**Overview:**

Visual Studio 2017 has the option to build, debug, and run code remotely on any Linux system that has gdb server and ssh. Following this guide will allow you to compile, run, and debug code on general.asu.edu server, while within the Visual Studio IDE. Gdb server is required and is not included on general.asu.edu. This guide will also cover installing gdb from source code which includes the needed gdb server and ensuring that it is added to the PATH variable, for every session, allowing full functionality with Visual Studio. This guide will also cover making a Linux project in VS and how to configure and interface it with general.asu.edu.

**Prerequisites:**

1. Visual Studio Enterprise 2017 INSTALLER is currently installed on your system. If VS 2017 Installer is not installed, students can follow the necessary steps at [https://myapps.asu.edu/app/microsoft-imagine](https://www.google.com/url?q=https://myapps.asu.edu/app/microsoft-imagine&sa=D&source=hangouts&ust=1535317664602000&usg=AFQjCNFhXsM5Lwy1U0cziNMCfTW7hLoYFg)
2. You have access to the general ASU Linux server.
3. Have a ssh program installed on your computer, such as Putty, MobaXterm, WSL, etc. And know how to use it for log in.
4. This guide will allow you to build, debug, and run code on general.asu.edu from inside the Visual Studio IDE; however, this guide does not cover cmake option. You **will still need to write your own make files**, test and include it with your submission for assignments. Or configure VS cmake option to generate make files given that your professor allows IDE auto-generated make files to be used.
5. Know general Linux commands or be willing to google and understand them. This guide is not comprehensive.
6. It will take at least 30-40 minutes from start to finish, much of it is due to install and download times. The actual “working” time should be about 15-20 minutes.
7. (Optional) SCP or SFTP with a windows compatible program such as MobaXterm, WinSCP, WSL, etc. or myfiles at <https://webapp4.asu.edu/myfiles/app>.

