

Документ подписан простой электронной подписью
Информация о владельце:
ФИО: Локтионова Оксана Геннадьевна
Должность: проректор по учебной работе
Дата подписания: 04.05.2022 14:55:03
Уникальный программный ключ:
0b817ca911e6668abb13a5d426d39e5f1c11eabb73e43a744e31fda660b9

МИНОБРАЗОВАНИЯ РОССИИ

Федеральное государственное бюджетное образовательное
учреждение высшего образования

«Юго-Западный государственный университет»

(ЮЗГУ)

Кафедра вычислительной техники

УТВЕРЖДАЮ
Проректор по учебной работе
О.Г. Локтионова

«15» 01

2021г.



ТЕСТИРОВАНИЕ, ОТЛАДКА И ОПТИМИЗАЦИЯ ПРОГРАММ

Методические рекомендации по выполнению лабораторных работ
для студентов направления 09.04.01

Курск 2021

УДК 004.652

Составители: И.Е. Чернецкая, С.Ю. Мирошниченко

Рецензент

Кандидат технических наук, доцент *Ю.А. Халин*

Тестирование, отладка и оптимизация программ: методические рекомендации по выполнению лабораторных и практических работ / Юго-Зап. гос. ун-т; сост.: И.Е.Чернецкая, С.Ю. Мирошниченко. – Курск, 2021. – 37 с.: Библиогр.: с. 37.

Описывается настройка среды виртуальной машины со средой разработки, установка и использование системы контроля версий, сборка свободно распространяемых библиотек, поиск в программах утечек оперативной памяти, а также поиск и устранение проблем производительности.

Методические рекомендации соответствуют рабочей программе дисциплины «Современные проблемы информатики и вычислительной техники».

Предназначены для студентов направления 09.04.01 Информатика и вычислительная техника.

Текст печатается в авторской редакции

Подписано в печать *15.01.21* Формат 60*84 1/16.
Усл. печ. л. 2,15. Уч.-изд. л. 1,95. Тираж 50 экз. Заказ *214* Бесплатно.
Юго-Западный государственный университет.
305040, г. Курск, ул. 50 лет Октября, 94.

Содержание

1 Общие положения.....	4
2 Структура цикла лабораторных работ	5
3 Защита лабораторных работ	7
4 Методические указания по выполнению лабораторных работ	8
4.1 Лабораторная работа №1 «Установка и настройка виртуальной машины и среды разработки»	10
4.2 Лабораторная работа № 2 «Изучение распределенной системы управления версиями GIT»	17
4.3 Лабораторная работа № 3 «Сборка библиотеки Boost»	23
4.4 Лабораторная работа № 4 «Сборка библиотеки QT»	28
4.5 Лабораторная работа № 5 «Поиск утечек памяти»	32
4.6 Лабораторная работа № 6 «Профилирование и оптимизация»	34
Библиографический список	37

1 Общие положения

Цикл лабораторных работ по дисциплине «Современные проблемы информатики и вычислительной техники» выполняется студентами направления подготовки по программе магистратуры 09.04.01 Информатика и вычислительная техника, направленность «Элементы и устройства вычислительной техники и информационных систем» с целью закрепления и углубления полученных теоретических знаний, а также приобретения навыков использования методов отладки и оптимизации производительности программного кода. В процессе выполнения лабораторных работ происходит развитие навыков самостоятельной учебной, и исследовательской работы студентов.

Каждому студенту выдается индивидуальное задание на цикл лабораторных работ в соответствии с номером варианта.

2 Структура цикла лабораторных работ

Цикл лабораторных работ состоит из следующих этапов:

1. Получение варианта задания в соответствии с индивидуальным номером.
2. Лабораторная работа №1 «Установка и настройка виртуальной машины и среды разработки».
3. Лабораторная работа №2 «Изучение распределенной системы управления версиями GIT».
4. Лабораторная работа №3 «Сборка библиотеки Boost».
5. Лабораторная работа №4 «Сборка библиотеки QT».
6. Лабораторная работа №5 «Поиск утечек памяти».
7. Лабораторная работа №6 «Профилирование и оптимизация».

Лабораторная работа № 1 описывает установку и настройку виртуальной машины и среды разработки. Содержит описание последовательности действий по установке и настройке используемых операционной системы и среды разработки.

Лабораторная работа № 2 посвящена изучению распределенной системы управления версиями GIT. Работа содержит приемы использования основного набора команд.

Лабораторная работа № 3. Содержит описание процесса сборки библиотеки Boost и интеграции в среду разработки Visual Studio, а также тестовое задание, выполненное с использованием функций библиотеки Boost.

Тема лабораторной работы № 4: сборка библиотеки QT. Содержит описание процесса сборки и интеграции в среду разработки Visual Studio, а также тестовое задание, выполненное с использованием функций библиотеки QT.

Лабораторная работа № 5 посвящена поиску утечек памяти. Содержит описание процесса поиска и устранения утечек памяти в приложении, выполненном согласно варианту задания. Кроме того, отчет должен содержать результаты тестирования, выявления ошибок и действий по их устранению.

Итоговая лабораторная работа № 6 посвящена профилированию и оптимизации производительности программного кода. Содержит описание процесса сбора характеристик работы программы, разработанной согласно варианту задания, и действий по ее оптимизации. Кроме того, отчет должен содержать результаты тестирования, выявления ошибок и действий по их устранению.

3 Защита лабораторных работ

Каждая лабораторная работа должна быть предоставлена в одном экземпляре и считается выполненной после исправления недостатков и ошибок, обнаруженных при проверке преподавателем. Итоговая программная реализация обязательно прилагается на диске и демонстрируется преподавателю.

Защиты лабораторных работ проводятся в сроки, определенные рабочей программой по дисциплине, и являются обязательной формой проверки выполненной работы.

Основными критериями при выставлении оценки является:

1. Самостоятельность выполненной работы.
2. Теоретическая и практическая подготовка студента.
3. Навыки исследовательского характера.
4. Грамотность изложения и свободное владение материалом.
5. Качество оформления лабораторной работы.
6. Правильность ответов на вопросы при защите.

Порядок защиты лабораторных работ является следующим:

1. Доклад студента о результатах работы (2-5 мин).
2. Ответы на вопросы.

Студент, не предоставивший в установленный срок все лабораторные работы или не защитивший их по неуважительной причине, не допускается до промежуточной аттестации.

4 Методические указания по выполнению лабораторных работ

Этапы выполнения работы.

1. Получить вариант задания в соответствии с индивидуальным номером.

2. Выполнить лабораторную работу №1: установить ПО виртуализации, создать и настроить виртуальную машину, установить в ней операционную систему и среду разработки.

3. Выполнить лабораторную работу №2: установить распределенную систему управления версиями GIT, изучить основные команды.

4. Выполнить лабораторную работу №3: собрать и интегрировать в среду разработки Visual Studio библиотеку Boost, разработать приложение, согласно варианту с использованием функций библиотеки.

5. Выполнить лабораторную работу №4: собрать и интегрировать в среду разработки Visual Studio библиотеку QT, разработать приложение, согласно варианту с использованием функций библиотеки.

6. Выполнить лабораторную работу №5: провести поиск утечек памяти и описать процесс их устранения в программе, согласно варианту задания.

7. Выполнить лабораторную работу №6: провести сбор характеристик работы программы, разработанной согласно

варианту задания, и осуществить оптимизацию, тестирование, отладку и устранение ошибок.

Базовые требования к программам, создаваемым и дорабатываемым в результате выполнения лабораторных работ.

Функциональные требования:

1. выполнение условий, заданных индивидуальным вариантом;
2. возможность сбора статистики тестируемого приложения;
3. отображение текущего состояния и результатов тестирования в консольном или графическом режиме с использованием стандартных средств Windows.

Требования к условиям эксплуатации:

1. IBM-совместимая ПЭВМ.
2. Операционная система MS Windows XP или выше.
3. Основной язык программирования C++ (допустимо выполнение работы на C# или Java, однако, необходимо обоснование выбора в пользу данных языков).
4. Все сообщения должны выводиться на русском языке.
5. Рекомендуемая среда программирования Visual Studio 2010 или выше.

4.1 Лабораторная работа №1 «Установка и настройка виртуальной машины и среды разработки»

Задание.

Установить и настроить виртуальную машину, используя программный продукт виртуализации для операционных систем Oracle VM VirtualBox. Установить и настроить среду разработки Microsoft Visual Studio Express, рекомендуемая версия Microsoft Visual Studio Express 2015.

