

Student Linux Box – SLB – User Guide

CEReAL

Computing **E**ducation **R**esearch & **A**dult **L**earning
<http://CEReAL.forest.usf.edu/>



Disclaimer

This material is based in part upon work supported by the National Science Foundation under award number 0836863. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Table of Contents

Table of Contents	2
Introduction.....	4
1. Overview.....	4
2. Terminology.....	4
3. Required Hardware / Software.....	5
Installing Virtual Box	6
1. Downloading the Virtual Box installer file.....	6
2. Download the Virtual Box Extension Pack	7
3. Run the Virtual Box installer	8
4. Install the Virtual Box extension pack.....	8
Installing the SLB Virtual Appliance	12
1. Download the SLB virtual appliance.....	12
2. Importing the SLB virtual Appliance	12
3. Login into your virtual appliance.....	13
4. Updating Ubuntu Linux	13
5. Shutting down your virtual appliance	13
Shutdown from within Ubuntu Linux.....	13
Shutdown from virtual box	13
Hibernate from Virtual Box	14
6. Taking & Restoring Snapshots	14
Why taking snapshots?	14
How to take a snapshot?	15
How to use snapshots?	16
7. Installing the “guest additions”	16
8. Sharing folders with the host OS	17
Preparing the virtual machine for sharing.....	17
Accessing the shared folder from the guest OS	19
9. Troubleshooting	20
Screen doesn’t resize.....	20
SLB to learn C Programming	21
1. Features overview	21
2. Accessing the available study material	22
3. Working on prepared projects with the Code::Blocks IDE	22
Starting Code::Blocks	22
Compiling, Building, Running your projects.....	23
4. Developing your own projects with Code::Blocks IDE	23
Preparing a new project	23
Compiling and Executing a project.....	30
5. Debugging with Code::Blocks IDE	31

Preparing your program to be ran with the debugger	31
Setting up “BreakPoints”	32
Stepping through your programs	34
Inspecting Variables Values on the go.....	37
6. Using “NED” to track down novice errors	41
Using NED from Code::Blocks	42
Using NED from a browser	42
Using the reports	43
What does it mean to validate your program with NED?.....	45
7. Validating your tests & solutions.....	45
Validating your solution against the instructor’s tests	45
Validating your solution against the instructor’s tests	45
8. Preparing your submissions	46
9. Other tools.....	46

Introduction

1. Overview

This guide is intended for students who plan on using the Linux virtual appliance we are making available to support learning of specific topics in our Information Technology Department. It will provide you with instructions on how to install the necessary software, deploy the virtual appliance & use the tools we have pre-installed on it for you.

2. Terminology

Virtualization tools uses a few key terms which we will define before we begin our installation.

- **The Host Operating System** (Host OS) is the operating system which is installed on your physical machine. You will need a recent version of Linux, Windows or MacOSX which is able to run the virtualization software specified below.
- **The virtualization software**, also referred to as hypervisor, is the software which runs on your host operating system and simulates a virtual machine's hardware. The specific virtualization software we will be using is the open source Virtual Box (<https://www.virtualbox.org/>).
- **The Guest Operating System** (Guest OS) is the operating system that runs inside the virtual machine. We use Ubuntu Linux as the operating system running the Linux virtual appliance you will be using.
- **The virtual appliance** is a package containing all you need to install a guest OS in your virtual box software. Virtual appliances are downloaded and imported into compatible virtualization software by end users regardless of their host OS. They are then operated, often without the need for the users to learn about system administration tasks.
- **The virtual machine** is the simulated computer in which the guest OS runs. We will often use the terms virtual machine & virtual appliance interchangeably since the latter is really a virtual machine meant to be distributed
- **SLB** stands for "Student Linux Box", it is the name of the virtual appliance we are providing to our students to work. By providing a pre-made work environment we guarantee that, regardless of their preference in terms of hardware or operating system, every student will have access to the same tools to help them with their work & will submit equivalent deliverables for evaluation.

To summarize and leverage the jargon we just learned; this guide will guide you through installing the SLB virtual appliance. This will entail installing the virtualization

software on your host OS, downloading the SLB virtual appliance, importing it in your virtualization software, & starting it.

3. Required Hardware / Software

The SLB virtual appliance may be hosted on any machine able to run the Virtual Box software. We therefore recommend you check the requirements on <http://www.virtualbox.org/> first.

You should be able to install and use it on Windows, Linux or MacOSX platforms. We recommend a machine with at least an Intel Core I(x) processor or Dual Core processor, about 4GB of free memory, and 20GB of free disk space.

You will need your machine to be on the internet while using the SLB virtual appliance. Your machine doesn't need to be solely dedicated to running the SLB virtual appliance; you may boot SLB, use it then shut it down as needed.

The installation steps detailed in the remainder of this guide will require you to have "administrator" access, if you are installing on a Windows machine, or "root" access if you are using a Linux machine. Without such privileges you will be unable to install and setup the Virtual Box software.

When the SLB virtual appliance has been successfully installed on your machine, it will operate without requiring further administrator or root privileges from you.

Installing Virtual Box

This section walks through downloading, installing & setting up the virtual box software.

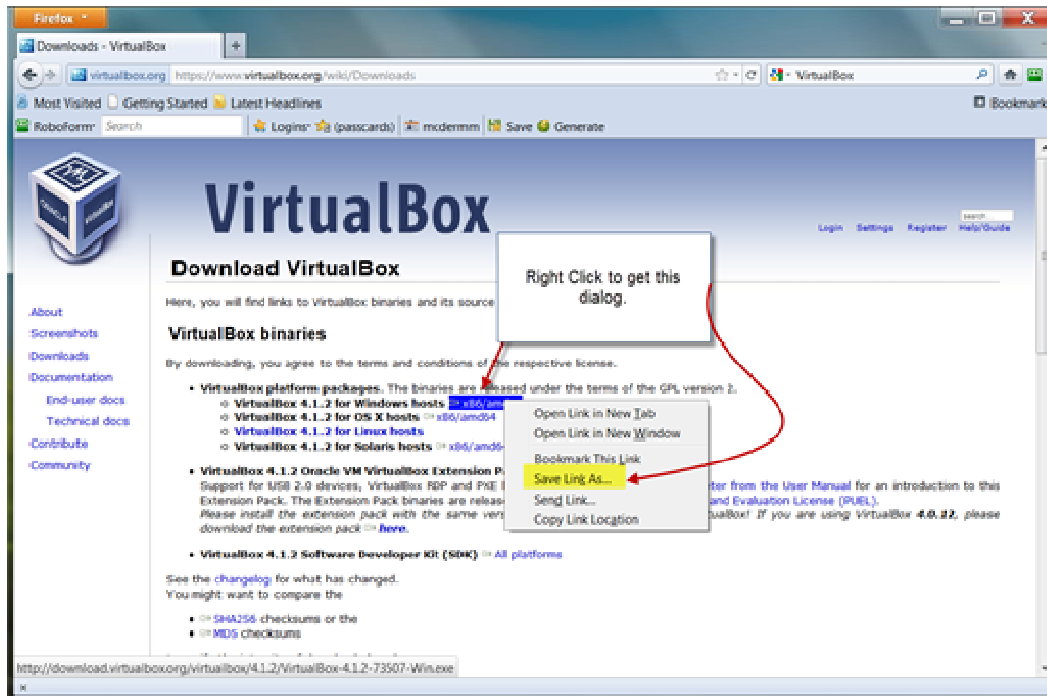
1. Downloading the Virtual Box installer file

The Virtual Box installer is available from the virtual box official web site

- Open your preferred web browser
- Go to the following URL <http://www.virtualbox.org/>
- A link named **Downloads** should be visible from the site's front page. See example illustration below.
- Please note that the version numbers will be more recent than those portrayed in the screenshots below.