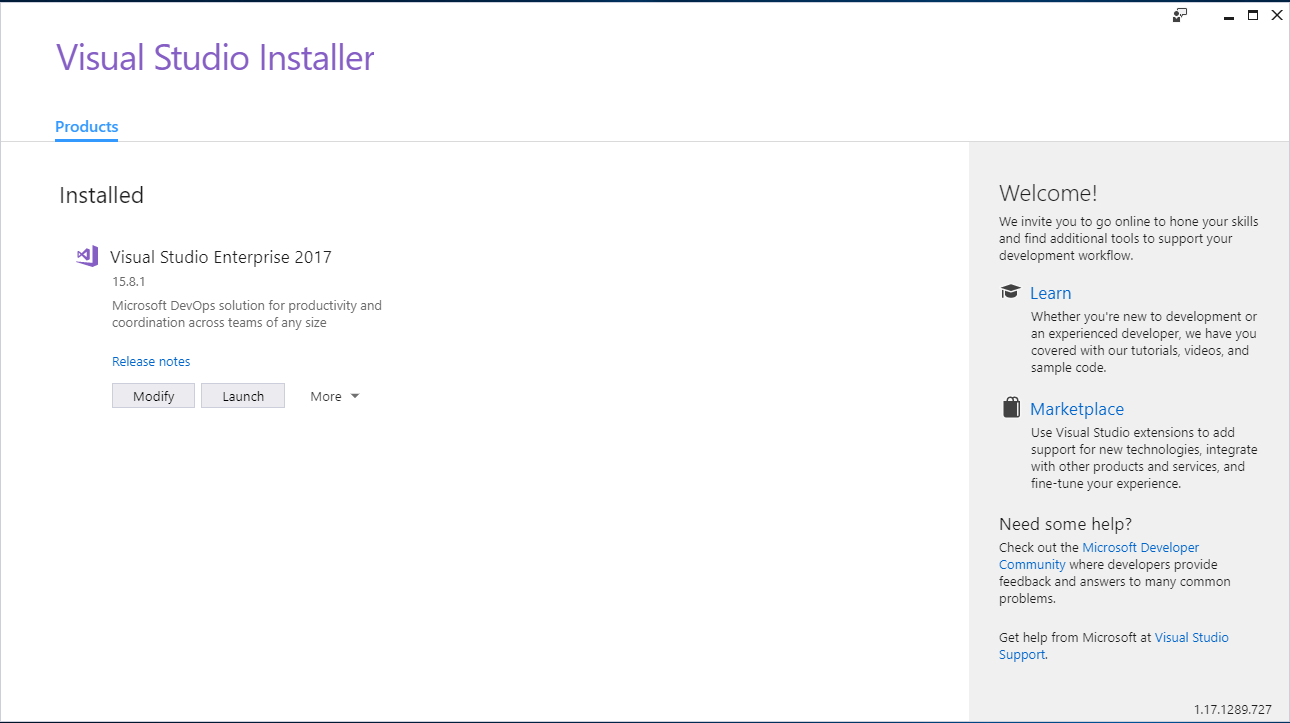
**Guide Contents:**

1. Install VS Linux Tool
2. Setting up and Configuring general.asu.edu
3. Using VS to Remote In
4. Configuring VS
5. Loading Input Files
6. Sources

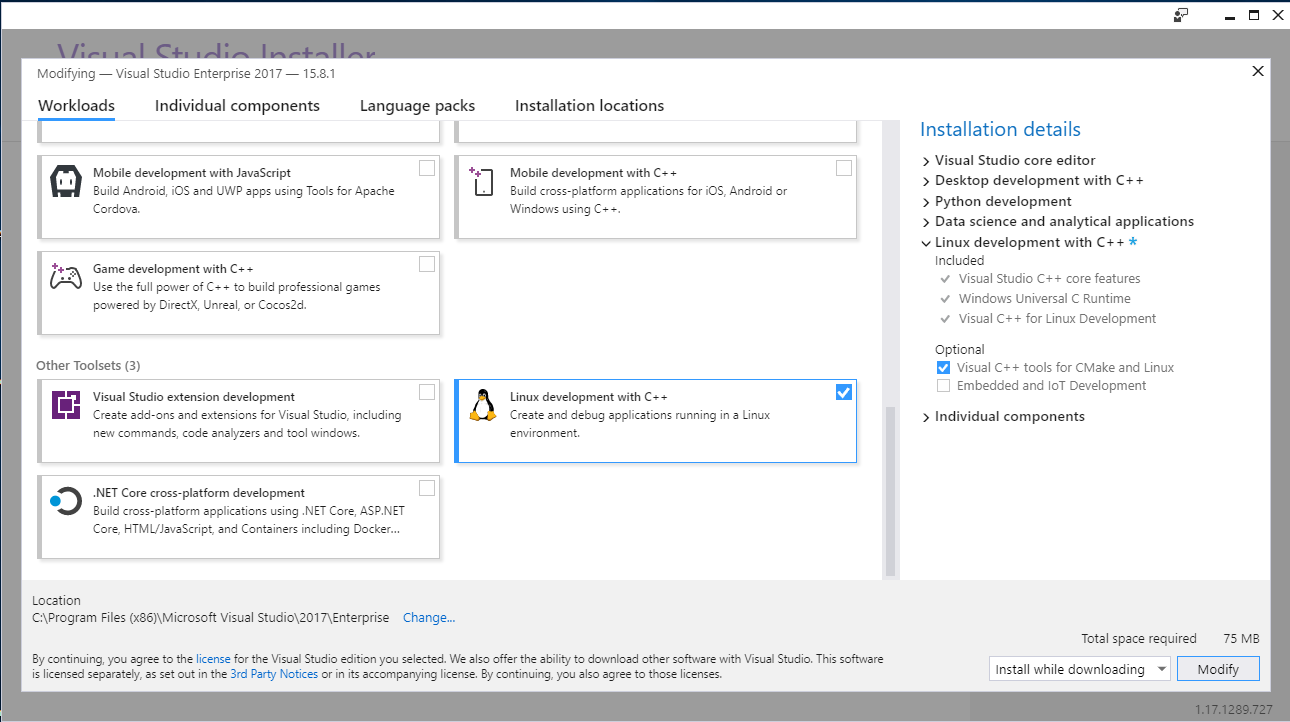
1. **Install VS Linux Tool**

Open Visual Studio Installer and update if necessary

Select modify



Select modify



1. **Setting up general.asu.edu**

The version of gdb that is installed on general.asu.edu does not contain gdb server which is required for debugging remotely. To install gdb without any errors there is another dependency to get textInfo. textInfo is not required for compatibility with Visual Studio.

