CST8277 Enterprise Application Programming

Assignment 2: Client/Server – See Brightspace for due date.

- **Read the requirements carefully, if your program does not meet the requirements expect a loss of marks for the assignment, even though your program code runs without errors.**
- **Refer to the Course Section Information (CSI) document posted in Brightspace under Course Information for additional requirements common to all assignments.**

Lab Partner: Optional

- For this lab, you may work with a lab partner.
  - Your lab professor must be notified via email of partners with names before the due date. Please include lab section number. <u>Both members must be registered in the same lab section.</u>
  - Both partners must demonstrate their program in lab together as a pair, all code files and other documents must have both members full names.
  - Both partners need to submit the assignment in full to Brightspace, your lab professor will select one student's submission and use it as the grade for both partners. Ensure that both submissions are the same.

Purpose:

In the first part of this exercise, you will compile and link a small C++ client and server (provided) to become familiar with compiling and linking outside the Java environment. Then we'll use our knowledge of Java syntax to make a small change to the C++ server, and recompile it. Because you know Java, you already know enough C++ to make small changes to a C++ program.

In the second part of this exercise we will work with a multithreaded Java server program with a client. Using starter code provided, and the multithreading Echo client-server from lecture as a reference you will create a client that sends Tuna objects to the server for insertion into a database using a simple text-based protocol and Java Serialization to marshal communications.

Lab Environment Issue:

- The wireless network in the labs can make it impossible for a client program to connect to a server program running on two physically separate computers. For testing and demonstration of the lab if this should happen fallback to testing and demonstrating the client and server by running both on the same computer.

Part 1: Client/Server in C++

A. Compile and Link the given C++ code from the EchoServerAndAPI folder (see the **Compiling and Linking** section below). Then create a simple test plan, and begin your testing by testing the client and server running on the same computer. Use the computer name *localhost* and an application number (port) available on your computer (8080 or 8081 are often safe choices). Then, find a testing partner and test the programs by running the client and server on different machines (this situation should be added to your test plan if it's not there already). First run one partner's server on their machine, and connect with the other partner's client running on their machine. Then do the opposite, so that both servers have been tested with a different programmer's client running from a different machine.

B. You'll notice in Step A above that the Server as given will terminate after servicing just one client. Modify the server code so that the server can accept connections from multiple clients (only one at a time, one after the other - no multithreading needed). Recompile the Server, and test it again (be sure your test plan includes all the appropriate tests) with your testing partner, as you did in Step A.

C. Write a brief essay, in your own words, to explain the difference between preprocessing, compiling, and linking in C++. Look up the answer online and cite your sources using IEEE reference style.

Note:

There is absolutely no need to add multithreading to the C++ program, just modify the code so that it handles more than one client sequentially.

C++ program usage notes

- Start the C++ echoserver program in a command prompt window, the cursor will not return, nor can you type, this indicates the server is running. (If the cursor immediately returns the server is not running, it experienced a problem).
- Start a C++ echoclient program in another command prompt window, providing the ip address of the server, if running both server and client on one machine use 127.0.01 or localhost.
- The client might take a moment to initialize, type messages to see them echoed back by the server.
- Use the enter key to send a blank message (no other test, just the enter key) to disconnect from the server.
- To shut down the server use the keyboard combination ctrl+c within it's command prompt window
- Note: Only the developer command prompt is needed to compile and link, a normal command prompt window can run the program executables.

Part 2: Multithreaded Server and Client in Java

Project setup

- Run the database script to create a new database for assignment2, it is identical to the database used in assignment1 but by separating databases in MySQL it will be easier to test.
- You may also want to create an assignment2 user with password set to password (or modify the relevant fields in the starter code to match your database).
- Create a project in Eclipse, then copy and paste the provided package folders into the src folder
- Reference (copy as starter code) the multithreaded EchoServer and the EchoClient when writing your Client and Server

