 1, x2 = - 2.
2	d=0	1	2	1	Корни равны: x1 = - 1, x2 = - 1.
3	d < 0	2	1	2	Действительных корней нет
4	a=0, b=0, c=0	0	0	0	Все коэффициенты равны нулю. x — любое число.
5	a=0, b=0, c<>0	0	0	2	Неправильное уравнение.
6	a=0, b<>0	0	2	1	Линейное урав-ние. Один корень:x=- 0,5.
7	a<>0,b<>0,c=0	2	1	0	x1= 0, x2= - 0,5.

Рисунок П1.2.1 – Система тестов для задачи нахождения действительных корней квадратного уравнения

При тестировании, например, важно получить ответ на следующие вопросы:

- что произойдет, если программа не рассчитана на обработку отрицательных и нулевых значений переменных, а в результате какой-либо ошибки придется иметь дело как раз с такими данными?
- что произойдет, если числа будут слишком малыми или слишком большими?

Наихудшая ситуация складывается тогда, когда программа воспринимает неверные данные как правильные и выдает неверный, но правдоподобный результат. Программа должна сама отвергать любые данные, которые она не в состоянии обрабатывать правильно. При этом ошибки могут быть допущены на всех этапах решения задачи – от ее постановки до разработки программного кода.

Приложение 1.3 Титульный лист

МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ
Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский технический университет связи и информатики»

Кафедра «Информатика»

Лабораторная работа №1
«Основные средства и технология разработки консольных программных проектов в интегрированной среде Visual Studio .NET»

по дисциплине
«Введение в информационные технологии»

Выполнил: студент гр. БИБ2501 Иванов И.И.
Вариант №15

Проверил: доц. Шакин В.Н.

Москва, 2025 г.

Приложение 1.4

Требования к оформлению отчета по лабораторным работам

в соответствии с ГОСТ 2.105-95.

1) Текст отчета по лабораторным работам набирается на одной стороне листа белой бумаги формата А4.
2) Отчет должен иметь титульный лист (образец прилагается), содержащий сведения о названии учебного заведения, кафедры, названии лабораторной работы, фамилию и имя студента, номер индивидуального варианта.
3) Заголовки структурных элементов работы располагают в середине строки без точки в конце и печатают заглавными буквами без подчеркивания.
4) Текст должен быть набран шрифтом Times New Roman. Интервал – 1,5 (полуторный); выравнивание по ширине без переносов; абзацный отступ – 1,25 см. При наборе текста использовать кегль (размер шрифта): 14 – для основного текста; 10 – для сносок и примечаний.
5) Размеры полей: правое — не менее 10 мм, верхнее и нижнее — не менее 20 мм, левое — не менее 30 мм.
6) На все рисунки в тексте должны быть даны ссылки (см. рисунок 1.3.1, и т.д.). Рисунки должны располагаться непосредственно после текста, в котором они упоминаются впервые.
7) Рисунки должны быть подписаны. Например, рисунок 1.3.1.

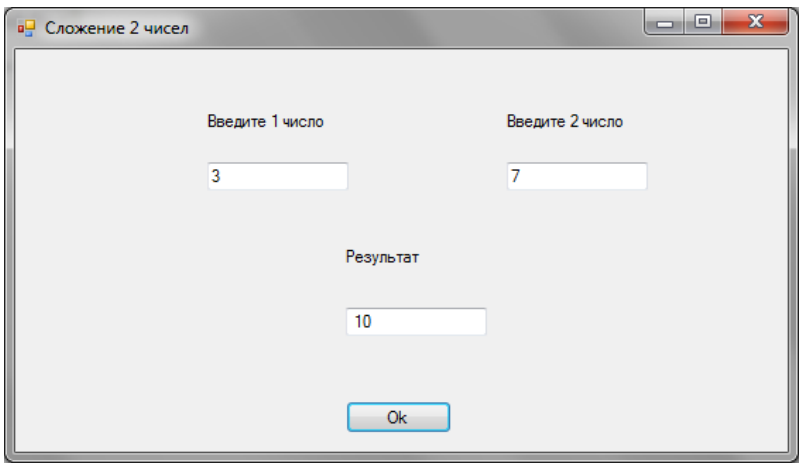


Рисунок 1.3.1 – Результат работы программы

8) На все таблицы в тексте должны быть ссылки (таблица 1.1 и т.д.). Таблица должна располагаться непосредственно после текста, в котором она упоминается впервые.
9) Таблицы должны быть подписаны. Например,

Таблица 1.1 – Проверка результатов

Имя ячейки	**Значение**	**Имя ячейки**	**Значение**
A1	x =	B1	1,4444
A2	t=	B2	0,318

10) Формулы и уравнения следует выделять из текста в отдельную строку. Над и под каждой формулой или уравнением нужно оставить по пустой строке.
11) Формулы должны быть пронумерованы. Например,

$$y = 9x^2 + Sin^2 x\sqrt{a+b} \qquad (1)$$

12) Пример оформления **маркированного** списка:
    - интерфейс программы;
    - программа решения задачи;
    - тестовый вариант исходных данных.

13) Пример оформления **нумерованного** списка:
    1) Название лабораторной работы.
    2) Фамилия, имя студента, номер группы, номер варианта.
    3) Цель лабораторной работы.
    4) Условие задачи.

14) Схемы алгоритмов должны быть выполнены в приложении Microsoft Visio).
15) Все аббревиатуры и сокращения должны быть расшифрованы при первом их употреблении в тексте.
16) Страницы отчета должны быть пронумерованы (номер страницы располагается внизу и справа). Титульный лист не нумеруется.
17) Программные коды проектов копируются в MS Word или включаются в виде рисунков со скриншотами текста программы.

Примечание.

Более подробная информация по оформлению работ содержится в ГОСТ 7.32-2001 «Отчет о научно-исследовательской работе. Структура и правила оформления».