Load your preferred ssh program and log into general.asu.edu

Make a folder to contain dependencies and downloaded files

Example

$ mkdir -p $HOME/local/downloads

**Download texInfo** (not required)

Texinfo is not required for full functionality of gdb. But will allow gdb installation to generate manuals for gdb.

Navigate to <https://ftp.gnu.org/gnu/texinfo/> and locate the latest release

$ cd $HOME/local/downloads

$ wget <https://ftp.gnu.org/gnu/texinfo/texinfo-6.5.tar.gz>

$ tar -xvf texinfo-6.5.tar.gz

The installation configuration file needs to be updated to target installation folder. The default folder is not accessible without super user access.

$ cd ./texinfo-6.5 #move into the directory

$ ./configure --prefix=$HOME/local #update the directory path

$ make install

$ export PATH=$PATH:$HOME/local/bin #add to PATH this session making

# #texInfo accessible to gdb installation

$ cd ~ #change to home directory

$ makeinfo --help #verify that texinfo is on PATH

**Downloading gdb** (required)

Navigate to https://ftp.gnu.org/gnu/gdb/ and locate the latest release. At writing the latest version is gdb 8.1

Record the url of the latest version

$ cd $HOME/local/downloads

$ wget <https://ftp.gnu.org/gnu/gdb/gdb-8.1.tar.gz>

$ tar -xvf gdb-8.1.tar.gz

$ cd ./gdb-8.1

$ ./configure --prefix=$HOME/local/ #update config file with correct path

$ make #This will take a about 15 minutes

$ make install

# If this is a new session and $HOME/local is not included in PATH add it now

$ export PATH=$PATH:$HOME/local/bin

$ cd ~

$ gdbserver --help #to test that it installed

**Configuring bash (required)**

Edits that have been made to PATH or any other variable will only be available in the current session. At next login gdbserver and texinfo will no longer be tied to PATH variable. To fix this we need to write two bash files to ensure PATH is updated at each login in with ssh.

**Writing .bashrc**

In the home folder use your preferred text editor. If you don’t have one nano is very simple to use and has a small learning curve. For this tutorial emacs was used. Note that other variable edits can be added to this file and/or aliases to existing commands.

Alternatively both of the files can be written locally then loaded. Both of the files need to be located in the home directory.

$ cd ~

$ emacs .bashrc

Enter the following:

# Copied from http://plato.la.asu.edu/MAT420/uenviron.pdf (page 35)

#

# A read in bash file to set path variables

#

umask 077

PATH="$HOME/local/bin:$PATH" ## add all executables in local/bin to PATH

#End of file

**Write .bash\_profile**

Note that if desired additional files can be loaded by modifying this bash file

$ cd ~

$ emacs .bash\_profile

#

# Copied from http://plato.la.asu.edu/MAT420/uenviron.pdf (page 35)

# .bash\_profile - read in every login.

#

if [ -f ~/.bashrc ]

then

source ~/.bashrc # runs .bashrc as if they where

# typed into this file

fi

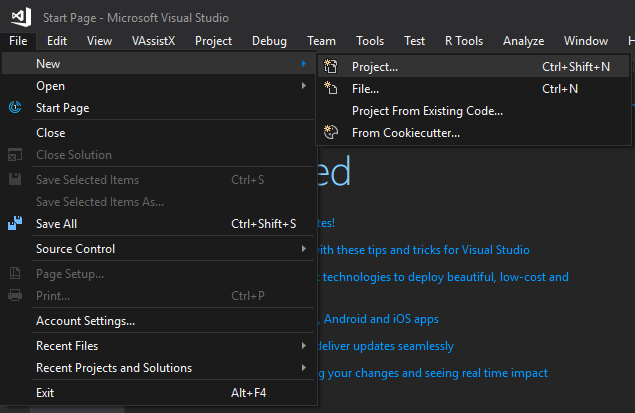
#End of file

1. Using VS to remote in

This portion of the guide will cover making a new project in VS and cover the necessary steps to remote into general.asu.edu.