Установка и настройка виртуальной машины.

1. Скачайте последнюю версию программного продукта виртуализации для операционных систем Oracle VM VirtualBox с официального сайта <https://www.virtualbox.org/>

2. Установите Oracle VM VirtualBox согласно инструкциям установщика.

3. Создайте новую виртуальную машину.

4. Задайте имя виртуальной машины вида «Название операционной системы» «разрядность системы» «фамилия студента, выполняющего работу». Например, «win7x64_Ivanov». Выберите тип системы и версию. Укажите необходимый объем оперативной памяти. Пример настроек создания новой виртуальной машины смотрите на рисунке 4.1.

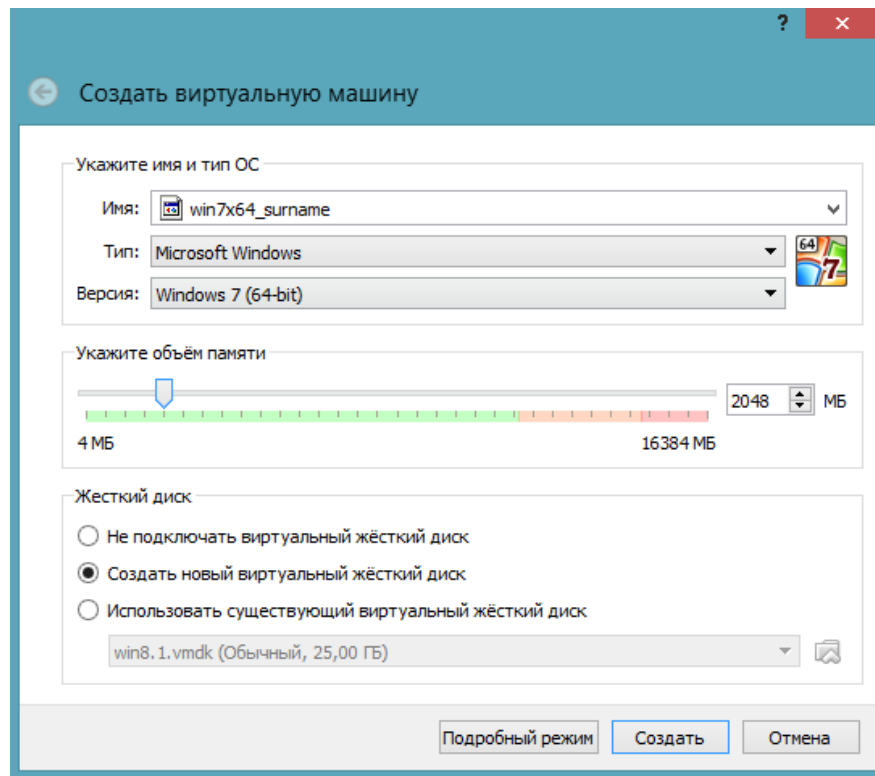


Рисунок 4.1 – Параметры создания новой виртуальной машины

Задайте создание нового виртуального жесткого диска, смотрите рисунок 4.2.

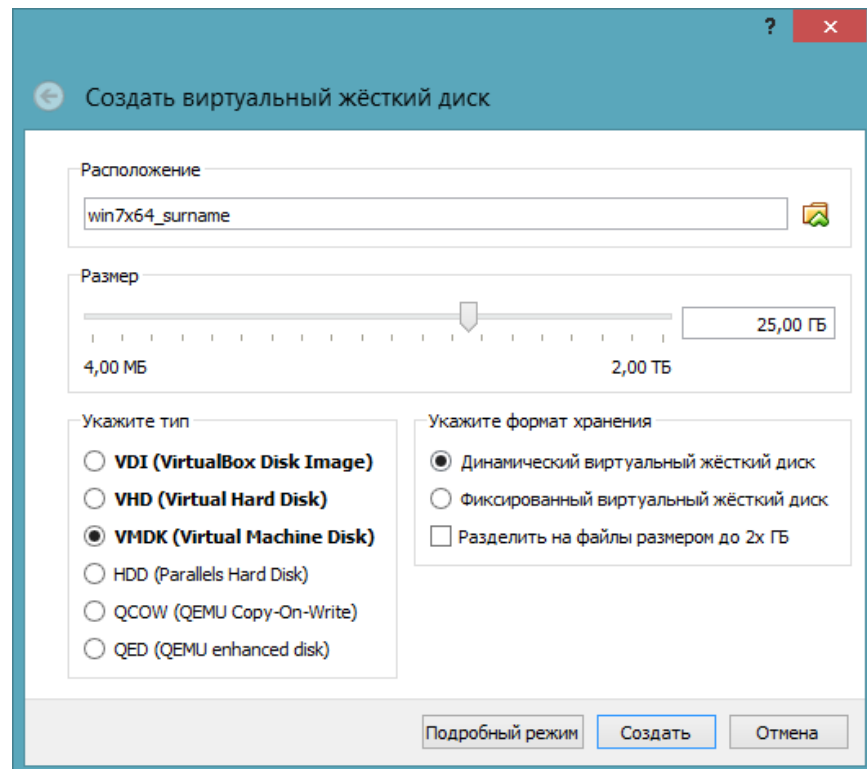


Рисунок 4.2 – Параметры создания нового виртуального жесткого диска

5. Запустите созданную виртуальную машину. В появившемся окне выберите загрузочный диск операционной системы, смотрите рисунок 4.3.

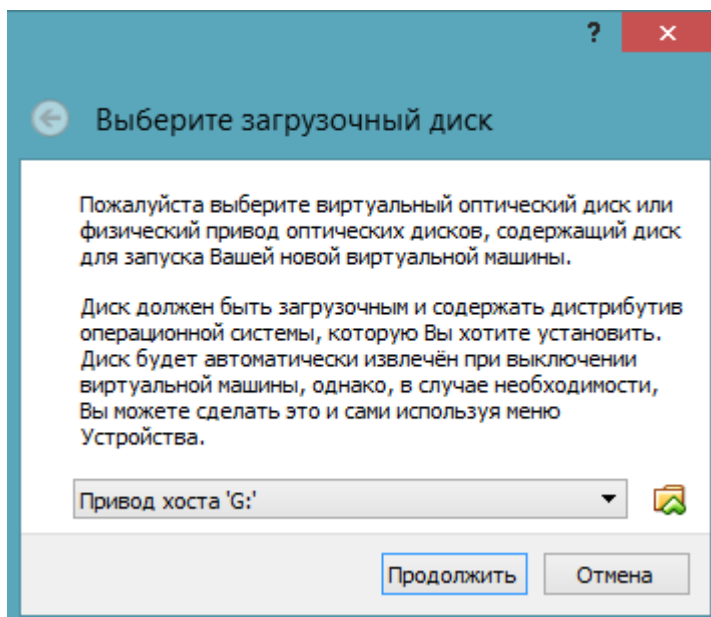


Рисунок 4.3 – Выбор загрузочного диска

6. Установите операционную систему согласно инструкциям установщика.

7. Настройте общий буфер обмена и функцию «Drag and Drop». Для этого в панели инструментов выберите «Устройства», в открывшемся списке включите данные функции – укажите параметр настройки «Двунаправленный». Далее в этом же списке выберите «Подключить образ диска Дополнительной гостевой ОС...», смотрите рисунок 4.4.



Рисунок 4.4 – Подключение образа диска дополнительной гостевой ОС

Установите «Oracle VM VirtualBox Guest Additions» согласно инструкциям установщика. Перезагрузите виртуальную машину.

8. Проверьте возможность копирования файлов из основной операционной системы в виртуальную.

9. Настройте параметры виртуальной машины. Выключите виртуальную машину. Из списка имеющихся виртуальных машин выберите настраиваемую, на панели инструментов главного окна VirtualBox нажмите на кнопку «Настроить». В открывшемся окне в левой вкладке выберите «Системы», измените объем основной памяти, во вкладке «Дисплей» измените объем видеопамати, смотрите рисунок 4.5.