- Download the Virtual Box installer by using the link which matches the host OS of the machine you plan to dedicate to running the CLUE virtual appliance. See example illustration below.



After following these steps you should have the installer on your disk as an executable file. Make a note of the location where it was downloaded since we will need to find it and execute it later.

2. Download the Virtual Box Extension Pack

In addition to the Virtual Box installer, we are going to download the “Oracle VM virtualbox Extension Pack” which will provide additional features facilitating the integration of the virtual machines running inside Virtual Box with our host OS.

Use the **All platforms** link to download the extension pack



Make a note of where the resulting file was downloaded.

3. Run the Virtual Box installer

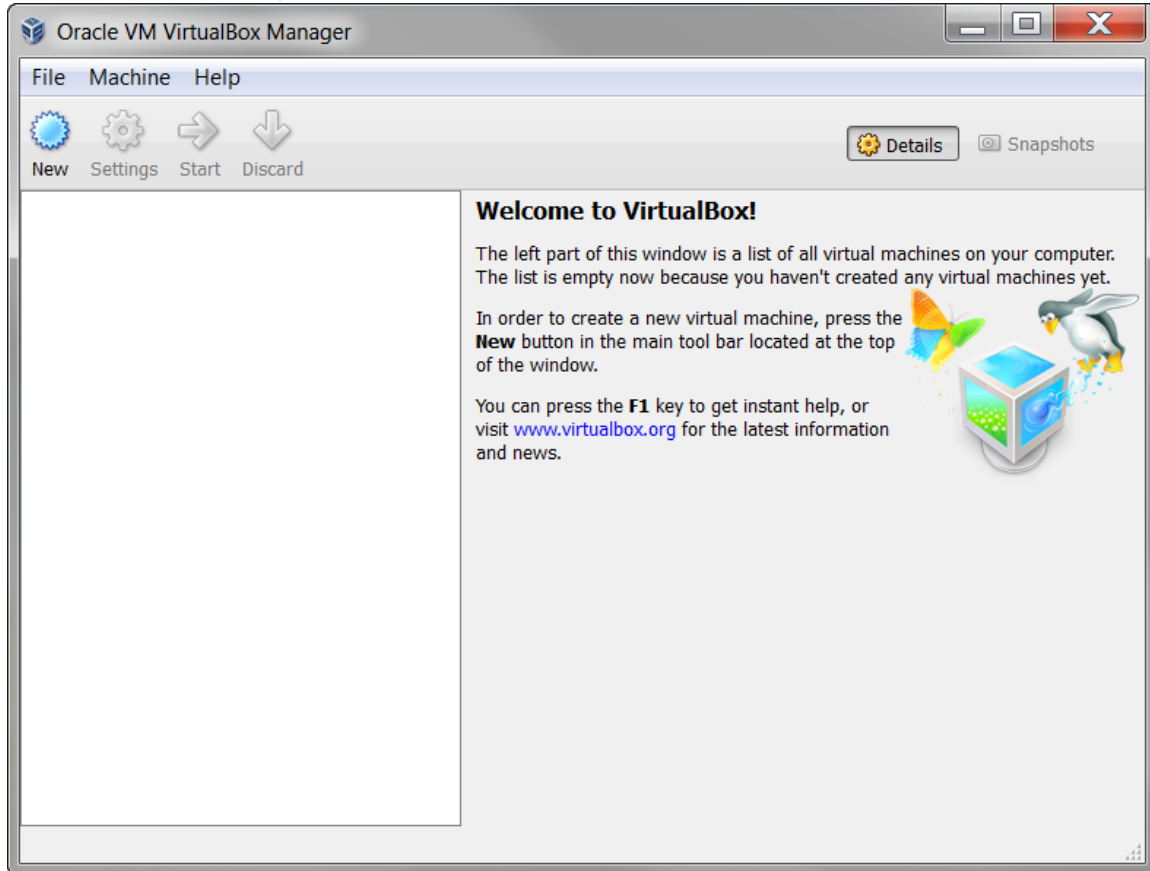
We are going to install the Virtual Box virtualization software on our host OS.

- Locate the Virtual Box installer file you downloaded in the previous section. Open a file navigation window on your host OS and double click the file to execute it. The Virtual Box's setup wizard will start.
- We will accept the default options until the last screens. During this setup additional network adapters will be added to your host OS. The whole process will take some time.
- When the installation is finished, you will be offered the option to run Virtual Box. Do not run it yet, we still have to install the extensions
- You will then be prompted to reboot your machine, always do so. If you do not, the next installation steps will not work properly.

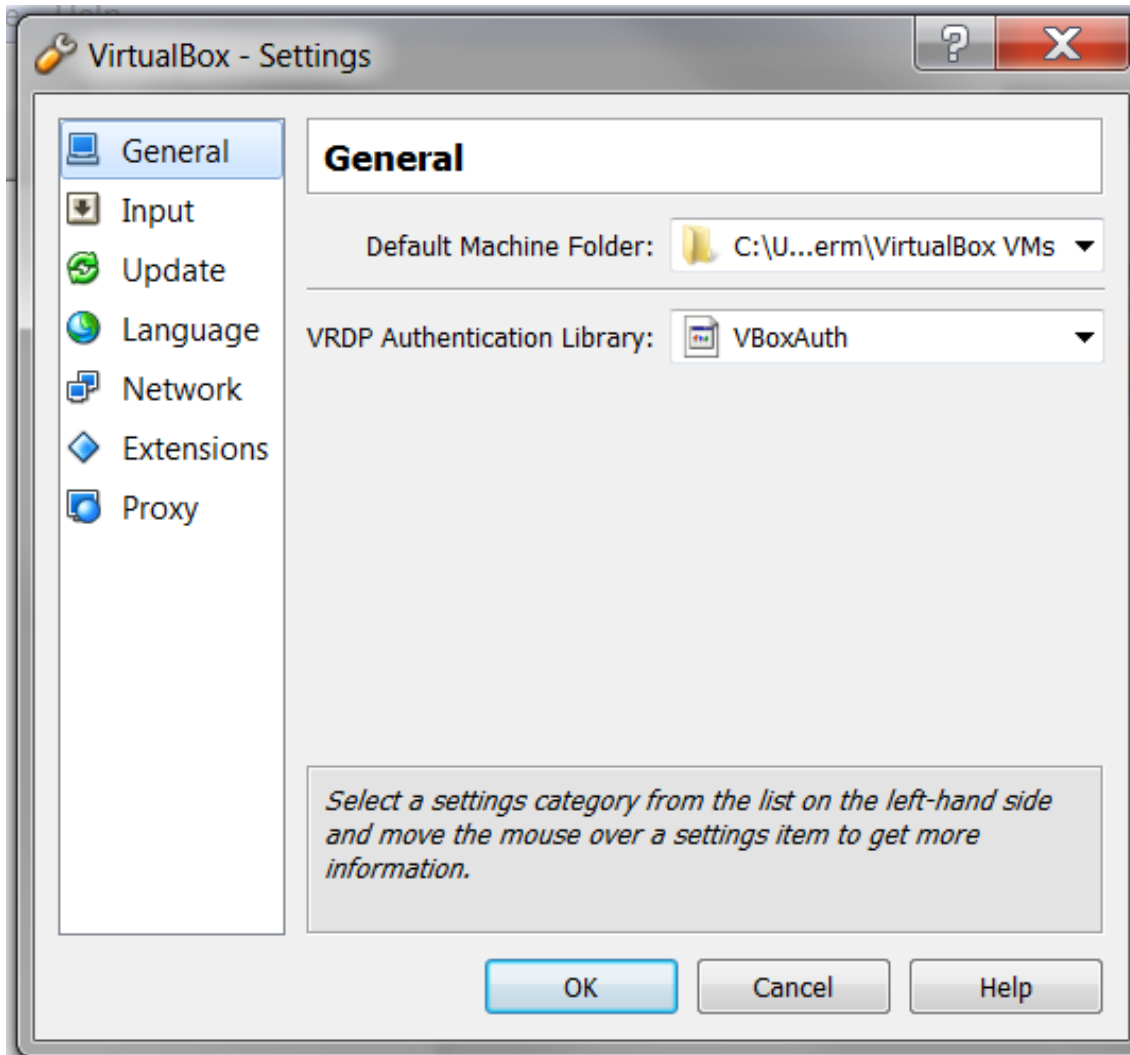
4. Install the Virtual Box extension pack


Virtual Box Extensions are add-ons that provide the capacity to share clipboards between the host and guest OS, share printing, and limited network sharing. These elements are not required to operate a virtual machine; however, these features make it easier to interact with the CLUE Virtual appliance.