UML Class Diagram class Message

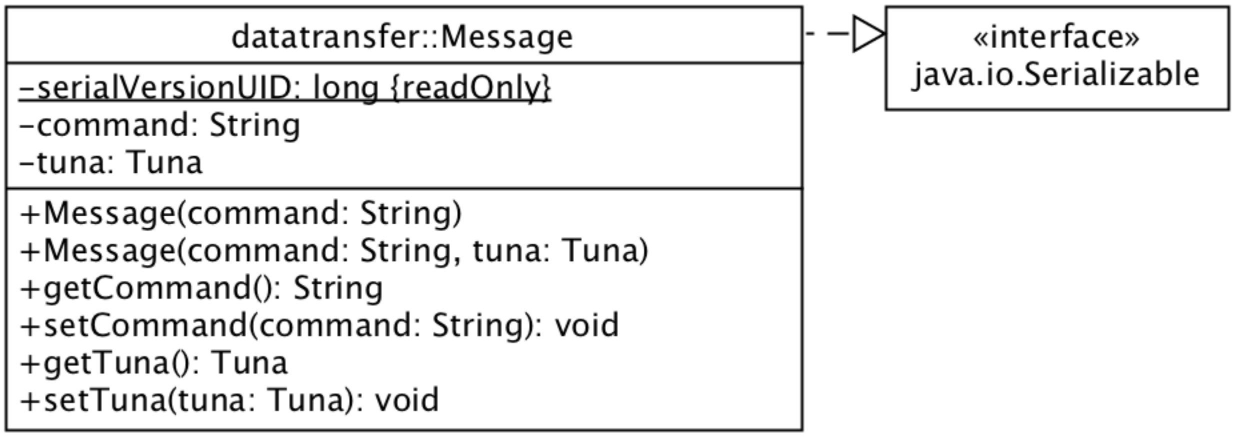

datatransfer::Message
-serialVersionUID: long {readOnly}
-command: String
-tuna: Tuna
+Message(command: String)
+Message(command: String, tuna: Tuna)
+getCommand(): String
+setCommand(command: String): void
+getTuna(): Tuna
+setTuna(tuna: Tuna): void
«interface»
java.io.Serializable

Figure 1.0 Class Message.
Class Message is simply designed to be a serializable wrapper around a String command and a Tuna object, the command is a String that both the server and client will use as a simplistic protocol.

The Simple Protocol – These are Strings, placed within the Message objects command field.

insert (client sends to server)
client would like the server to insert the Tuna into the database
insert_success (server sends to client)
server inserted Tuna, and is returning it with generated primary key inserted.
insert_failed (server sends to client)
server is reporting to client that a problem occurred, there will be no Tuna sent back to client, additionally server and client should treat this as a disconnect (see below)
disconnect (client and server send to each other)

sent between both client and server to indicate that the network sockets and streams should be closed, the client will exit while the server keeps running for more clients.

TunaServer

Use the multi-threaded Java EchoServer and modify it to use Message objects and the protocol above to do work. The data-access-layer is provided to you, it is partially complete but all the functionality you need for this assignment already done. Tip: if(message.getCommand().equalsIgnoreCase("insert"))…

TunaClient

Use the Java EchoClient and modify it to use Message objects and the protocol above to do work.

Short Answer Questions

1. What is a serialVersionUID and why can omitting it cause problems (consider cross-platform development with different Java compilers and run-times)?
2. What is a UUID? What can it be used for?
3. Research and very briefly describe each of these protocols (cite your sources): SMTP, FTP, HTTP, SOAP+XML, and REST+JSON (What does the acronym mean, in general what is it used for?).

Typical workflow between Client and Server using serialized Messages (Java Part)

1. Start Server and then Start Client, Server & Client will set up Streams (etc.)
2. Client: Prompt the user to enter the values for a new Tuna
3. Client: Generate a new UUID for the Tuna, set as String into Tuna
4. Client: Place the Tuna into a Message with Command of add
5. Client: Send Message to Server via serialization
6. Server: Receive and de-serialize Message from Client

   If Message is insert
   - Insert the Tuna to the database, then…
   - Select the Tuna from the database using UUID to obtain the generated PK
   - Send a Message with command insert_success and Tuna to Client with PK and other values set
   - Note: If any exceptions occur
     - send a new Message with insert_failed, and null for Tuna to the client

   If Message is disconnect
   - Return message to client with Command disconnect and a null for Tuna
   - Exit processing loop for Client, closing Streams and Socket, before Thread dies.
7. Client: Receive and de-serialize Message from Server

   If Message is insert_success
   - Display the Tuna (with PK that is not zero) to Console
   - Ask user if they want to insert another Tuna