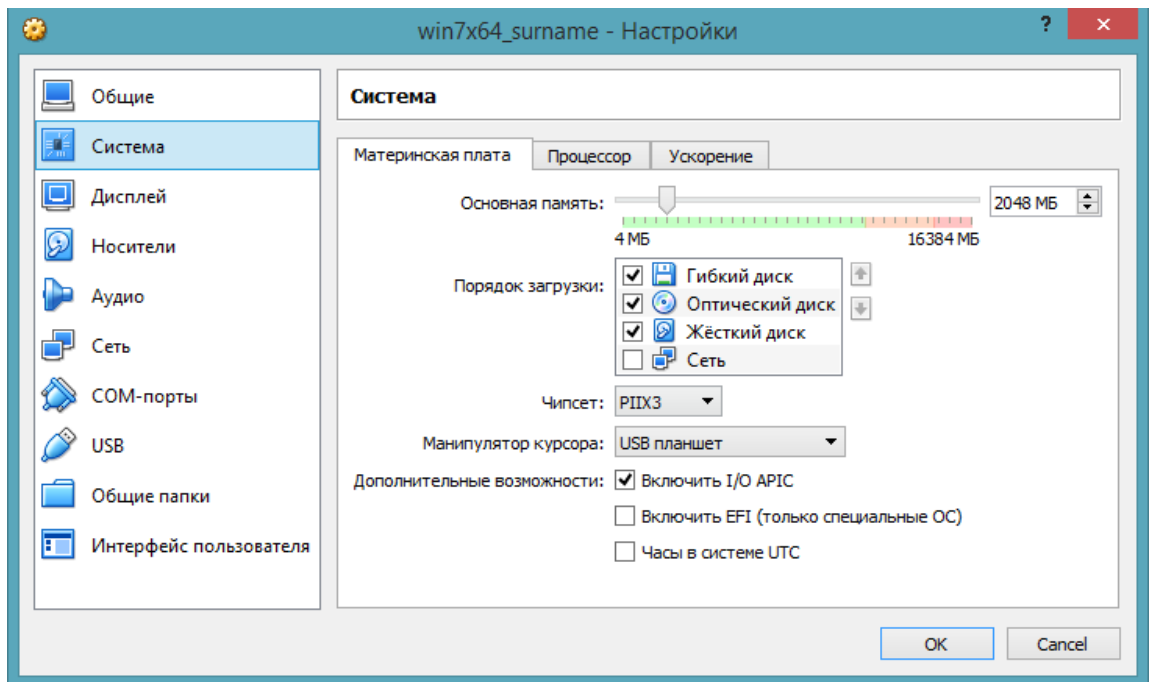


Рисунок 4.5 – Изменение параметров виртуальной машины

Запустите виртуальную машину, убедитесь в том, что настройки виртуальной машины были применены.

10. Настройка общей папки. Выключите виртуальную машину. Из списка имеющихся виртуальных машин выберите настраиваемую, на панели инструментов главного окна VirtualBox нажмите на кнопку «Настроить». В открывшемся окне в левой вкладке выберите «Общие папки», добавьте общую папку, смотрите рисунок 4.6.

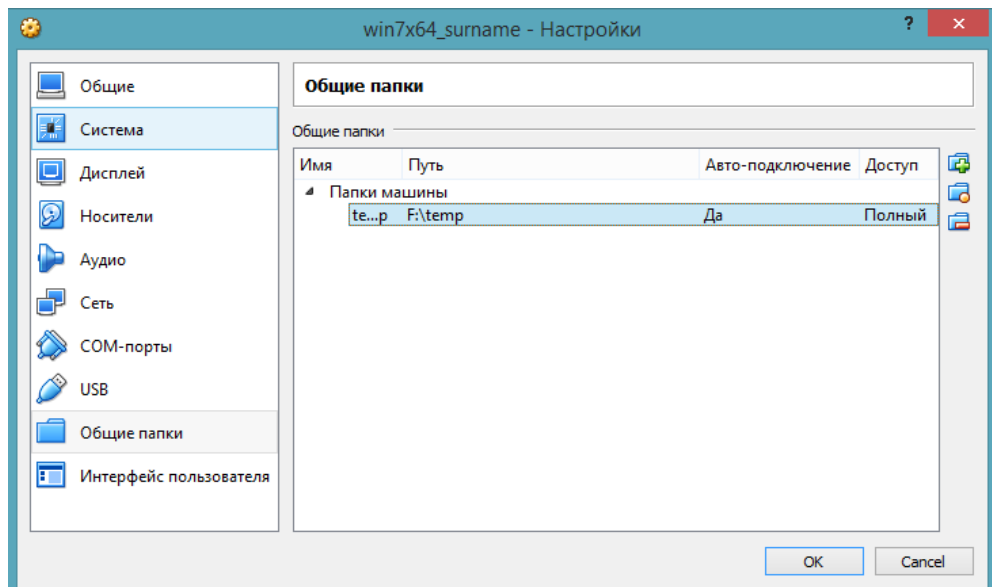


Рисунок 4.6 – Настройка общей папки

Запустите виртуальную машину, убедитесь в том, что виртуальная машина имеет доступ к общей папке.

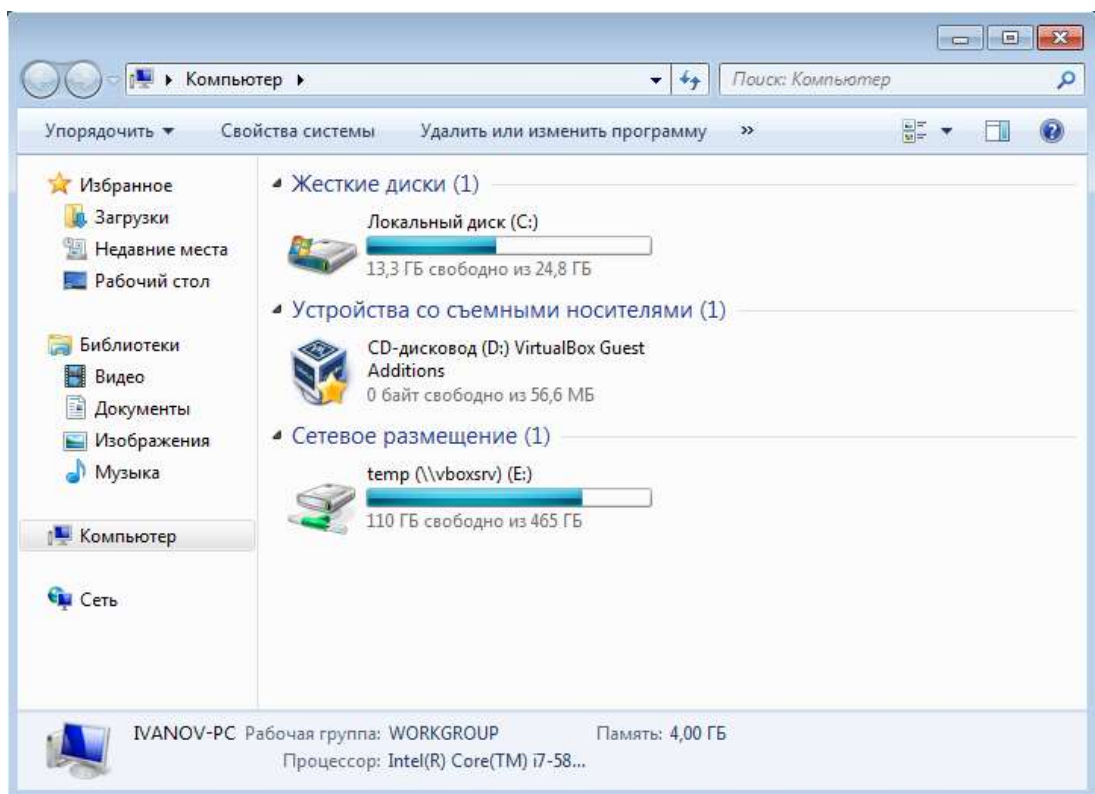


Рисунок 4.7 – Доступ к общей папке

Установка и настройка среды разработки Microsoft Visual Studio Express

1. Скачайте Microsoft Visual Studio Express с официального сайта <https://www.visualstudio.com/>

2. Установите Microsoft Visual Studio Express согласно инструкциям установщика.

4.2 Лабораторная работа № 2 «Изучение распределенной системы управления версиями GIT»

Задание.

Ознакомится с основными принципами работы распределенной системы контроля версий на примере Git-клиента.

Работа с Git-клиентом.

1. В первую очередь необходимо установить Git-клиент. Скачать установочный файл можно с официального сайта *git-scm.com*

После загрузки и установки в контекстном меню Windows появятся новые пункты: Git GUI Here, Git Bash Here, смотрите рисунок 4.8.