Open VS

File -> New -> Project

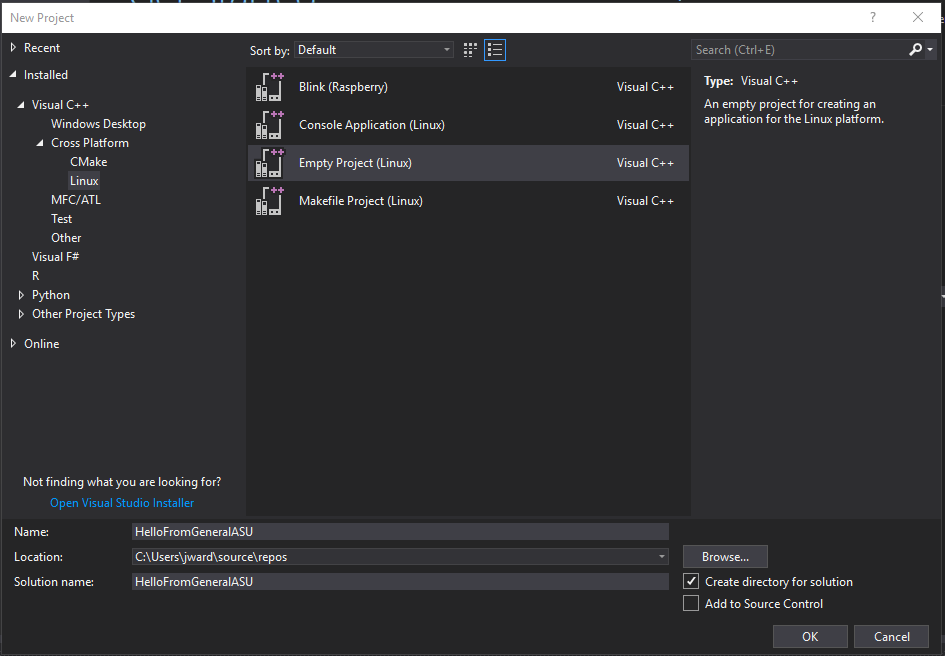


Installed -> Visual C++ -> Cross Platform -> Linux

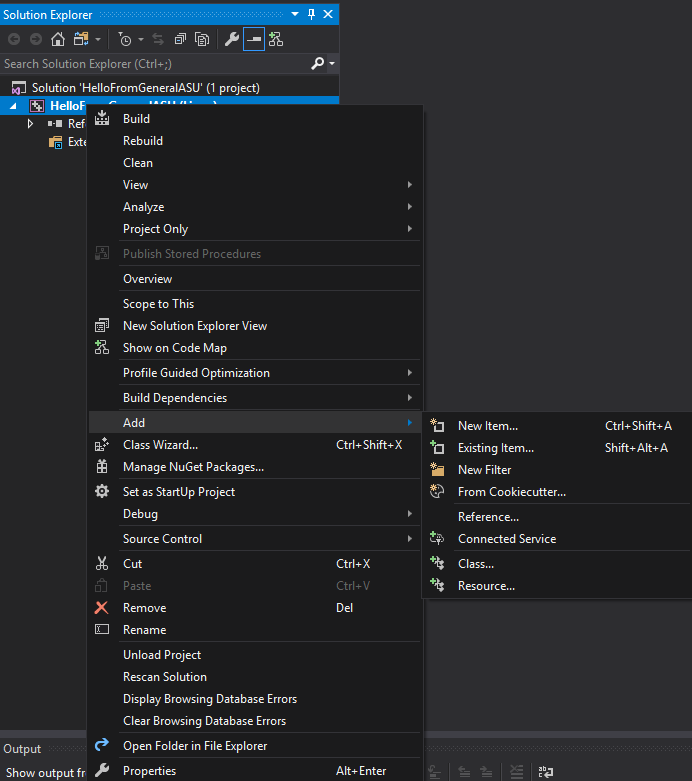
Select -> Empty Project Linux

Name it HelloFromGeneralASU

Click OK



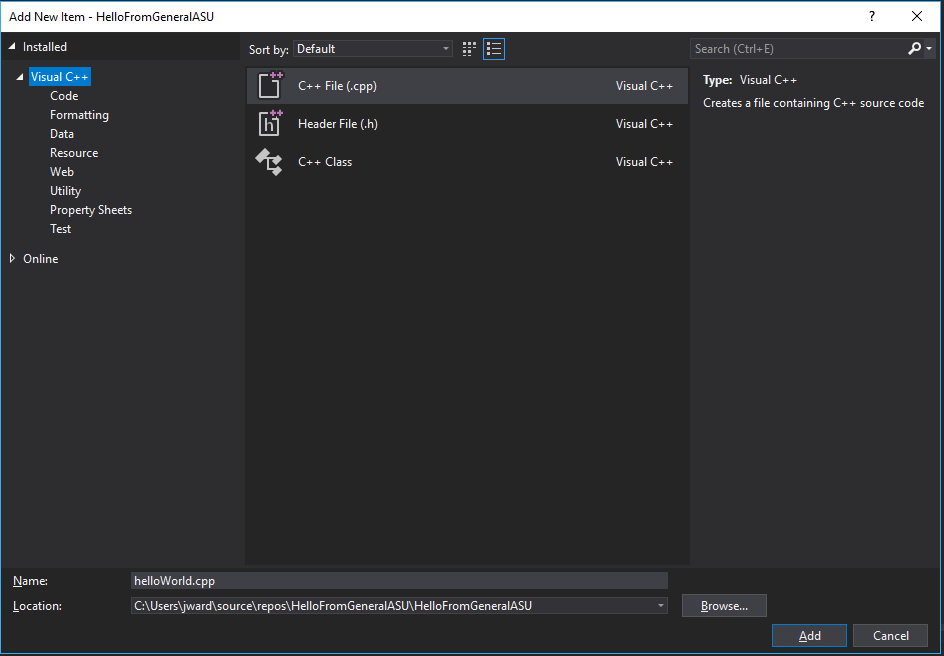
Right click the project -> Add -> New Item OR press ctr + shift + A



Select C ++ file (.cpp)

Name it helloWorld.cpp

Click Add



Contents of helloWorld.cpp:

#include <iostream>

#include <string>

using namespace std;

int main()

{

string anEcho = "";