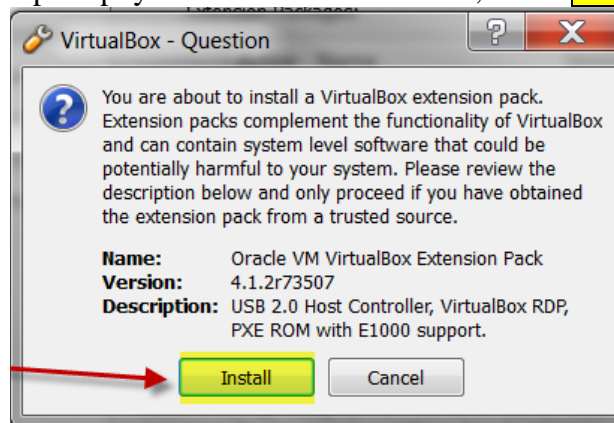
- Start the Virtual Box software.
- The Virtual Box main interface will open. It should be very similar to the one in the screenshot below;



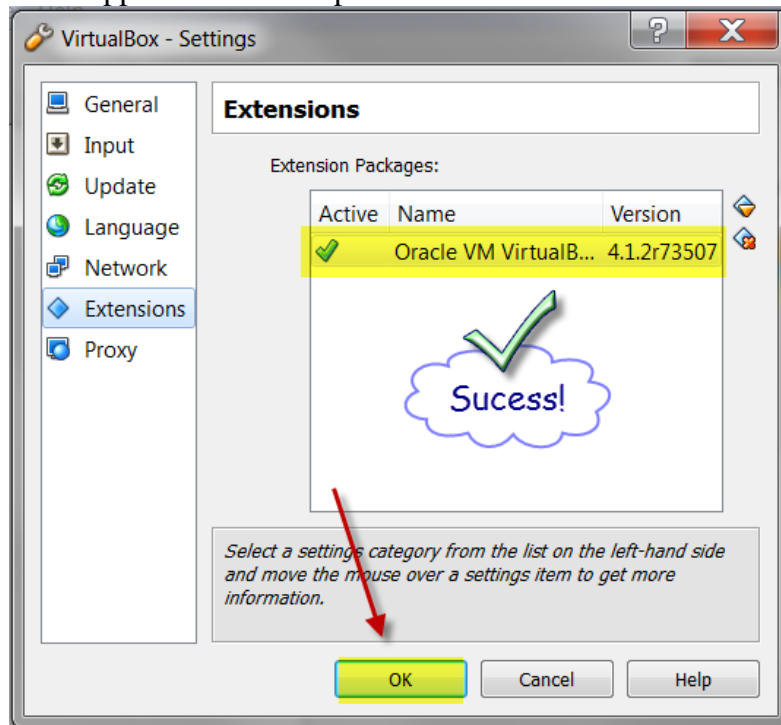
- Go in the **File** menu, select the **Preferences** option. The following panel will open.



- Select the **Extensions** option from the left panel. The main panel will show an empty list of installed extensions
- Locate the following icon to the right of the list. 
- A file selection dialog box will open. Navigate your file system to select the extension file you downloaded in previous section
- A dialog box will prompt you to start the installation, select **Install**.



- You will have to read and agree to the license agreements by selecting **I Agree**
- A dialog box will inform you of the successful installation of the extension pack which will then appear in the main panel's list



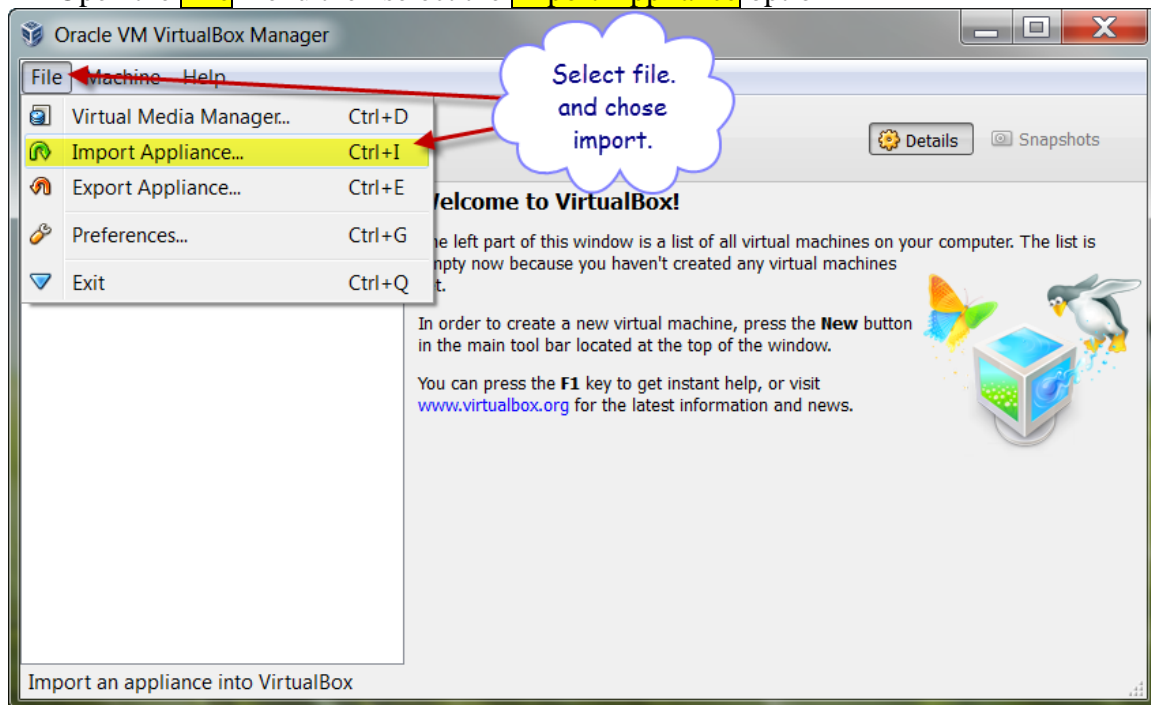
Installing the SLB Virtual Appliance

1. Download the SLB virtual appliance

This virtual appliance is available as a package which Virtual Box will be able to manage for you via its “import” feature. The file is available from the CLUE project’s web site; <http://CEReAL.forest.usf.edu/slb/>

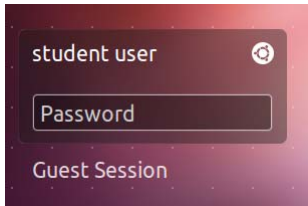
2. Importing the SLB virtual Appliance

- Start the Virtual Box software if it is not still opened on your desktop
- Open the **File** menu then select the **Import Appliance** option



- The import wizard window will open allow you to navigate your file system to select the SLB virtual appliance file you previously downloaded
- The setup of the virtual appliance will then be displayed; select **Import** to proceed with installing it in Virtual Box.
- The import should take a few minutes after which the Virtual Box main window will show your new virtual machine listed in the left panel.
- You are now ready to start the SLB virtual appliance

3. Login into your virtual appliance



As you start the virtual appliance, you will be shortly prompted to login with a specific account. Simply click on the **Student user** account to begin the login process.