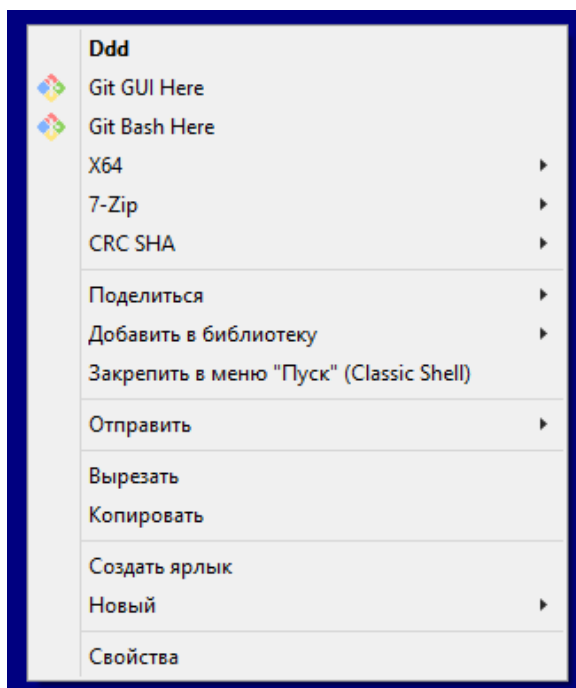


Рисунок 4.8 – Пункты Git GUI Here, Git Bash Here в контекстном меню

2. В папке с проектом, разработанным согласно полученному варианту, выполните команду “Git Bash Here”. Для того, чтобы

создать из текущего каталога git репозиторий выполните в консоли команду

```
git init
```

3. В результате выполнения данной команды вы увидите на экране следующую строку:

```
Initialized empty Git repository in CURRENT_PATH/.git/
```

где CURRENT_PATH – это путь к папке с проектом.

4. Добавьте файлы проекта в созданный на прошлом шаге репозиторий. Для того, чтобы добавить все файлы, выполним команду

```
git add .
```

Если необходимо добавить какой-то конкретный файл, следует выполнить команду

```
git add file_name.cpp
```

где file_name.cpp – это имя добавляемого файла.

В случае успешного выполнения команды не будет выведено никаких сообщений.

5. Выполните команду

```
git commit -m "First Commit"
```

В результате вы получите сообщение вида:

```
$ git add hello.cpp
```

```
$ git commit -m "First Commit"
```

```
[master (root-commit) 911e8c9] First Commit
```

```
1 files changed, 1 insertions(+), 0 deletions(-)
```

```
create mode 100644 file_name.cpp
```

6. Для проверки состояния репозитория используйте команду

```
git status
```

В результате выполнения которой на экран будет выведено сообщение вида:

```
On branch master
```

```
nothing to commit, working tree clean
```

Это означает, что на данный момент в репозитории хранится текущее состояние рабочего каталога и нет никаких изменений, ожидающих записи.

7. Если на данном этапе внести какие-либо изменения в один из файлов проекта, например, в `file_name.cpp`, и после этого выполнить команду `git status`, то результат будет иным:

```
$ git status
```

```
On branch master
```

```
Changes not staged for commit:
```

```
(use "git add <file>..." to update what will be committed)
```

```
(use "git checkout -- <file>..." to discard changes in working directory)
```

```
modified: file_name.cpp
```

```
no changes added to commit (use "git add" and/or "git commit -a")
```

Первое, что нужно заметить, это то, что `git` знает, что файл `file_name.cpp` был изменен, но при этом эти изменения еще не зафиксированы в репозитории.

Также обратите внимание на то, что сообщение о состоянии дает вам подсказку о том, что нужно делать дальше. Если вы хотите добавить эти изменения в репозиторий, используйте команду `git`

`add`. В противном случае используйте команду `git checkout` для отмены изменений.

8. Выполните команду `git add .` или `git add file_name.cpp` для того чтобы проиндексировать изменения во всех файлах проекта или только в одном.

Теперь, после выполнения команды `git status` вы получите сообщение вида:

```
$ git status
```

```
On branch master
```

```
Changes to be committed:
```

```
(use "git reset HEAD <file>..." to unstage)
```

```
    modified:   file_name.cpp
```

Изменения в файле `file_name.cpp` были проиндексированы, это означает, что `git` теперь знает об изменении, однако, изменение пока еще не занесено в репозиторий навсегда. Если вы решили, что не хотите подтверждать изменения, команда состояния напомнит Вам о том, что с помощью команды `git reset` можно снять индексацию этих изменений.

9. Отдельный шаг индексации в `git` позволяет вам продолжать вносить изменения в рабочий каталог, а затем, в момент, когда вы захотите взаимодействовать с версионным контролем, `git` позволит записать изменения в малых коммитах, которые фиксируют то, что вы сделали.

Предположим, что Вы отредактировали три файла (`a.cpp`, `b.cpp`, и `c.cpp`). Теперь вы хотите подтвердить все изменения, при этом чтобы изменения в `a.cpp` и `b.cpp` были одним коммитом, в

то время как изменения в `c.cpp` логически не связаны с первыми двумя файлами и должны идти отдельным коммитом.

В теории, вы можете сделать следующее:

```
git add a.cpp
```

```
git add b.cpp
```

```
git commit -m "Changes for a and b"
```

```
git add c.cpp
```

```
git commit -m "Unrelated change to c"
```

Разделяя индексацию и коммит, вы имеете возможность с легкостью настроить, что идет в какой коммит.

Внесем некоторые изменения в произвольный файл из нашей директории и сохраним его. После чего выполним команды

```
git add .
```

```
git commit -m "second commit"
```

Допустим, требуется откатиться к изменениям, выполненным до последнего коммита. Выполните команду

```
git log
```

в результате будет выведено сообщение вида

```
$ git log
```

```
commit 970e48a40f76f8acc02a93612c2c2a5022cb6eb1
```

```
Author: mosin <22mosin@gmail.com>
```

```
Date: Wed Jan 25 15:47:52 2017 +0400
```

```
Second Commit
```

```
commit aca2d2215a86ce924f740834cb526d8c38377265
```

```
Author: mosin <22mosin@gmail.com>
```

```
Date: Wed Jan 25 15:28:34 2017 +0400
```

First Commit

Здесь строка вида

```
commit 970e48a40f76f8acc02a93612c2c2a5022cb6eb1
```

отображает хэш коммита, его уникальный идентификатор. Для того, чтобы перейти к предыдущему состоянию проекта, воспользуемся командой `git checkout aca2d2215`, где `aca2d2215` – это укороченный хэш последнего коммита в результате будет выведено сообщение вида

```
$ git checkout aca2d2215
```

```
Note: checking out 'aca2d2215'.
```

```
You are in 'detached HEAD' state. You can look around, make experimental
```

```
changes and commit them, and you can discard any commits you make in this
```

```
state without impacting any branches by performing another checkout.
```

```
If you want to create a new branch to retain commits you create, you may
```

```
do so (now or later) by using -b with the checkout command again.
```

Example:

```
git checkout -b <new-branch-name>
```

```
HEAD is now at aca2d22... frst
```

Теперь проект вернулся к состоянию первого коммита.

4.3 Лабораторная работа № 3 «Сборка библиотеки Boost»

Задание.

Собрать библиотеку Boost согласно варианту индивидуального задания.

Сборка библиотеки Boost

1. С сайта **Boost C++ Libraries** (www.boost.org), смотрите рисунки 4.9, 4.10, скачайте версию библиотеки Boost, указанную в индивидуальном варианте.