     If yes

       repeat step 2 onwards with collecting data for fresh Tuna

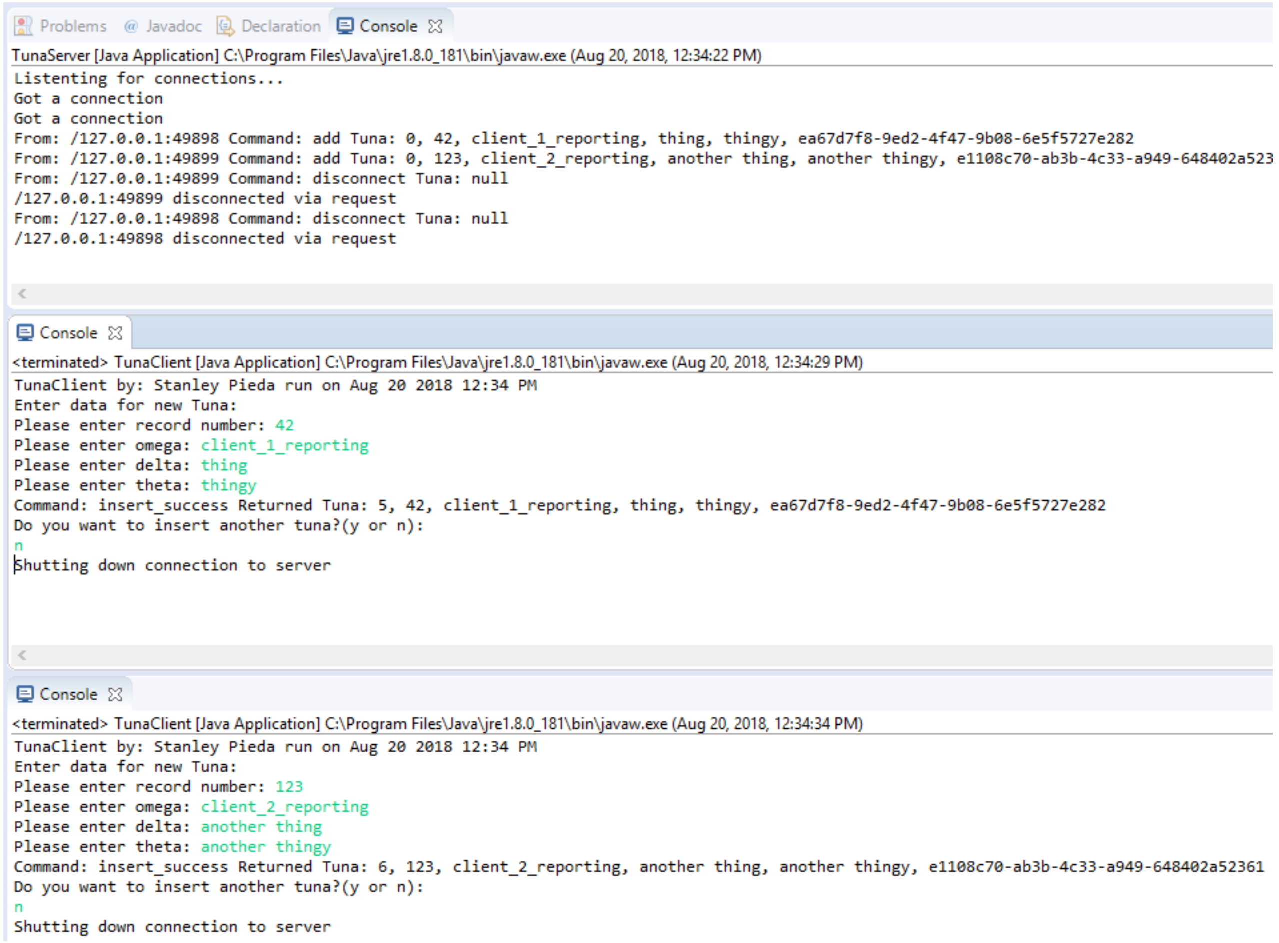
     If no

       send Message with Command disconnect to Server

   If Message is insert_failed
   - Display ambiguous message similar to: "Insert did not succeed or similar", then…
   - Send disconnect to server, should get disconnect back (see below)

   If Message is disconnect
   - Exit processing loop, close streams and socket, tell user Client program shut down

Example Screen Shot of TunaServer with two Tuna Clients, normal use.