You will now be asked for your password which is the same as your login name; **student**.

4. Updating Ubuntu Linux

As you work with your Ubuntu Linux environment, you will occasionally be prompted to update software. Feel free to update any & all of the available software but make sure to take a snapshot of the virtual machine before hand – see section on snapshots below.

When you update software, you will be prompted for the administrative password. It is the same as your login password.

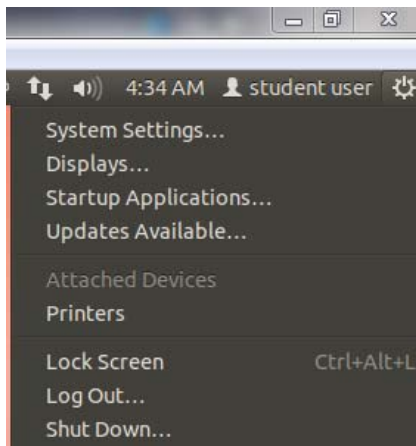
The process will require you to be on the internet for the entire duration of the update.

5. Shutting down your virtual appliance

When you are done working with your virtual appliance, you should shut it down. There are different ways to shut it down.

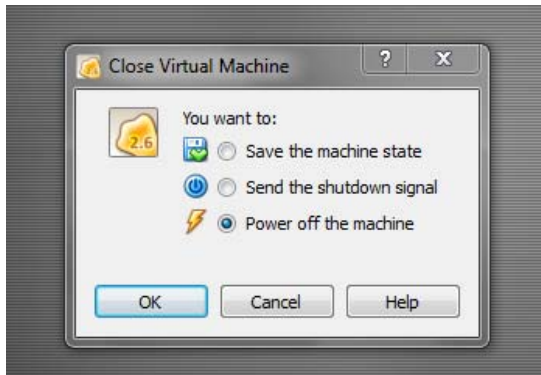
Shutdown from within Ubuntu Linux

Use the gear icon located at the top-right corner of your Linux desktop. When you click it, you will be offered several options, select **Shut Down...**.



Shutdown from virtual box

If you attempt to close the window in which your virtual machine runs, you will be offered 3 options;



The **Send the shutdown signal** option will allow you to let the virtual machine handle a proper shutdown. The **Power off the machine** option will shutdown the virtual machine without giving the guest OS an opportunity to handle it. Save this option for a situation where the guest OS is no longer responding as it might result in damage to the files on your virtual disk.

Hibernate from Virtual Box

The **Save the machine state** option is equivalent to hibernating. The machine will save its state before to shut down. When you restart it, it will resume operation exactly where you left it.

6. Taking & Restoring Snapshots

While it is not our intent to turn this guide into a virtual box user manual, being able to make snapshots goes a long way in making your life easier.

Why taking snapshots?

A snapshot is a way to tell virtual box to remember the exact settings of your guest OS so you may revert back to it later.

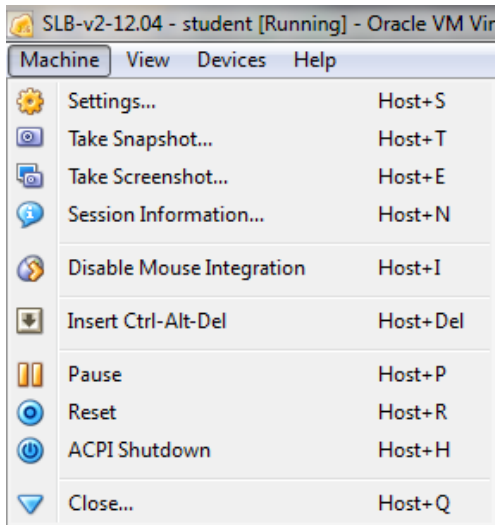
It is a good idea to take a snapshot before any operation which might potentially “break” the virtual machine. The most notable example is upgrading the system. You should be able to update any software package on the system without problems. However, if you update your kernel to a newer version, the virtual box guest addons will need to be re-installed in order to allow you to resize dynamically your screen. Installing the new Guest addons will require you to update your version of virtual box to the latest available then re-perform their installation.

Sometimes, there is a delay between the release of a new Linux kernel & the release of new addons. During this time, you won't be able to resize your screen.

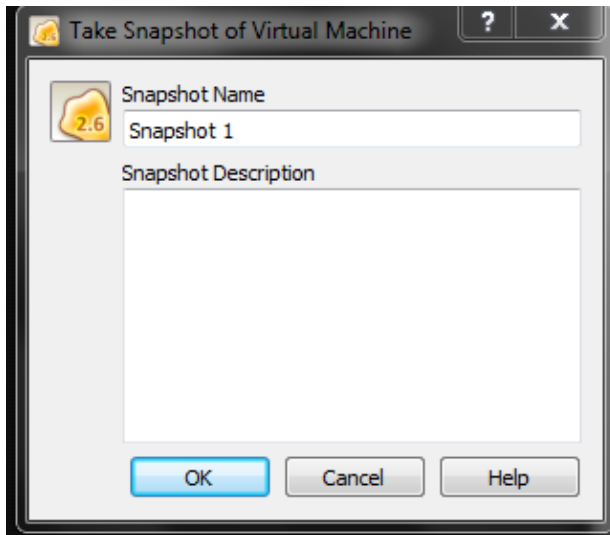
We recommend that you snapshot the virtual machine before any update, so you may easily revert back to it if the result is not to your likings.

How to take a snapshot?

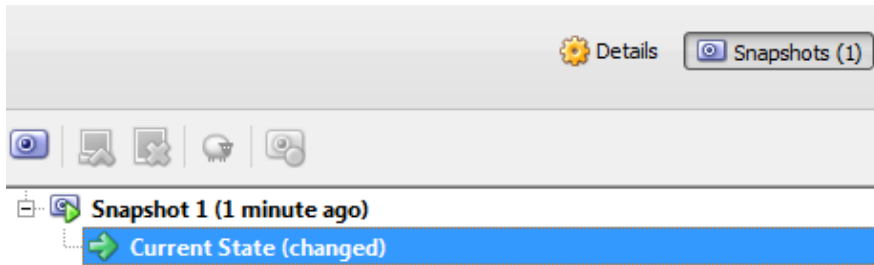
Go to the **Machine** menu of your virtual machine's window & select the **Take Snapshot** option.



By default the system will give a name to your snapshot based on how many previous snapshots you already took. We suggest you keep it but add some notes to the **Snapshot Description** field to help you remember what it is for; e.g. "Before update".

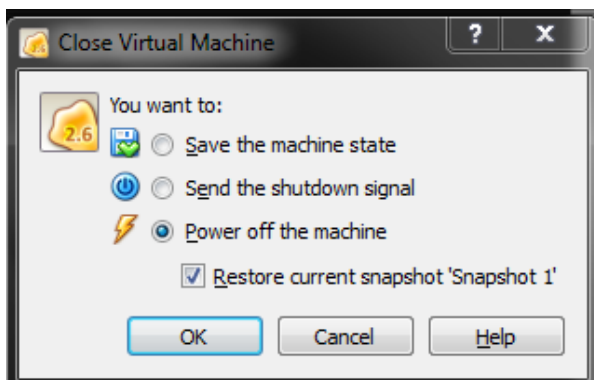


Your snapshot will now be visible in the virtual box main window when you select your virtual machine & press the **Snapshots** button in the top-right area of the window;



How to use snapshots?