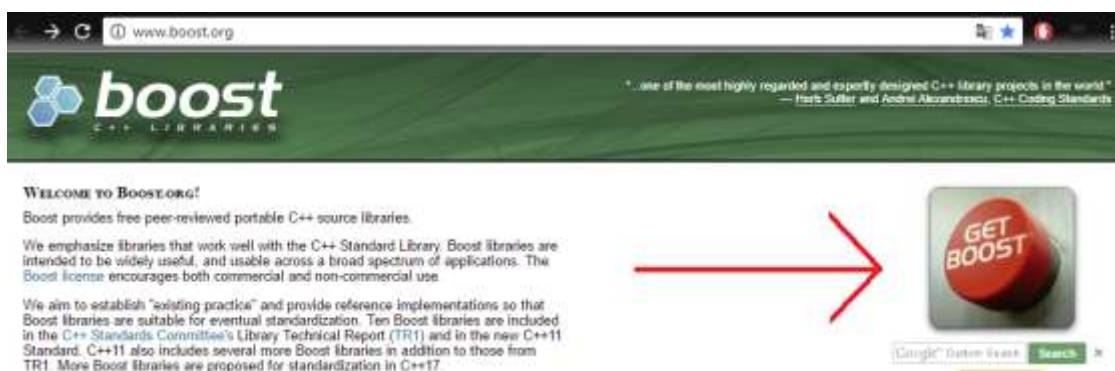


Рисунок 4.9 – Кнопка для скачивания Boost

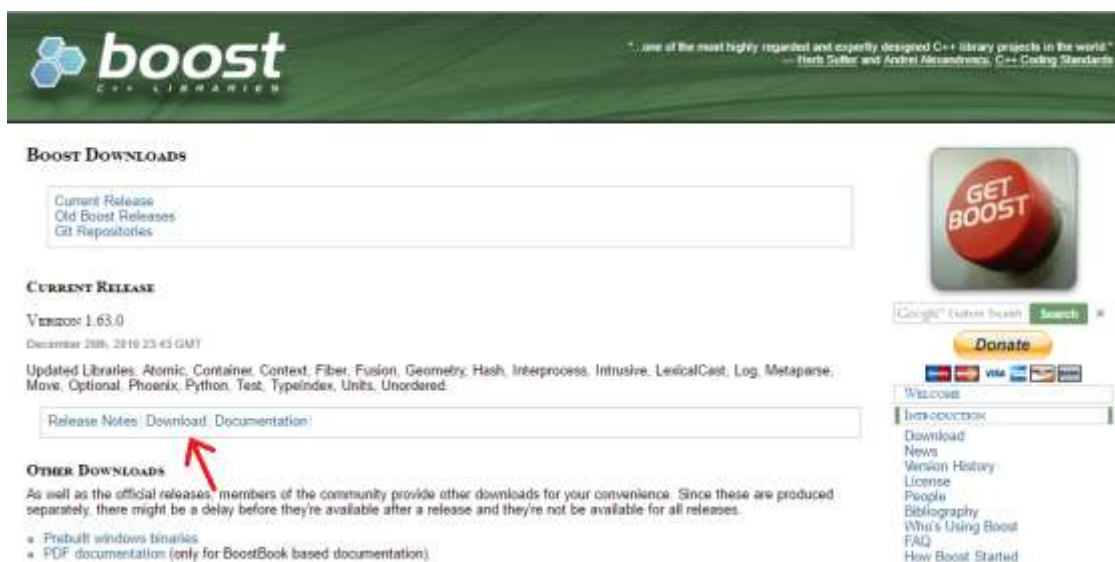


Рисунок 4.10 – Ссылка для скачивания Boost



Boost C++ Libraries

Free peer-reviewed portable C++ source libraries

Brought to you by: beman_dawes, danieljames, eric_niebler, grafik, and 2 others

Summary Files Reviews Support Wiki Mailing Lists News Discussion Code

Looking for the latest version? [Download boost_1_63_0.7z \(71.5 MB\)](#)

Home / boost / 1.63.0

Name	Modified	Size	Downloads / Week
↑ Parent folder			
boost_1_63_0.zip	2016-12-26	142.7 MB	3,593
boost_1_63_0.tar.gz	2016-12-26	97.5 MB	3,717

Рисунок 4.11 – Выбор соседней директории с имеющимися версиями Boost

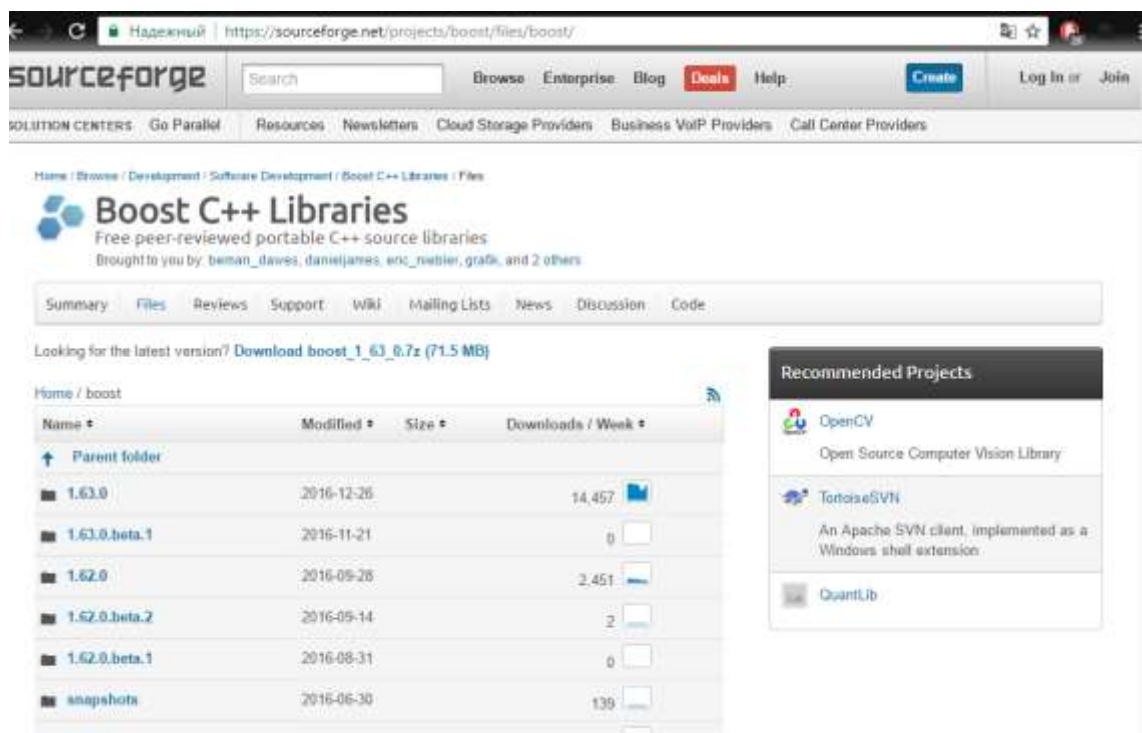


Рисунок 4.12 – Выбор версии Boost согласно варианту задания

2. Распакуйте содержимое архива в подходящую директорию.



Рисунок 4.13 – Распаковка архива с исходниками библиотеки Boost

3. Для сборки необходимо воспользоваться средствами из комплекта программ, поставляемых вместе со средой разработки Microsoft Visual Studio, для этого откройте: Пуск → Все Программы → Visual Studio 2013 → Visual Studio Tools → Developer Command Prompt for VS2013.

Примечание:

В данном примере используется программный продукт Developer Command Prompt из комплекта Visual Studio 2013, в зависимости от версии среды разработки название и путь могут незначительно различаться, например, для 2010 редакции консоль разработчика будет называться Visual Studio x64 Win64Command Prompt (2010).

4. Находясь в консоли разработчика необходимо перейдите в директорию, в которую была распакована библиотека Boost, рисунок 4.14.

```

D:\Libraries\boost>cd boost_1_63_0
D:\Libraries\boost\boost_1_63_0>bootstrap.bat
Building Boost.Build engine

Bootstrapping is done. To build, run:

    .\b2

To adjust configuration, edit 'project-config.jam'.
Further information:

- Command line help:
  .\b2 --help

- Getting started guide:
  http://boost.org/more/getting_started/windows.html

- Boost.Build documentation:
  http://www.boost.org/build/doc/html/index.html

D:\Libraries\boost\boost_1_63_0>

```

Рисунок 4.14 – Переход в директорию с исходниками библиотеки Boost

5. Запускаем bootstrap.bat, начнется построение exe файла bjam.exe. bjam- система построения boost'a. Он предназначен для построения C++ проектов.

6. После завершения процесса сборки bjam.exe необходимо собрать собственно boost. Для этого в консоли нужно выполнить команду: bjam toolset=msvc-12.0 architecture=x86 address-model=64 variant=debug,release threading=multi link=static

Примечание:

toolset=msvc-12.0- параметр, указывающий версию компилятора, зависит от версии Microsoft Visual Studio.