cout << "Hello! messages are coming from general.asu.edu" << endl;

cout << "Enter 'quit' to exit program" << endl;

while (anEcho != "quit") //loop until quit is entered

{

cout << "Enter a message: ";

cin >> anEcho;

cout << "Echo: " + anEcho << endl;

}

//System Pause----------------------------------------------------------------

//A way to pause a console that is compatible with nearly any system

char pause;

cout << "Enter any key to exit: ";

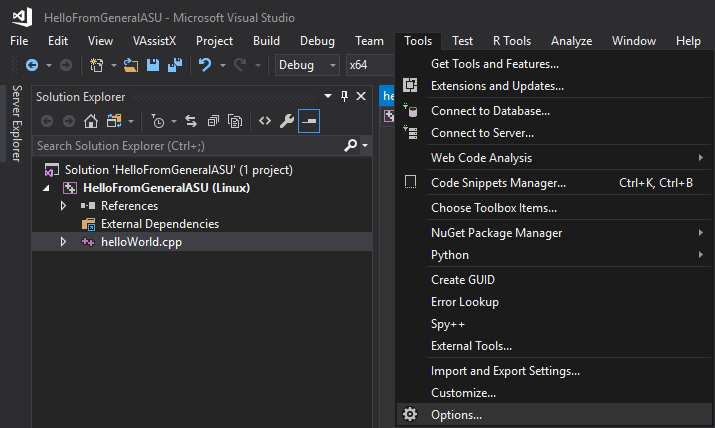
cin >> pause;

//System Pause----------------------------------------------------------------

return 0;