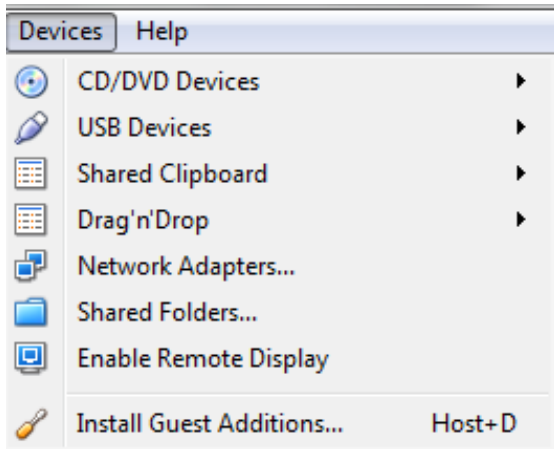
When you try to close the virtual machine window, the **Power off the machine** option will now allow you to reset it to a specific snapshot thus allowing you to revert to a previous state.



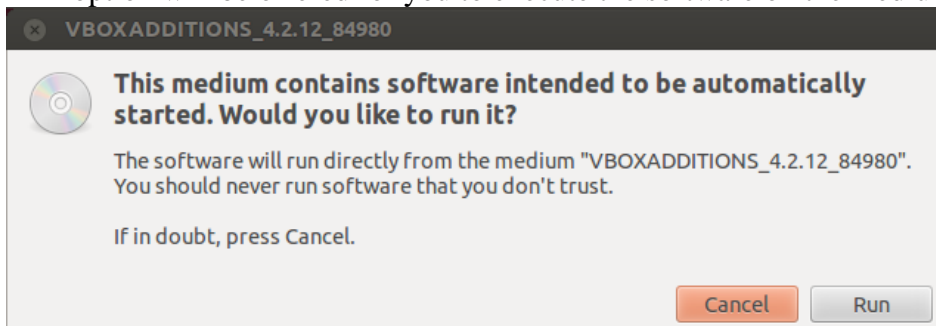
7. Installing the “guest additions”

In order to benefit from the full set of virtual box features, you need to install software in your guest OS allowing it to communicate efficiently with virtual box. This is necessary for some of the features we will be using such as sharing folders between the guest & host OS. Before to do the following, we suggest you take a snapshot of your virtual machine.

The **Devices** menu of the virtual machine’s window has an option to **Install Guest Additions**. Selecting this option will insert a virtual CD which the Linux virtual machine will detect.



An option will be offered for you to execute the software on the medium inside Ubuntu.



Running the software will open another dialog which will prompt you to provide the administrator password, type in your user password. You will then see a text window detailing what is being installed. The installation should take a few minutes after which you will be instructed to “Press return to close this window”. Do so then shutdown your virtual machine. When you boot it again, you should be able to see the guest OS adjust its resolution every time you resize the virtual machine’s window. This will also enable your virtual machine to share a folder with the host OS as will be detailed below.

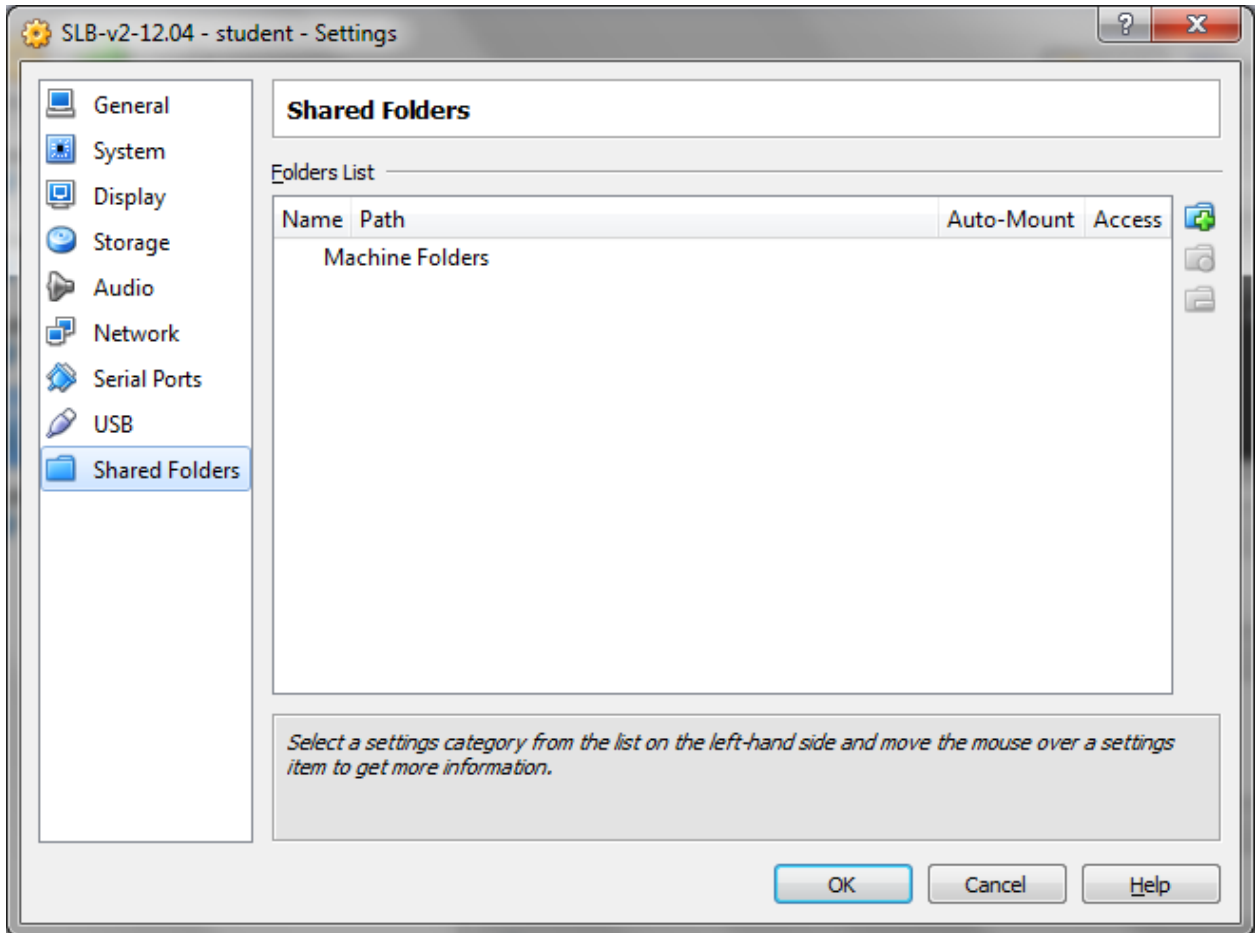
8. Sharing folders with the host OS

It is often useful to be able to share files between your guest & host OS. The virtual box extension pack allows you to mount inside your SLB a folder from the host. We will detail how to do so for Windows, please refer to the Virtual Box website for more details;

<http://www.virtualbox.org/manual/ch04.html#sharedfolders>

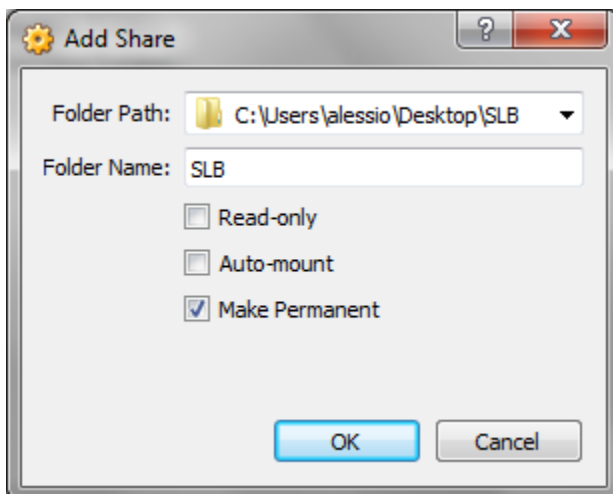
Preparing the virtual machine for sharing

Shutdown your virtual machine, then open its settings. You need to go to the **Shared Folders** section of the settings dialog box.