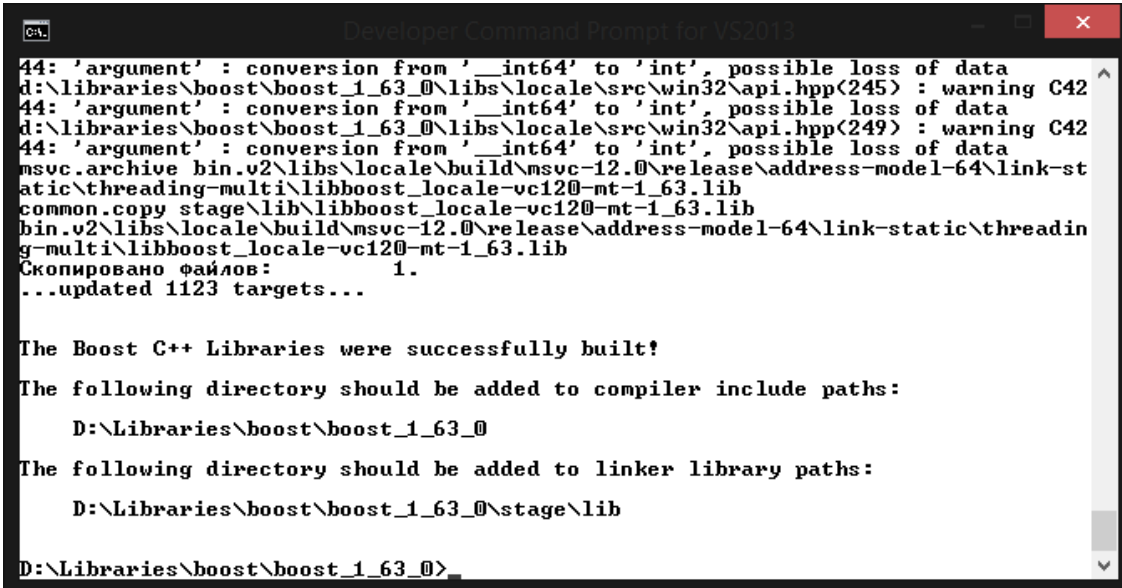
Версия среды разработки:	Версия компилятора:
2010	10
2012	11
2013	12

architecture=x86- параметр, определяющий архитектуру целевой платформы;

address-model=64- параметр, определяющий разрядность целевой платформы;

variant=debug,release- параметр, определяющий варианты сборки библиотек, при работе с многими библиотеками стоит уделить внимание тому, что для сборки отладочных (Debug) и распространяемых (Release) версий требуются различные версии библиотек.

7. В случае успешной сборки будет выведено соответствующее сообщение, смотрите рисунок 4.15.



```
44: 'argument' : conversion from '__int64' to 'int', possible loss of data
d:\libraries\boost\boost_1_63_0\libs\locale\src\win32\api.hpp(245) : warning C42
44: 'argument' : conversion from '__int64' to 'int', possible loss of data
d:\libraries\boost\boost_1_63_0\libs\locale\src\win32\api.hpp(249) : warning C42
44: 'argument' : conversion from '__int64' to 'int', possible loss of data
msvc.archive bin.v2\libs\locale\build\msvc-12.0\release\address-model-64\link-st
atic\threading-multi\libboost_locale-vc120-mt-1_63.lib
common.copy stage\lib\libboost_locale-vc120-mt-1_63.lib
bin.v2\libs\locale\build\msvc-12.0\release\address-model-64\link-static\threadin
g-multi\libboost_locale-vc120-mt-1_63.lib
Скопировано файлов: 1.
...updated 1123 targets...

The Boost C++ Libraries were successfully built!
The following directory should be added to compiler include paths:
    D:\Libraries\boost\boost_1_63_0
The following directory should be added to linker library paths:
    D:\Libraries\boost\boost_1_63_0\stage\lib
D:\Libraries\boost\boost_1_63_0>
```

Рисунок 4.15 – Сообщение об успешной сборке

8. Далее необходимо выполнить команду «bjam install». Данная команда позволяет проинсталлировать по указанному ранее пути скомпилированные файлы.

4.4 Лабораторная работа № 4 «Сборка библиотеки QT»

Задание.

Собрать библиотеку Qt (рекомендуемая версия QT-5.5) и драйвер PostgreSQL (рекомендуемая версия PostgreSQL 9.5.5) согласно варианту индивидуального задания.

Сборка библиотеки QT.

1. Скачайте исходники библиотеки QT с официального сайта (ссылка для скачивания исходников QT-5.5 <https://download.qt.io/archive/qt/5.5/5.5.0/single/qt-everywhere-opensource-src-5.5.0.zip>) и распакуйте архив.

2. Переименуйте распакованную папку и поместите её по желаемому пути. Необходимо учесть, что при компиляции имена папок будут вкомпилированы в qmake и поменять их будет нельзя.

3. Для сборки рекомендуется использовать средства из комплекта программ поставляемых вместе со средой разработки Microsoft Visual Studio, для этого открываем: *Пуск → Все Программы → Visual Studio 2012 → Visual Studio Tools → VS2012 x64 Cross Tools Command Prompt.*

Примечание:

В данном примере используется программный продукт *Developer Command Prompt* из комплекта *Visual Studio 2012*, в зависимости от версии среды разработки название и путь могут незначительно различаться.

4. Запустите «VS Command Prompt» от имени администратора, смотрите рисунок 4.16.

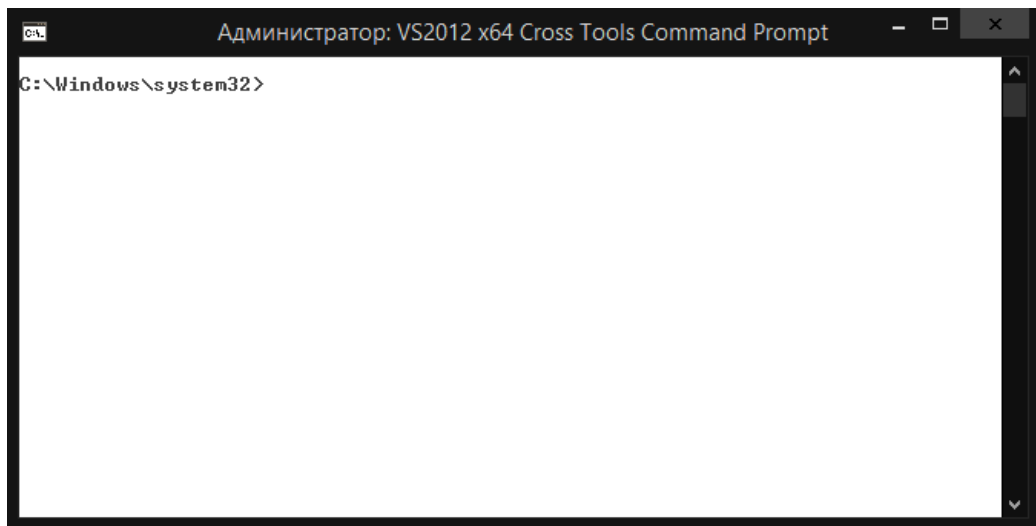


Рисунок 4.16 – VS Command Prompt

5. Находясь в консоли разработчика перейдите в папку с загруженными исходниками QT, используя команду вида «cd directory_path», смотрите рисунок 4.17.