}

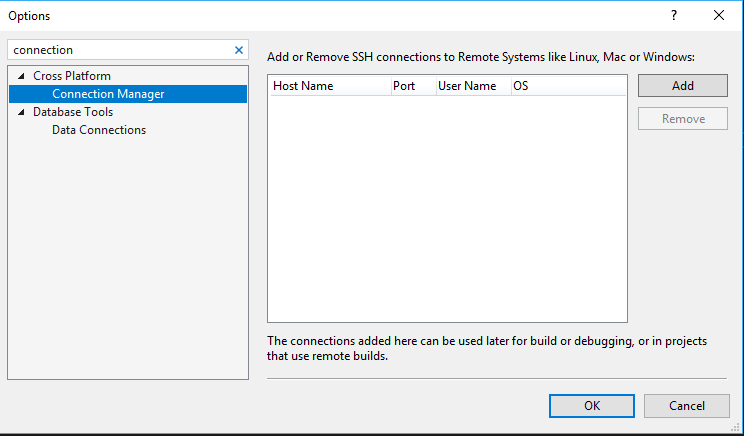
On the menu bar select tools -> options



Search for connection

Select Connection Manager

Select Add



A new window will popup

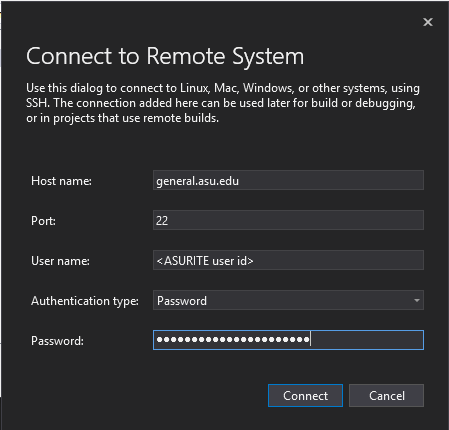
Enter the following information

Host Name: general.asu.edu

User Name: <general.asu.edu username>

Password: <Your Password>

Click Connect

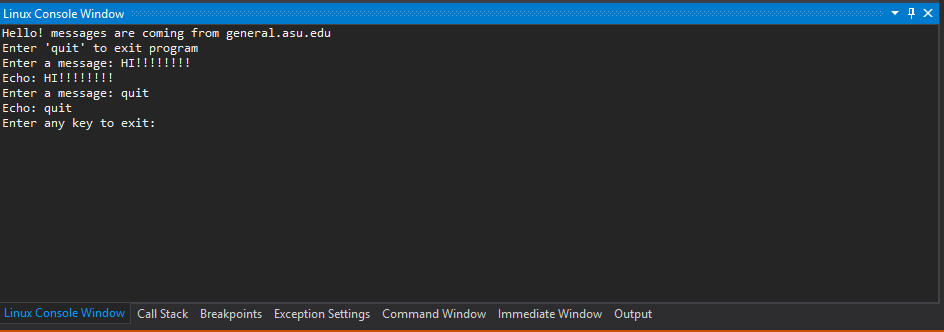


In the options window click OK

You should be able to run the debugger at this time

From the menu bar Debug -> Start Debugging OR press F5

Files will automatically get transfer to general.asu.edu. VS will automate the process of compiling, linking and creating an executable. VS will launch the executable on the server and debug it with gdb, giving you a Linux Console to interact with the program.

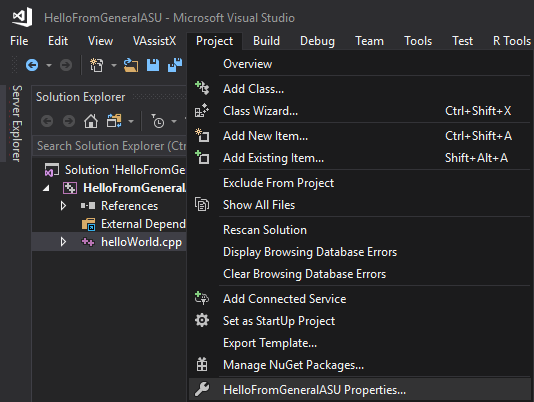


1. Configuring

At the time of writing the current GCC version on general.asu.edu is 4.8.5. In this version the default standard that g++ currently uses is c++98. You will either need to change configuration setting in visual studio to use the default standard or leave it as 2011 std. **If you choose to leave c++11 std you will need to add flag “-std=c++11” to your make file.** Warning! Be aware that the GCC 4.8.5 manual states “The 2011 ISO C++ standard plus amendments. Support for C++11 is still experimental, and may change in incompatible ways in future releases.”