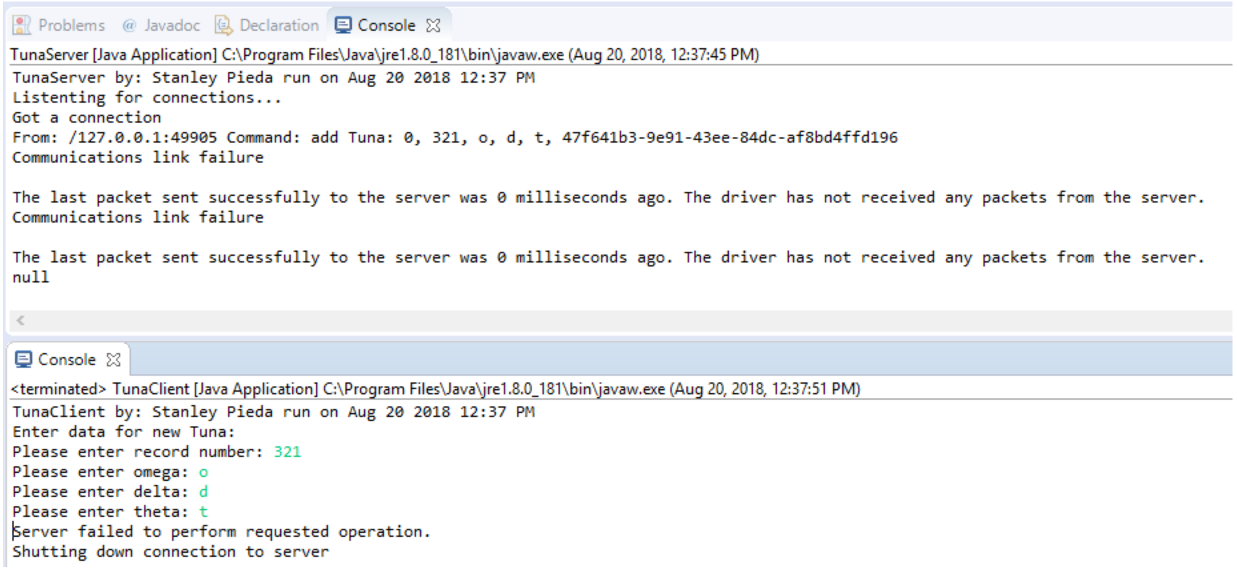
```
TunaServer [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Aug 20, 2018, 12:34:22 PM)
Listenting for connections...
Got a connection
Got a connection
From: /127.0.0.1:49898 Command: add Tuna: 0, 42, client_1_reporting, thing, thingy, ea67d7f8-9ed2-4f47-9b08-6e5f5727e282
From: /127.0.0.1:49899 Command: add Tuna: 0, 123, client_2_reporting, another thing, another thingy, e1108c70-ab3b-4c33-a949-648402a523
From: /127.0.0.1:49899 Command: disconnect Tuna: null
/127.0.0.1:49899 disconnected via request
From: /127.0.0.1:49898 Command: disconnect Tuna: null
/127.0.0.1:49898 disconnected via request
```

```
<terminated> TunaClient [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Aug 20, 2018, 12:34:29 PM)
TunaClient by: Stanley Pieda run on Aug 20 2018 12:34 PM
Enter data for new Tuna:
Please enter record number: 42
Please enter omega: client_1_reporting
Please enter delta: thing
Please enter theta: thingy
Command: insert_success Returned Tuna: 5, 42, client_1_reporting, thing, thingy, ea67d7f8-9ed2-4f47-9b08-6e5f5727e282
Do you want to insert another tuna?(y or n):
n
Shutting down connection to server
```

```
<terminated> TunaClient [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Aug 20, 2018, 12:34:34 PM)
TunaClient by: Stanley Pieda run on Aug 20 2018 12:34 PM
Enter data for new Tuna:
Please enter record number: 123
Please enter omega: client_2_reporting
Please enter delta: another thing
Please enter theta: another thingy
Command: insert_success Returned Tuna: 6, 123, client_2_reporting, another thing, another thingy, e1108c70-ab3b-4c33-a949-648402a52361
Do you want to insert another tuna?(y or n):
n
Shutting down connection to server
```

Here both clients could insert one Tuna object via the server and get back generated primary keys with the other information that was sent.

Example TunaServer with one TunaClient, MySQL service shut down results in exception.



```
TunaServer [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Aug 20, 2018, 12:37:45 PM)
TunaServer by: Stanley Pieda run on Aug 20 2018 12:37 PM
Listening for connections...
Got a connection
From: /127.0.0.1:49905 Command: add Tuna: 0, 321, o, d, t, 47f641b3-9e91-43ee-84dc-af8bd4ffd196
Communications link failure

The last packet sent successfully to the server was 0 milliseconds ago. The driver has not received any packets from the server.
Communications link failure

The last packet sent successfully to the server was 0 milliseconds ago. The driver has not received any packets from the server.
null
```

```
<terminated> TunaClient [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Aug 20, 2018, 12:37:51 PM)
TunaClient by: Stanley Pieda run on Aug 20 2018 12:37 PM
Enter data for new Tuna:
Please enter record number: 321
Please enter omega: o
Please enter delta: d
Please enter theta: t
Server failed to perform requested operation.
Shutting down connection to server
```

Behind the scenes, the command “net stop mysql57” was used in an administrative console to shut down MySQL while the client and server were running, you can use Control Panel, Admin Tools, Services.
The DAO code prints out the exception message, then re-throws the exception, so the exception message appears more than once in the Server console window as it gets bubbled upwards.

Submission Requirements

Demonstrate C++ program in Lab

- Compiling and Linking the echoserver.cpp file in a command prompt window
- Running echo server in one command prompt window
- Running two echo clients sequentially in separate command prompt windows
- Note: The echoserver should keep running when the first client disconnects, then start communicating with the second client.

Demonstrate Java program in lab

- Run the TunaServer
- Run two clients, show how they can both communicate to the server at the same time
- Shutdown MySQL and show how a client is notified of a problem and shuts down gracefully

Documentation and Code Uploads

1. From Part I:
   - Your C++ server source file (echoserver.cpp).
   - Your essay answering part one questions.
   - Test plan for C++ program.
2. From Part II
   - Your Java Project Folder.
   - Your essay answering part two questions.
   - Your test plan for the Java program.