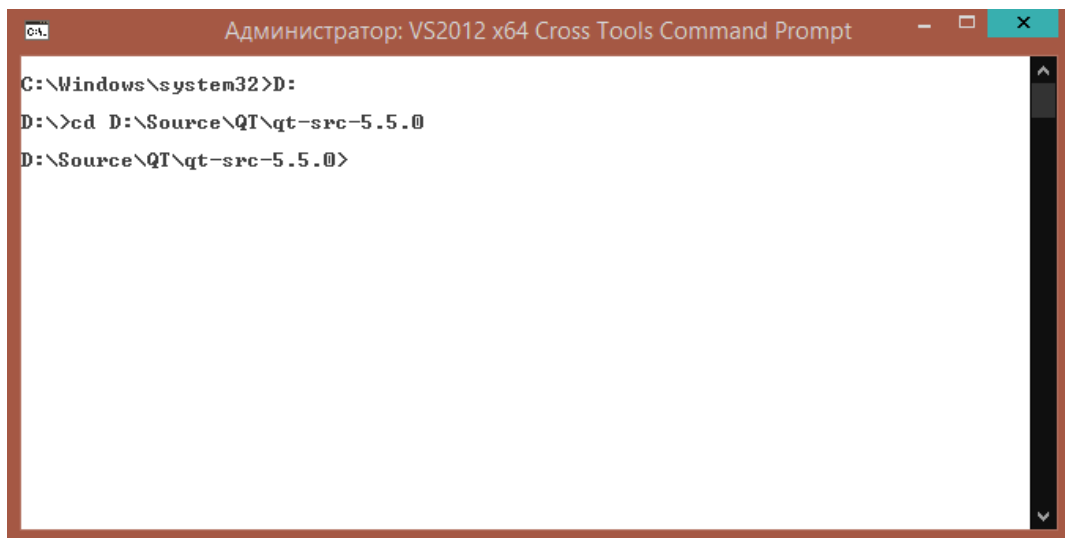
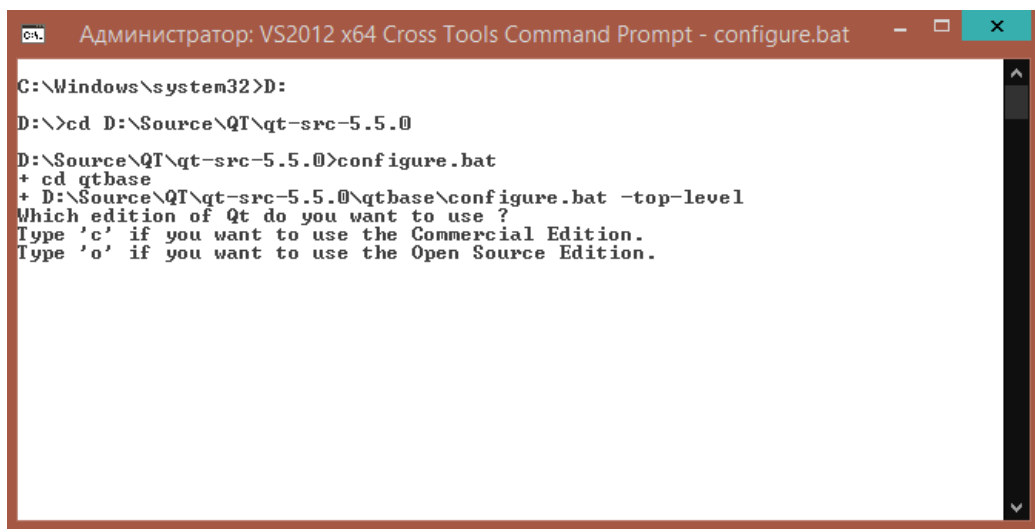


Рисунок 4.17 – Переход к директории исходников QT

6. Запустите «configure.bat», позволяющий создать makefile для сборки.

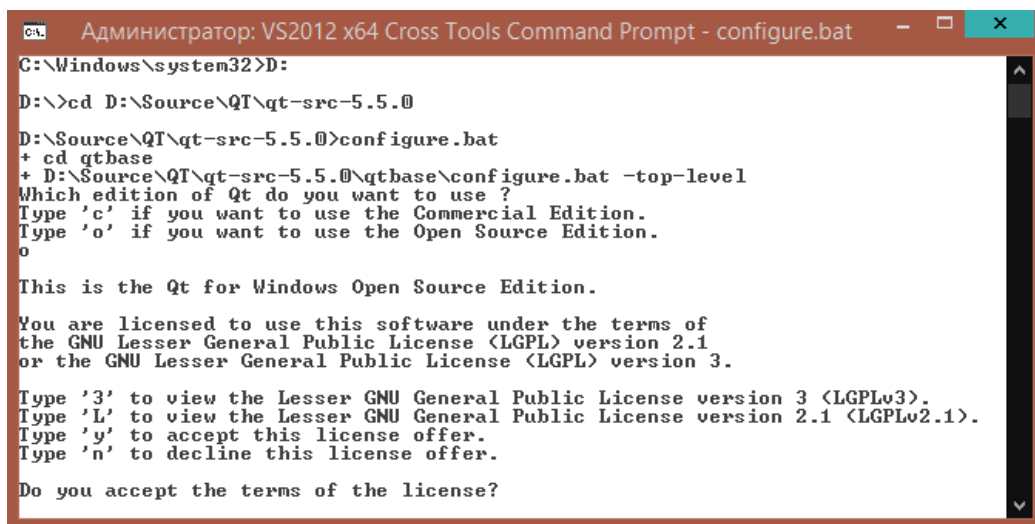
7. Выберите тип лицензии, рекомендуется выбрать «open source» (бесплатную лицензию), для этого введите «o», смотрите рисунок 4.18.



```
C:\Windows\system32>D:
D:\>cd D:\Source\QT\qt-src-5.5.0
D:\Source\QT\qt-src-5.5.0>configure.bat
+ cd qtbase
+ D:\Source\QT\qt-src-5.5.0\qtbase\configure.bat -top-level
Which edition of Qt do you want to use ?
Type 'c' if you want to use the Commercial Edition.
Type 'o' if you want to use the Open Source Edition.
```

Рисунок 4.18 – Выбор типа лицензии

8. Подтвердите соглашение с лицензией, для этого введите «y», смотрите рисунок 4.19.



```
C:\Windows\system32>D:
D:\>cd D:\Source\QT\qt-src-5.5.0
D:\Source\QT\qt-src-5.5.0>configure.bat
+ cd qtbase
+ D:\Source\QT\qt-src-5.5.0\qtbase\configure.bat -top-level
Which edition of Qt do you want to use ?
Type 'c' if you want to use the Commercial Edition.
Type 'o' if you want to use the Open Source Edition.
o

This is the Qt for Windows Open Source Edition.

You are licensed to use this software under the terms of
the GNU Lesser General Public License (LGPL) version 2.1
or the GNU Lesser General Public License (LGPL) version 3.

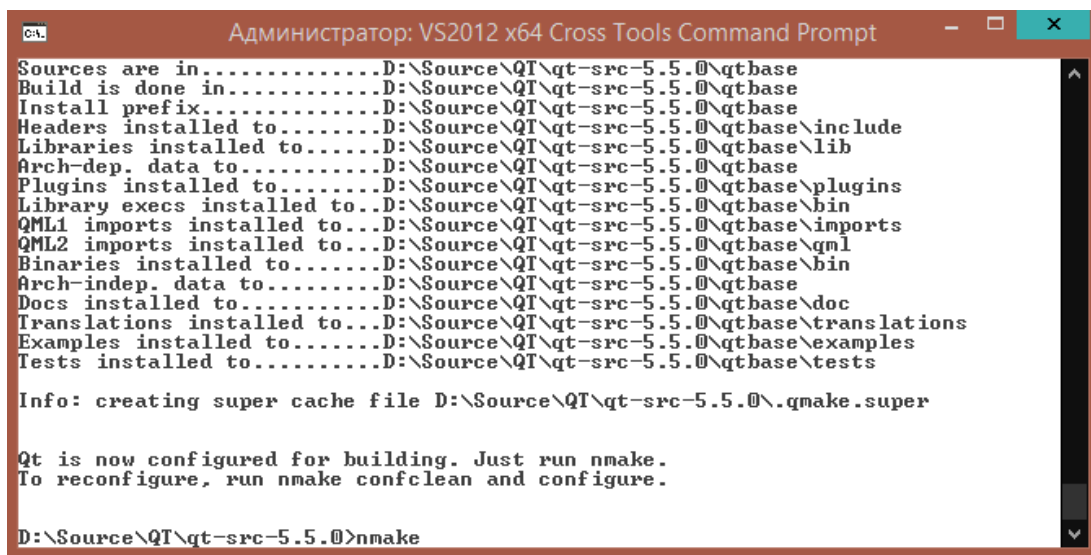
Type '3' to view the Lesser GNU General Public License version 3 (LGPLv3).
Type 'L' to view the Lesser GNU General Public License version 2.1 (LGPLv2.1).
Type 'y' to accept this license offer.
Type 'n' to decline this license offer.

Do you accept the terms of the license?
```

Рисунок 4.19 – Соглашение с лицензией

9. После того, как makefile будет создан, вызовите nmake. Вызов nmake – это вызов утилиты от Microsoft для чтения ранее

сформированного makefile и компиляции. Введите «nmake»,
рисунок 4.20



```
Администратор: VS2012 x64 Cross Tools Command Prompt
Sources are in.....D:\Source\QT\qt-src-5.5.0\qtbase
Build is done in.....D:\Source\QT\qt-src-5.5.0\qtbase
Install prefix.....D:\Source\QT\qt-src-5.5.0\qtbase
Headers installed to.....D:\Source\QT\qt-src-5.5.0\qtbase\include
Libraries installed to.....D:\Source\QT\qt-src-5.5.0\qtbase\lib
Arch-dep. data to.....D:\Source\QT\qt-src-5.5.0\qtbase
Plugins installed to.....D:\Source\QT\qt-src-5.5.0\qtbase\plugins
Library execs installed to..D:\Source\QT\qt-src-5.5.0\qtbase\bin
QML1 imports installed to...D:\Source\QT\qt-src-5.5.0\qtbase\imports
QML2 imports installed to...D:\Source\QT\qt-src-5.5.0\qtbase\qml
Binaries installed to.....D:\Source\QT\qt-src-5.5.0\qtbase\bin
Arch-indep. data to.....D:\Source\QT\qt-src-5.5.0\qtbase
Docs installed to.....D:\Source\QT\qt-src-5.5.0\qtbase\doc
Translations installed to...D:\Source\QT\qt-src-5.5.0\qtbase\translations
Examples installed to.....D:\Source\QT\qt-src-5.5.0\qtbase\examples
Tests installed to.....D:\Source\QT\qt-src-5.5.0\qtbase\tests

Info: creating super cache file D:\Source\QT\qt-src-5.5.0\.qmake.super

Qt is now configured for building. Just run nmake.
To reconfigure, run nmake confclean and configure.

D:\Source\QT\qt-src-5.5.0>nmake
```

Рисунок 4.20 – Вызов nmake

10. После того как сборка закончена, посмотрите выведенные сообщения.