To change the standard setting in VS (Optional)

On the menu bar select Project -> HelloFromGeneralASU…



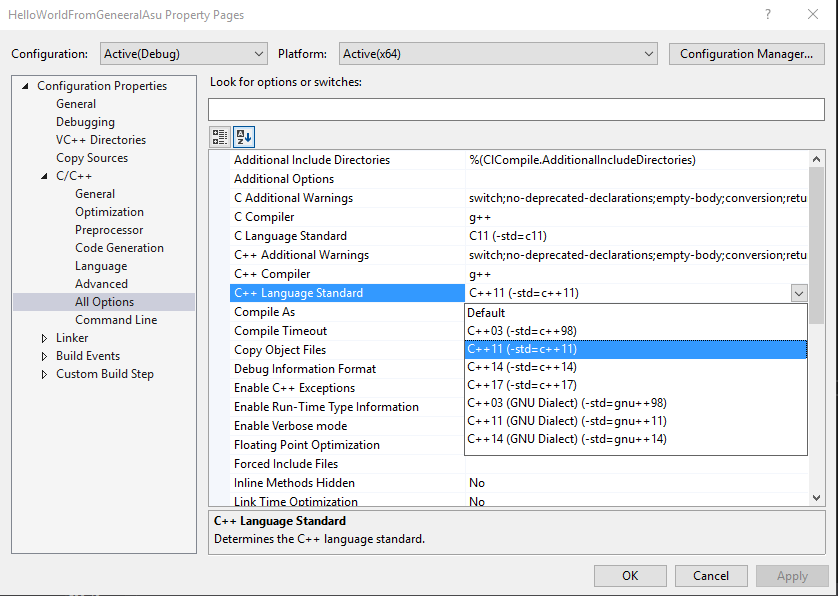
Configuration Properties -> C/C++ -> All Options

C++ Language Standard -> Drop down. Pick your preferred std and ensure your make file submission includes the same std. GCC 4.8.5’s default is standard c++98

To get more information about the current version and what standard are available the following command can be entered into general.asu.edu.

$ g++ --version

$ man g++



This is all that is needed to compile, run, and debug remotely on general.asu.edu. The rest of the guide is how to load and run inputs and command files. And possibly more to come. You can visit “https://github.com/jwardwell7077/VS-2017-LinuxRemote-general.asu.edu-doc/raw/master/GuideLinuxRemoteGenASU.docx” for the latest version of this guide.

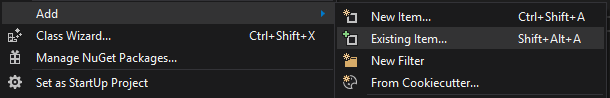
1. Extras: Loading files

Loading inputs and command files

The current release, Visual Studio Enterprise 2017 Ver 15.8.1, does not allow using file redirects “<” and “>”. When using Linux remote. In future releases this may change. Currently, if redirects are needed for testing sample inputs and outputs they will be required to run on general.asu.edu using a method other than VS remote.

VS can be used to transfer sample inputs and outputs from to general.asu.edu. Include them in the project