Grading (10 points total)

EchoClient, EchoServer with C++ (4 points)

- Demonstrate Compiling and Linking echoserver.cpp (1)
- Demonstrate program use, one server, two clients, each client handled in sequence (1)
- Essay on C++ preprocessor, compiler, linker, with IEEE reference style to cite sources (1)
- C++ Test Plan (1)

TunaClient, TunaServer with Java (6 Points)

- Demonstrate Server handling two clients at same time, inserting one Tuna each, both disconnecting and shutting down because user(s) entered 'n' for more Tuna (1)
- Demonstrate Client gracefully shutting down because of failure on server-side (1)
- Comment block at top of file(s), all classes, class-members have Javadoc comments (1)
  - Your name must appear at the top of each file in a comment block
- Client and Server do not have resource leaks (1)
- All short answer questions answered (1)
- Java Test Plan (1)

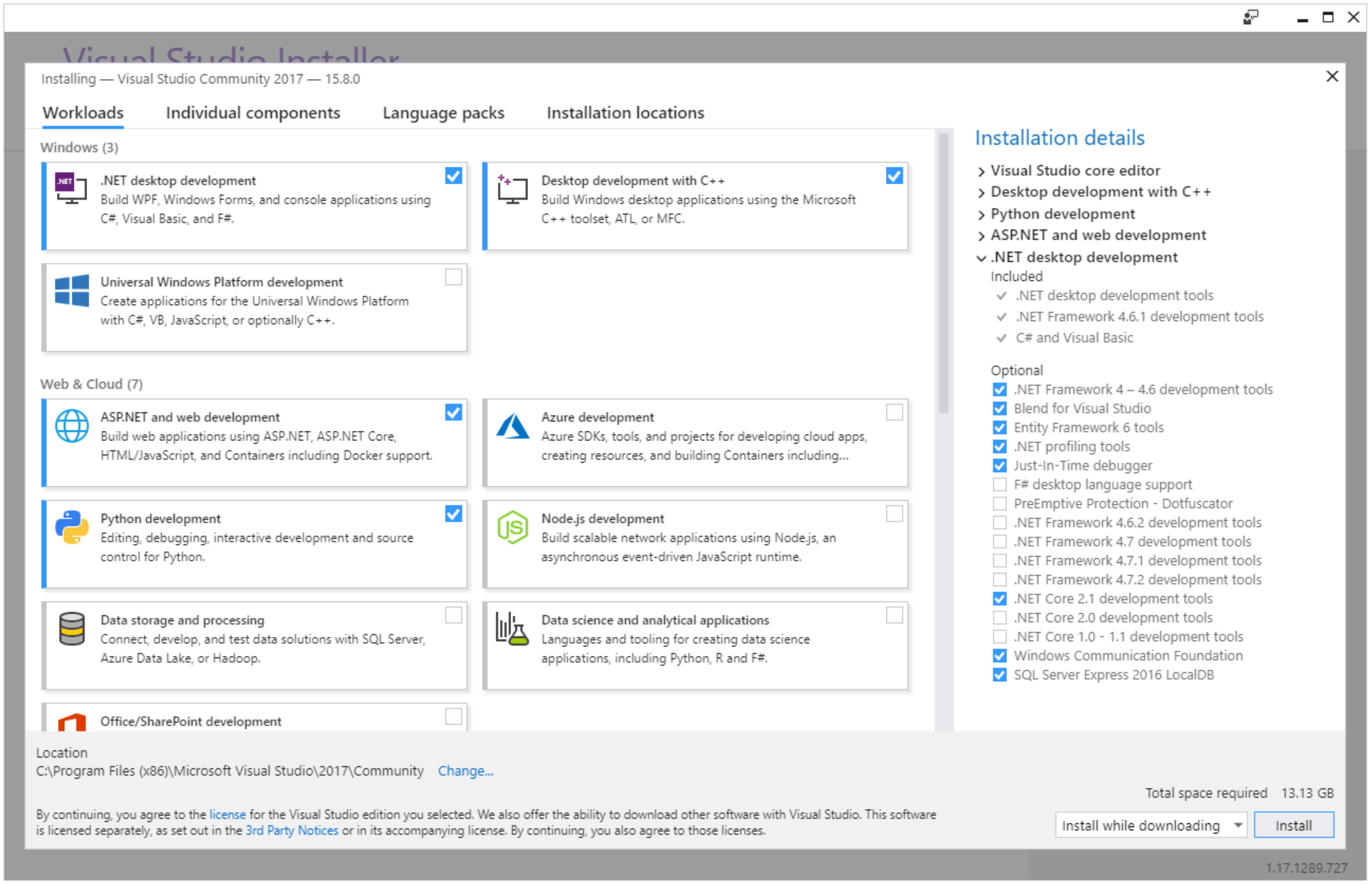
Note:

- Create separate test plans for the C++ and Java programs

C++ Compiling and Linking Instructions

Visual Studio Download and Install

- Get Visual Studio 2017 Community Edition via Algonquin College Digital Resources Portal
- Visual Studio 2017 Community Edition (or newer) does not install C++ support by default.
- If installing Visual Studio for the first time, you will be given the option to install C++ support, see screen shot below:



Visual Studio Installer

Installing — Visual Studio Community 2017 — 15.8.0

Workloads | Individual components | Language packs | Installation locations

Windows (3)

- ✓ .NET desktop development — Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F#.
- ✓ Desktop development with C++ — Build Windows desktop applications using the Microsoft C++ toolset, ATL, or MFC.
- ☐ Universal Windows Platform development — Create applications for the Universal Windows Platform with C#, VB, JavaScript, or optionally C++.