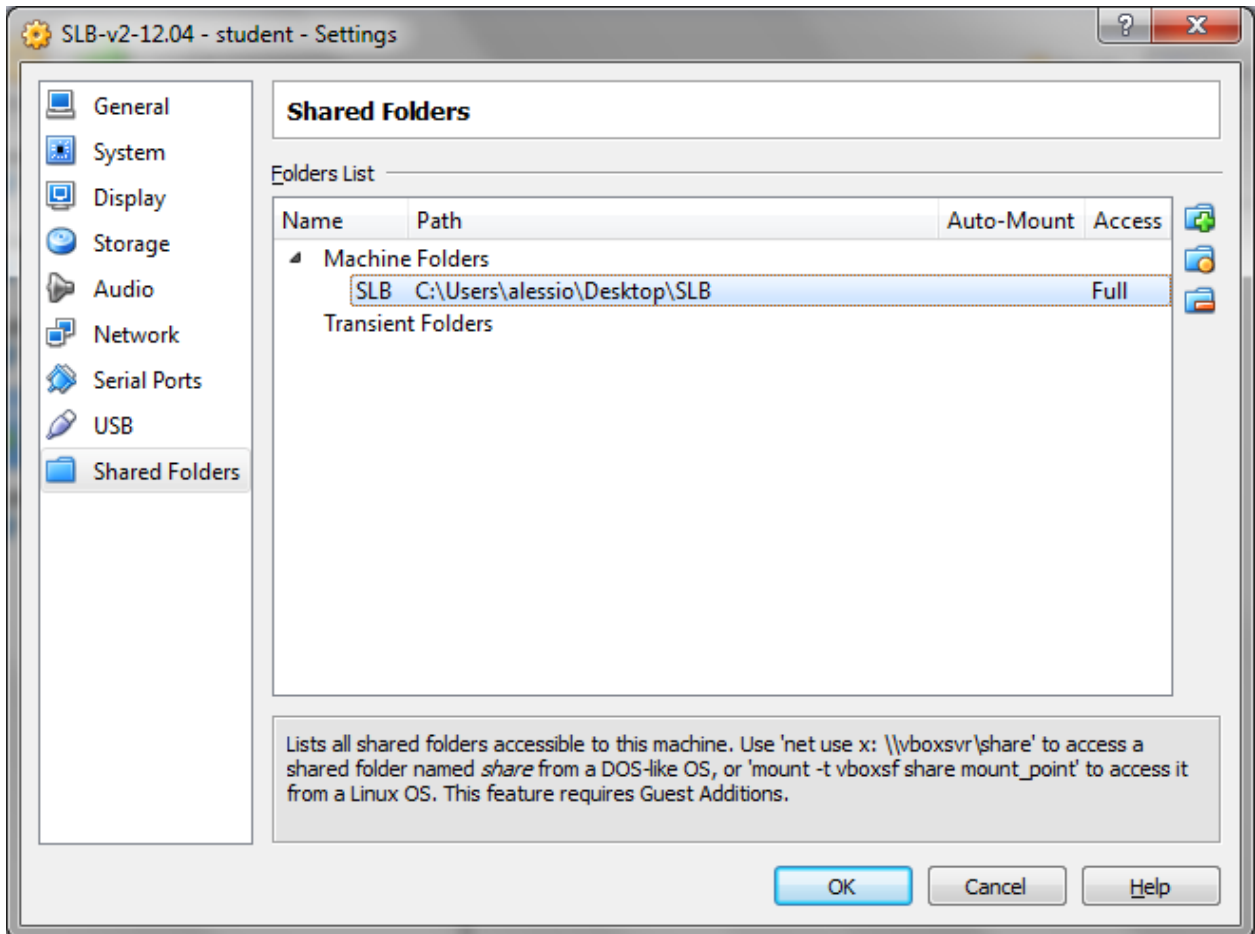


There, you will be able to use the **folder +** icon on the right of the empty list to add a new shared folder. Make sure the shared folder exists on the host. For this example, we created a “SLB” folder on the desktop of the Windows host.

The dialog box will allow you to browse to locate the folder you want to share from the host & assign it a name. Make sure to check the **Auto-mount** option to allow for this folder to be made available to your virtual machine every time you boot it.



You should now see the new shared folder in the list.



Close the settings dialog box by using the **OK** button & restart your virtual machine. The folder will be mounted automatically.

Accessing the shared folder from the guest OS

Because you need administrator access to copy files to it, we suggest you open a file browser in Ubuntu with such privileges when you have to exchange files.



This may be done by opening a terminal window, using the terminal icon on the left launcher bar. In the new window, type the following; **gksudo nautilus &**

You will be prompted to enter your password. A new file explorer window will then open which will be able to enter the shared folder. In the Ubuntu Linux virtual machine, this folder will be accessible under the name **/media/sf_SLB/** which you may reach by

- Selecting **File System** on the left panel
- Selecting **media** folder
- Selecting **sf_SLB** folder

9. Troubleshooting

Screen doesn't resize

If you resize the virtual box window, the guest OS should adapt immediately the resolution to fit available screen real estate. If it doesn't, modify its size to force a refresh; e.g. maximize the window.

This is likely to happen if you start the virtual machine with a maximized window, if you resume it from hibernation, or if you are on the login screen. For the latter situation, you will have to wait to be logged in to be able to modify the resolution.

SLB to learn C Programming

In this section, we will walk through using the Student Linux Box features which are relevant to “Program Design”.

1. Features overview

The initial motivation for this virtual appliance was to make easily accessible to our students a few tools specifically meant to simplify the learning of the C programming language. For more details, please refer to the project’s home pages;

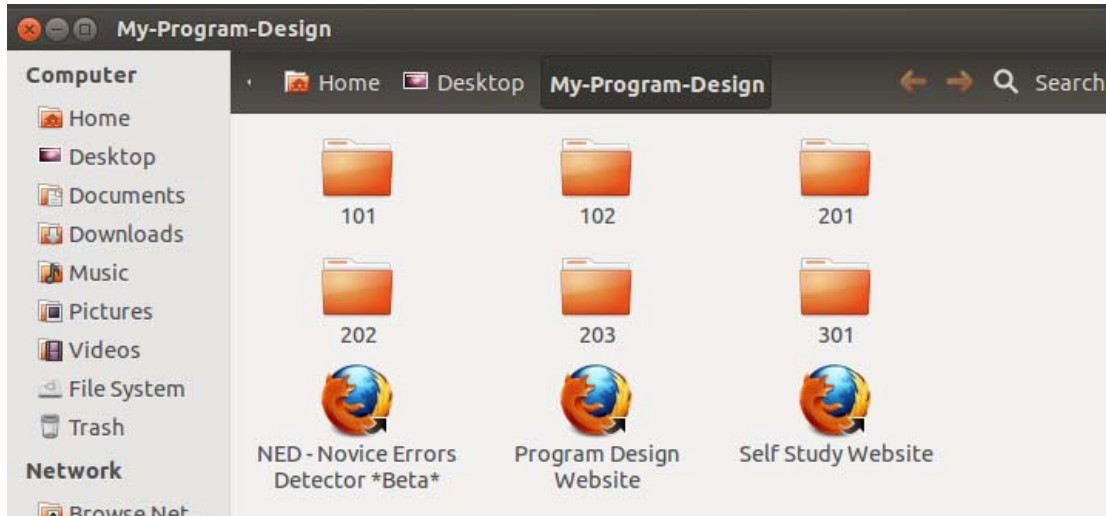
- <http://CEReAL.forest.usf.edu/clue/>
- <http://clue-ide.sourceforge.net/>
- <http://sourceforge.net/projects/clue-ide/>