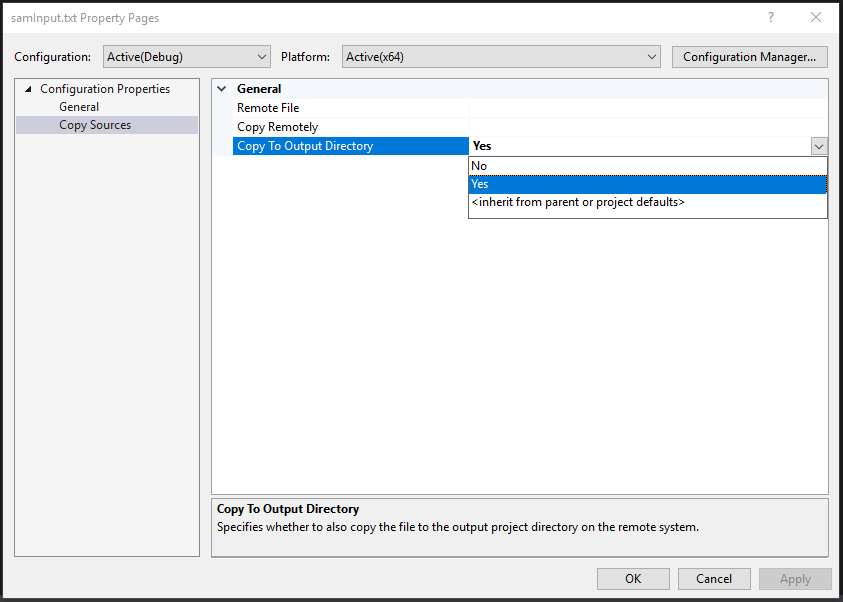
Right click in the solution explorer window-> Add -> New Item or press Ctrl + Shift + A



Copy and paste sample input and save

Another option is to add an existing item. It would be advantageous to copy the input file(s) to the local project directory then adding existing item. If you add the item from a different location the file path will be copied into the remote project.

Right click samInput.txt -> properties ->Copy Sources -> Copy To Output Directory -> Yes



Log into your preferred SSH and navigate to

$ cd ~/projects/HelloFromGeneralASU/bin/x64/Debug

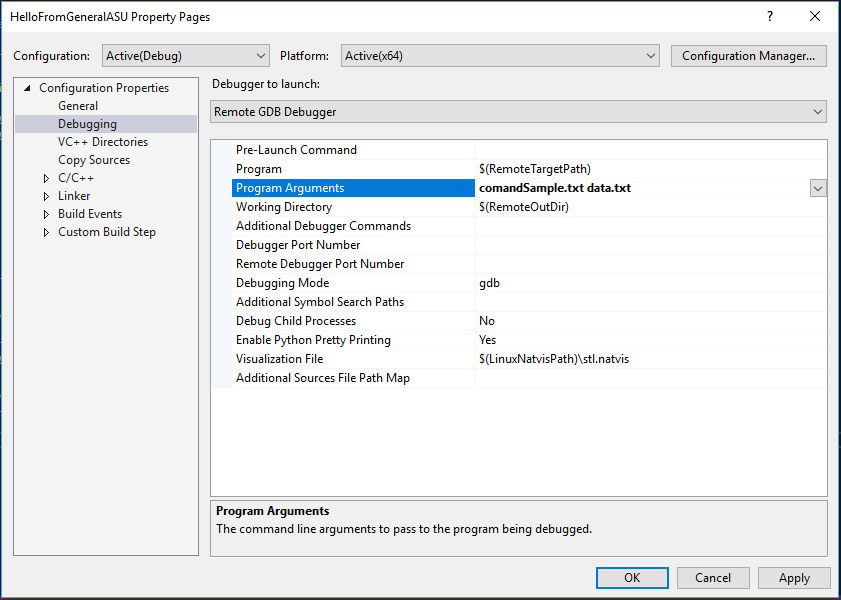
$ ./HelloFromGeneralASU.out < samInput.txt

OR

$ ./HelloFromGeneralASU.out < samInput.txt > myout.txt

Visual Studio has a way to pass arguments by command line. For use in “int main(int argc, \*argv[])”. This is use full if there are data or command files need to get read into program using fstream or there are values passed to your program at launch.

From the menu bar Debug->HelloFromGeneralASU Properties…



Sources:

<https://docs.microsoft.com/en-us/cpp/linux/>

<https://sourceware.org/gdb/current/onlinedocs/gdb/Configure-Options.html#Configure-Options>

https://unix.stackexchange.com/questions/42567/how-to-install-program-locally-without-sudo-privileges

<http://plato.la.asu.edu/MAT420/uenviron.pdf> (page 32)