Web & Cloud (7)

- ✓ ASP.NET and web development — Build web applications using ASP.NET, ASP.NET Core, HTML/JavaScript, and Containers including Docker support.
- ☐ Azure development — Azure SDKs, tools, and projects for developing cloud apps, creating resources, and building Containers including...
- ✓ Python development — Editing, debugging, interactive development and source control for Python.
- ☐ Node.js development — Build scalable network applications using Node.js, an asynchronous event-driven JavaScript runtime.
- ☐ Data storage and processing — Connect, develop, and test data solutions with SQL Server, Azure Data Lake, or Hadoop.
- ☐ Data science and analytical applications — Languages and tooling for creating data science applications, including Python, R and F#.
- ☐ Office/SharePoint development

Installation details

- › Visual Studio core editor
- › Desktop development with C++
- › Python development
- › ASP.NET and web development
- ⌄ .NET desktop development
  - Included
    - ✓ .NET desktop development tools
    - ✓ .NET Framework 4.6.1 development tools
    - ✓ C# and Visual Basic
  - Optional
    - ✓ .NET Framework 4 – 4.6 development tools
    - ✓ Blend for Visual Studio
    - ✓ Entity Framework 6 tools
    - ✓ .NET profiling tools
    - ✓ Just-In-Time debugger
    - ☐ F# desktop language support
    - ☐ PreEmptive Protection - Dotfuscator
    - ☐ .NET Framework 4.6.2 development tools
    - ☐ .NET Framework 4.7 development tools
    - ☐ .NET Framework 4.7.1 development tools
    - ☐ .NET Framework 4.7.2 development tools
    - ✓ .NET Core 2.1 development tools
    - ☐ .NET Core 2.0 development tools
    - ☐ .NET Core 1.0 - 1.1 development tools
    - ✓ Windows Communication Foundation
    - ✓ SQL Server Express 2016 LocalDB

Location
C:\Program Files (x86)\Microsoft Visual Studio\2017\Community Change...

By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.

Total space required 13.13 GB

Install while downloading | Install

1.17.1289.727

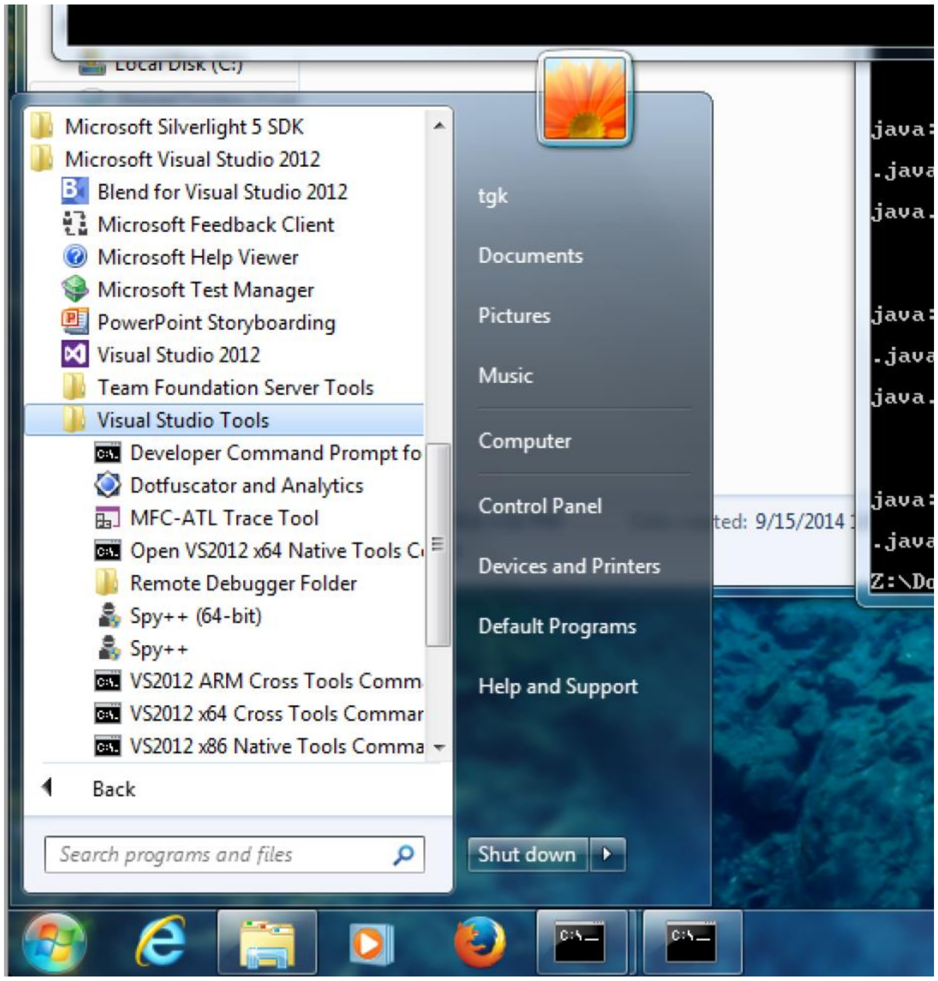
You may also select programming languages and tools that interest you, here we need only C++, keep an eye on the install size. In the screen shot above the professor was interested in .Net, C++, ASP.Net and Python making the install size 13.13 GB (look in lower right corner of screen shot).