As this virtual appliance was initially developed to improve support for the learning of this programming language, most of the available features to date are as follows;

- **The Code::Blocks IDE** which we have been using for years with Program Design students. It is simpler to use than other industry-standard IDEs so it doesn’t get in the way of your learning while still providing relevant features; syntax coloring, debugger, build...
- Integration in the IDE of the “**NED**” **Novice Error Detector** web application which allows you to run your programs through a suite of tools designed to detected potential errors.
- Integration in the IDE of tools to validate your work on specific “**Programming Assignments**” we use with our students. For each “PA”, you will be able to run directly from the IDE your solution against the tests we provided to get you started, or your tests against our reference implementation.
- Integration in the IDE of a “**prepare submission**” **tool** which will not only store your work in an archive ready to submit to your instructor, but also package along with it the entire history of your work on the assignment.
- Desktop links to the **educational material** used to teach the semester-long version of IT Program Design along with a 3-week long version of the material meant for advanced students’ self-study.
- A plethora of Linux development tools which might be used by your instructor to teach you more advanced topics; valgrind, electric fence, subversion, git, bsr, make... Relevant IDE plugins have also been pre-installed.
- The following sections will discuss how to leverage each of these features

2. Accessing the available study material

You will find a folder named **My-Program-Design** on your desktop. Double clicking it will open the file explorer.



Various programming assignments are available, each in their own folder. In addition, links to the following resources will open a Firefox browser when double clicked;

- A link to self-study modules; <http://CEReAL.forest.usf.edu/clue/self-study/>
- A link to the full semester version of the above;
<http://CEReAL.forest.usf.edu/clue/progdesign/>
- A link to the Novice Error Detection web application which is also integrated in your IDE; <http://CEReAL.forest.usf.edu/clue/ned/>

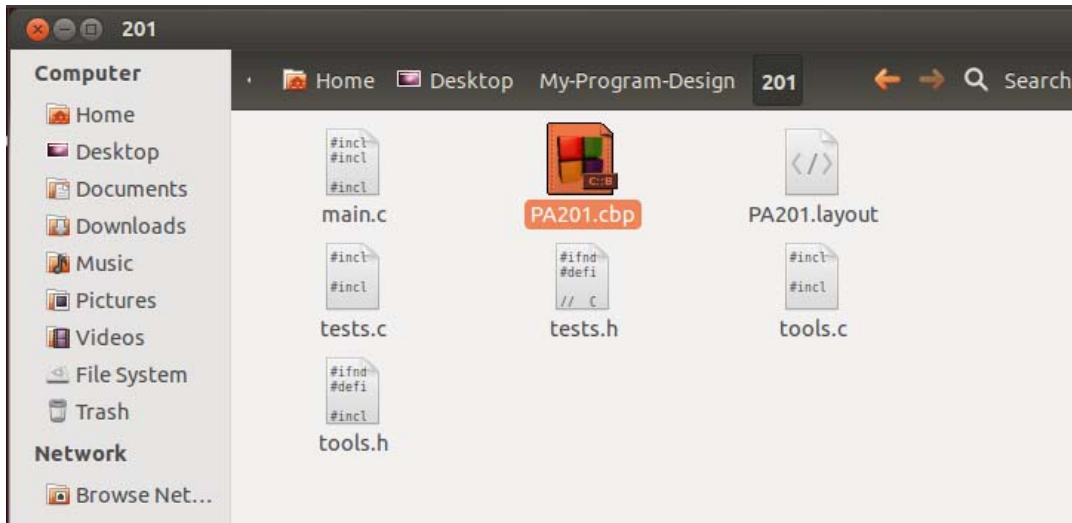
3. Working on prepared projects with the Code::Blocks IDE

The Code::Blocks IDE is provided with this virtual appliance. This section provides you with an overview of how to use it. Refer to your instructor or the online documentation for more details.

Starting Code::Blocks



You may open it by using its icon on the left-panel launcher.
You may also open it by double-clicking any .cbp file in a Programming assignment folder;



Compiling, Building, Running your projects

Open any of the PAs provided in the **My-Program-Design** folder on your desktop.

In order to run your program, you must build it. This entails compiling each of the .c source files which are part of your project, linking them together along with any needed library with a tool named the linker.

- We recommend you compile .c files individually by double-clicking them in the left-panel list, then using the menu **Build** → **Compile current file**. This allows you to see the error & warnings reported by the compiler in the log window at the bottom. Make sure you address all the warnings / errors for each file before to go to the next step.
- When your individual files are compiling, use the **Build** → **Build** menu option to generate the executable file. You might find some more warnings or errors during this process. Address them all.
- When your project is built, you may run it with **Build** → **Run**.

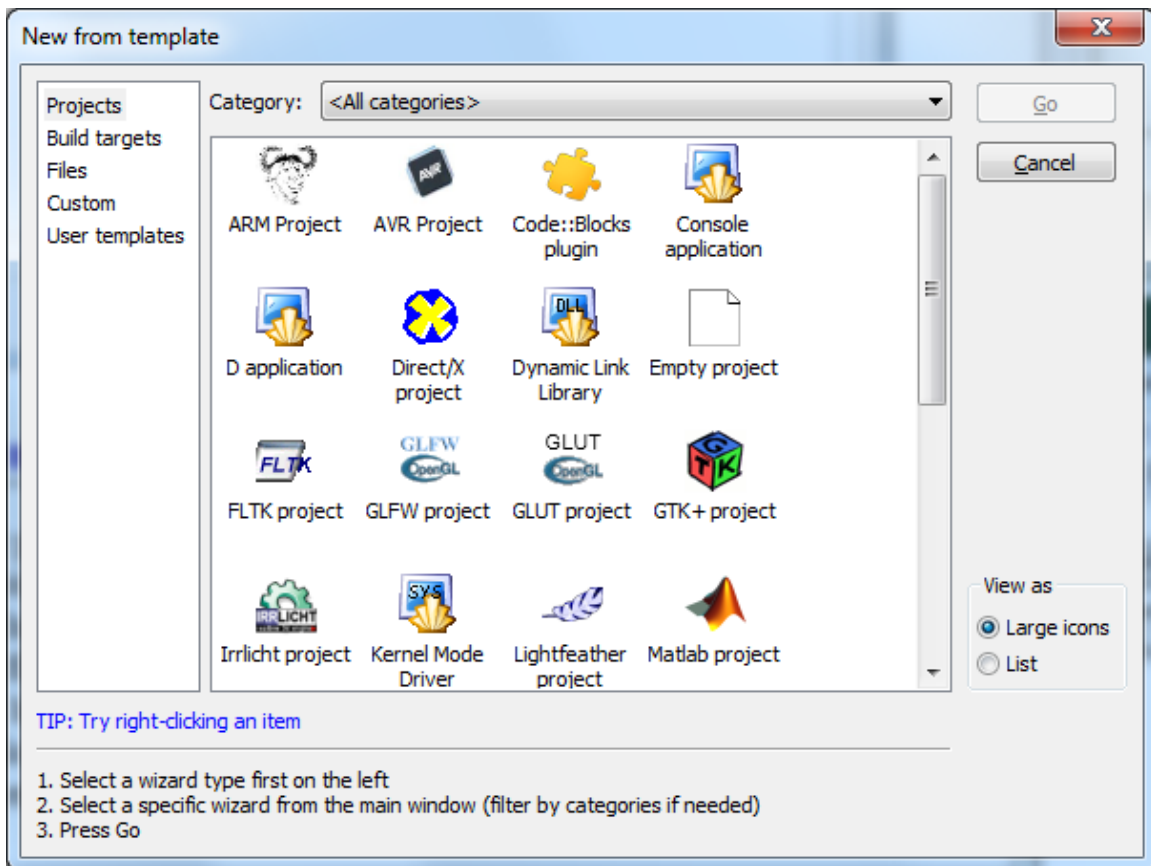
The interaction with your program will take place in a text window which will prompt you to press any key before to close when your program is done executing.