Сборка может закончиться с ошибкой. Например, «qt fatal error u1077 “python”». В дальнейших лабораторных работах будет использоваться язык C/C++, нет необходимости в сборке библиотеки QT, для других языков, поэтому сообщение о данной ошибке можно проигнорировать.

11. Скомпилированные файлы находятся в папке ... \qt\qtbase\

Файлы .dll в папке ... \qt\qtbase\bin.

Файлы .lib в папке ... \qt\qtbase\lib.

4.5 Лабораторная работа № 5 «Поиск утечек памяти»

Задание.

Провести поиск утечек памяти и описать процесс их устранения в программе, согласно варианту индивидуального задания.

Установка Visual Leak Detector:

1. Перейдите на сайт проекта <http://vld.codeplex.com/>

Смотрите рисунок 4.21.

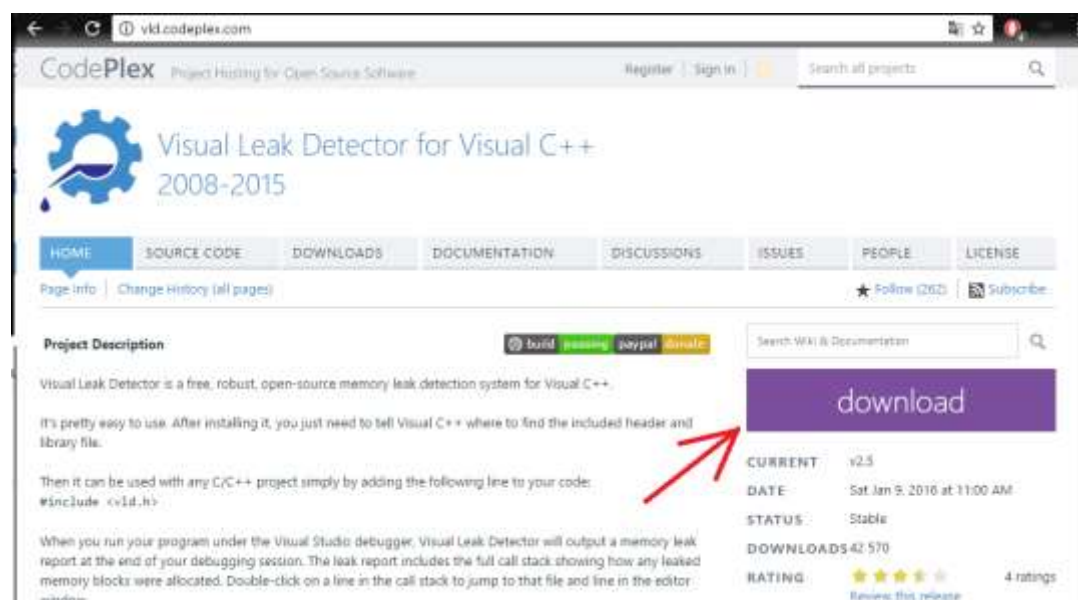


Рисунок 4.21 – Сайт проекта Visual Leak Detector

2. Нажмите на кнопку download – на ПК загружается исполняемый файл. Для установки следуйте инструкциям мастера установки.

3. Для использования средства Visual Leak Detector необходимо добавить в код строку:

```
#include <vld.h>
```


4. Теперь при запуске программы из среды разработки в режиме «Debug» в окне «Output» будут выведены сообщения об имеющихся утечках памяти с указанием файла исходного кода и строки, выполнение которой вызывает утечку памяти.

5. Создайте новый проект в среде разработки Visual Studio на компьютере с установленным Visual Leak Detector. Добавьте в проект файлы исходного кода согласно варианту индивидуального задания.

6. Скомпилируйте и запустите проект в режиме отладки “Debug”.

7. В окне результатов «Output» найдите информацию о первой утечке памяти. Откройте в редакторе указанный файл, найдите указанную строку. Найдите участок кода, по завершении выполнения которого указанная память больше не будет использоваться и освободите участок памяти с помощью функции “free()” или оператора “delete”.

8. Повторите действия пунктов 6 и 7 для всех утечек памяти, выявленных в файлах исходного кода.

9. По завершении отладки убедитесь в отсутствии утечек памяти.

4.6 Лабораторная работа № 6 «Профилирование и оптимизация»

Задание.

Провести сбор характеристик работы программы, разработанной согласно варианту индивидуального задания. Осуществить оптимизацию, тестирование, отладку и устранение ошибок.

Методика поиска "горячих" точек программы посредством профилировки отдельных ее участков.

1. Для проведения замеров скорости выполнения участков можно воспользоваться любым из способов, описанных в [3]. Для примера воспользуемся функциями из файла profiler.h.

2. В первую очередь необходимо разделить программу на подпрограммы, и выяснить, какой из участков кода выполняется дольше всего. После того, как наиболее ресурсоемкий участок кода будет локализован- необходимо произвести его оптимизацию.

```
void foo1()
{
//тело функции
}
void foo2()
{
//тело функции
}
int main()
```

```
{
    foo2();
    foo1();
    return 1;
}
```

3. В данном случае в функции `main` происходит вызов двух функций `foo1` и `foo2`, каждая из которых выполняет некоторые действия. Для того, чтобы понять, как можно добиться прироста скорости выполнения программы, разделим функцию `main` на участки и замерим скорость выполнения каждого из участков.

```
int main()
{
    profiler p1, p2;
    p1.set_freq(3.3 * 1000000000);
    p2.set_freq(3.3 * 1000000000);

    p1.start();
    foo1();
    p1.stop();

    p2.start();
    foo2();
    p2.stop();

    std::cout << "foo1 time= " << p2.get_rel() << std::endl;
    std::cout << "foo2 time= " << p1.get_rel() << std::endl;
    return 1;
}
```

}

Примечание:

"profiler" это класс, предоставляющий методы для работы с измерением времени выполнения участков кода. Функция `set_freq()` устанавливает тактовую частоту, на которой работает процессор (в герцах), функции `start` и `stop` запускают и останавливают замер времени. Функция `get_rel()` возвращает время выполнения участка кода.

4. Выбрать функцию с самым большим временем выполнения. Определить причину низкой производительности кода. При необходимости разбить функцию на несколько участков меньшего размера и провести измерения повторно.

5. После выявления "горячей точки" определить, возможно ли оптимизировать данный участок и провести необходимые модификации программного кода.

6. Повторять действия пунктов 4 и 5 до тех пор, пока не будут модифицированы все низкопроизводительные участки.

Библиографический список

1. Скотт М. Эффективное использование С++ – 3-е изд. – М.: «ДМК Пресс», 2012. – 300 с.
2. Страуструп Б. Программирование: принципы и практика использования С++, исправленное издание. – М.: «Вильямс», 2011. – 1248 с.
3. Ватутин Э.И. Методика измерения времени выполнения заданного фрагмента программы [Электронный ресурс] Дата обновления 08.06.16. URL: <http://evatutin.narod.ru/forstd.html> (дата обращения: 05.10.2016).
4. Интерактивный учебник по системе контроля версий Git [Электронный ресурс] URL: <https://githowto.com/ru>.
5. Шлее М. Qt 5.3. Профессиональное программирование на С++. – СПб: БХВ_Петербург, 2015. – 928 с.
6. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. П. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – СПб: Питер, 2001. – 368 с.