Microsoft Windows

- Note: The screen shots and instructions below were created for Visual Studio 2012 running on Windows 7. There are no changes needed to the instructions, they work for Visual Studio 2017 and Windows 10.
- In Windows 10 use the start-menu search for “Developer Command Prompt for VS 2017” then right-click and either add to start menu or taskbar for quicker access then run.

To compile the C++ api library, the client, and the server on MS Windows, we will use MS Visual Studio. You may use the full Visual Studio graphical IDE if you wish, but instructions for using the full graphical environment of Visual Studio are not provided here (they would involve many screenshots that might be out of date compared to the Visual Studio version you have). For this simple little C++ project, it's easier to use the Command Line.

First, invoke a development Command Line. In Windows 7, you can find this under **Start->Microsoft Visual Studio 2012->Visual Studio Tools->Developer Command Prompt for VS2012** (adjust as necessary for other versions of Visual Studio).


Local Disk (C:)
Microsoft Silverlight 5 SDK
Microsoft Visual Studio 2012
Blend for Visual Studio 2012
Microsoft Feedback Client
Microsoft Help Viewer
Microsoft Test Manager
PowerPoint Storyboarding
Visual Studio 2012
Team Foundation Server Tools
Visual Studio Tools
Developer Command Prompt fo
Dotfuscator and Analytics
MFC-ATL Trace Tool
Open VS2012 x64 Native Tools C
Remote Debugger Folder
Spy++ (64-bit)
Spy++
VS2012 ARM Cross Tools Comm
VS2012 x64 Cross Tools Comma
VS2012 x86 Native Tools Comma
Back
Search programs and files
tgk
Documents
Pictures
Music
Computer
Control Panel
Devices and Printers
Default Programs
Help and Support
Shut down
ted: 9/15/2014

In this Command Window, change to the API directory:

```
cd EchoServerAndAPI\api
```

Use the C++ compiler command (**cl**) to compile all of the C++ files there, using the following command line:

```
cl /c /DWIN32 *.cpp
```

We used the **/c** switch to tell the **cl** command that it should just create the object files and not try to make executable programs out of these files. If you have never heard of an "object file" you probably owe it to yourself to do a quick google for "object file" and find out what it is. We also used the **/DWIN32** switch to define the **WIN32** symbol during preprocessing so that anywhere in our C++ files where there is section of C++ code that has both **LINUX** and **WIN32** versions of that piece of code, the **WIN32** version of that piece of code is used.
Now, create a library out of those compiled files (***.obj**) called **api.lib** using the following command line:

```
lib /out:api.lib *.obj
```

You now have the **api.lib** library, and you will not need to recompile it again as long as you don't change any of the C++ files in the API directory (and you're not expected to change those files!) This **api.lib** is a static library. In the past you may have encountered dynamic libraries, which have the **.dll** extension. You're now doing the same kinds of operations that are done to create Microsoft Windows itself (or Linux, or OSX, etc). Is that cool, or what?
Now change to the APPS directory:

```
cd ..\apps
```

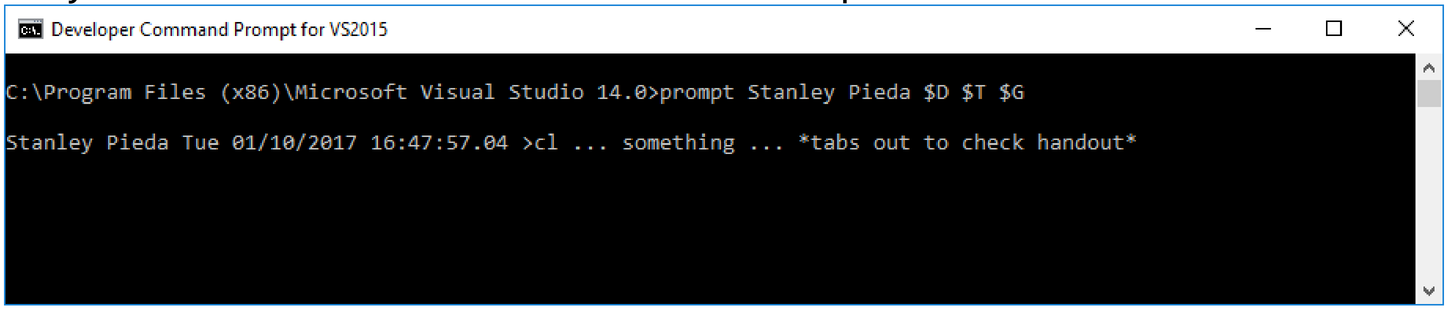
Compile the **echoclient.cpp** file:

```
cl /c /DWIN32 echoclient.cpp
```

Compile the **readln.cpp** file:

```
cl /c /DWIN32 readln.cpp
```

Note:
Optionally use the following command before the next step: `prompt Student Name $D $T $G`
($D for date, $T for time, $G for greater-than, See: http://ss64.com/nt/prompt.html)
`Use your own name instead of “Student Name” for example:`



```
C:\Program Files (x86)\Microsoft Visual Studio 14.0>prompt Stanley Pieda $D $T $G

Stanley Pieda Tue 01/10/2017 16:47:57.04 >cl ... something ... *tabs out to check handout*
```

Compile the **echoserver.cpp** file
```
cl /c /DWIN32 echoserver.cpp
```

Now we will create the **echoserver.exe** program. We do this by *linking* the echoserver object module, **echoserver.obj**, with the **api.lib** library we created and the Microsoft Windows **ws2_32.lib** library, provided by Microsoft Windows itself, to create the **echoserver.exe** program. Libraries contain common functions that are used by many other different programs that need those functions. To link, we use the **link** command:

```
link /out:echoserver.exe echoserver.obj  ..\api\api.lib ws2_32.lib
```

Now we link the echoclient program, which is split into two C++ files **echoclient.cpp** and **readln.cpp**. The linking creates the **echoclient.exe** program:

```
link /out:echoclient.exe echoclient.obj readln.obj  ..\api\api.lib ws2_32.lib
```

You're finished building the programs. To run them as intended, you need to run the server first, having it listen to port 8080 for example (from the folder where **echoserver.exe** resides):

```
echoserver 8080
```

Run the clients, telling them which machine the server is running on, and which port to connect to (from the folder where **echoclient.exe** resides):

```
echoclient  localhost  8080
```

or

```
echoclient  10.50.225.13  8080
```

etc...

Appendix Expected Format for Test Plan (Sample entries would be for the C++ program)

Description	Pre-Conditions	Post-Conditions	Test Methodology	Expected	Actual	Notes
First C++ Client can connect to server	Server is already running on port 8081	First Client connects to server	Run client and server enter hello and view response	Client connects, sends “hello”, gets echo response of “hello” with additional information from server.	Matches	
Second C++ Client can connect to server	Server is already running on port 8081 with first client still connected	Second client can connect but is not responsive until first client disconnects	Run server, run first client, run second client, shutdown first client	Second client should send and receive hello2 after first client is shutdown	Matches	

Test boundary conditions:
Can the C++ server handle more than one C++ client, sequentially?

Can the Multithreaded Java Server handle more than one client simultaneously on separate threads?
Does the server keep running if a client shuts down?
Are Tunas inserted into the database?
Does the server signal success and failure?
Does the server send the PK value to the client for the newly inserted record?
(etc.)

Test Methodology can include “view output”, “use debugger”, use JVisualVM, Etc.

Linux and Mac – Archival Only, not Supported or needed for lab

- This section is for archival purposes, the commands may or may not work with current Linux or Mac OSX operating systems, the course professor(s) will only provide troubleshooting and support for technologies running on Windows 10.

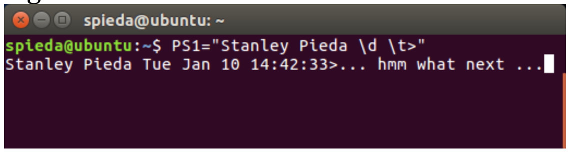
The LINUX version of the code works for both Linux and Mac, and the basic process is the same as for Windows, but the commands have different names.

Modify Terminal Prompt

Use:
`PS1="Student Name \d \t>"`

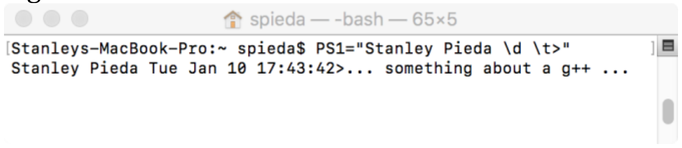
Change Student Name to your actual name

E.g. Unix



```
spieda@ubuntu:~$ PS1="Stanley Pieda \d \t>"
Stanley Pieda Tue Jan 10 14:42:33>... hmm what next ...
```

E.g. Mac



```
Stanleys-MacBook-Pro:~ spieda$ PS1="Stanley Pieda \d \t>"
Stanley Pieda Tue Jan 10 17:43:42>... something about a g++ ...
```

(when you re-open the terminal the default prompt will come back)

Compiling and Linking Instructions

```
cd EchoServerAndAPI/api

for f in *.cpp ; do  gcc -DLINUX -c $f; echo $f done; done

ar -q libapi.a *.o

cd ../apps

g++ -DLINUX -o echoclient echoclient.cpp readln.cpp -L../api -lapi

g++ -DLINUX -o echoserver echoserver.cpp -L../api -lapi
```