4. Developing your own projects with Code::Blocks IDE

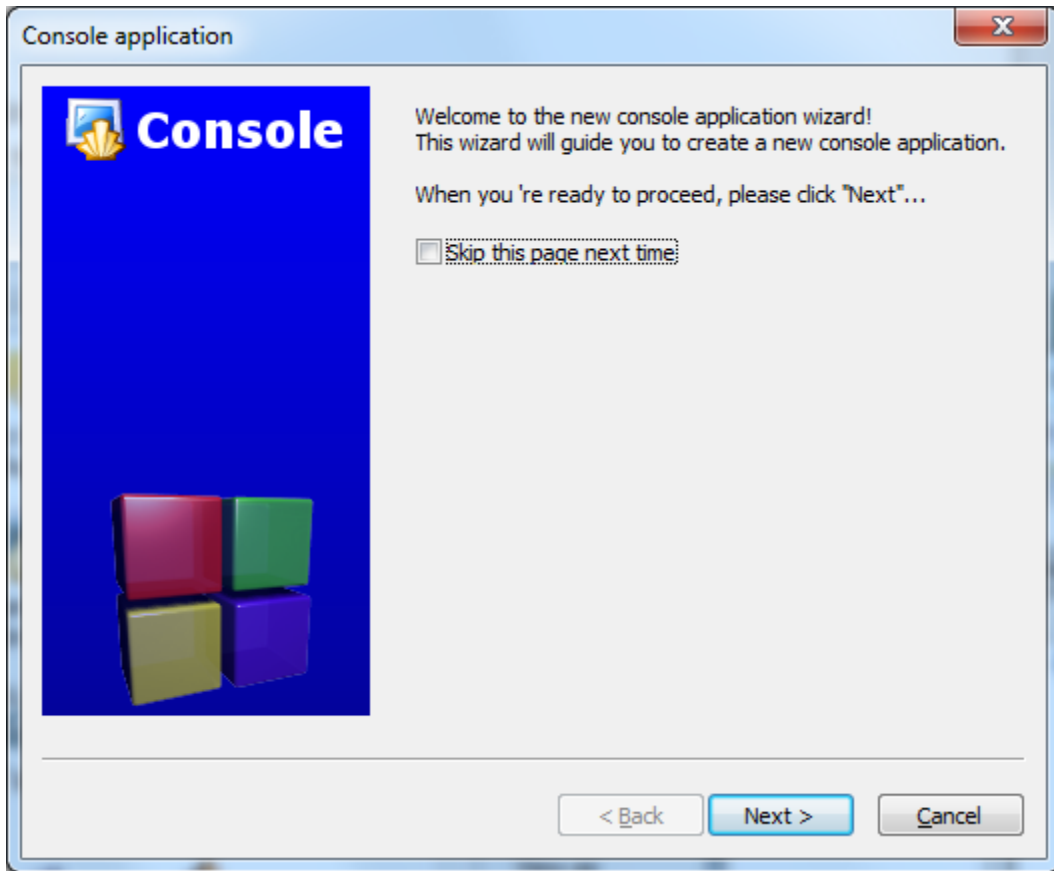
In addition to working on the projects already provided to you in this SLB, you might want to implement from scratch new projects to learn by “playing” with the language or simply to work on practice exercises. This section guides you through such a project.

Preparing a new project

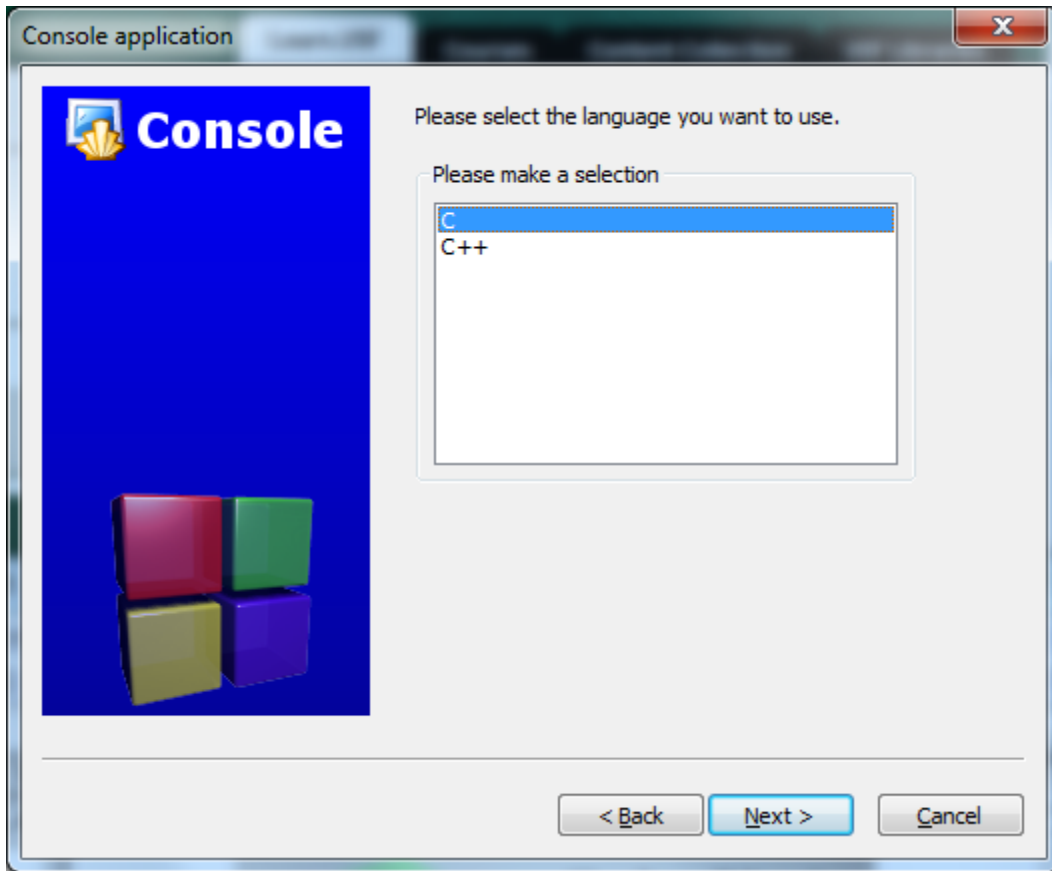
Let us prepare a new project by navigating to the Code::Block menu **File** → **New** → **Project**. This will open the project pane illustrated in the screenshot below.



We will be always selecting **Console Application** in this offering.

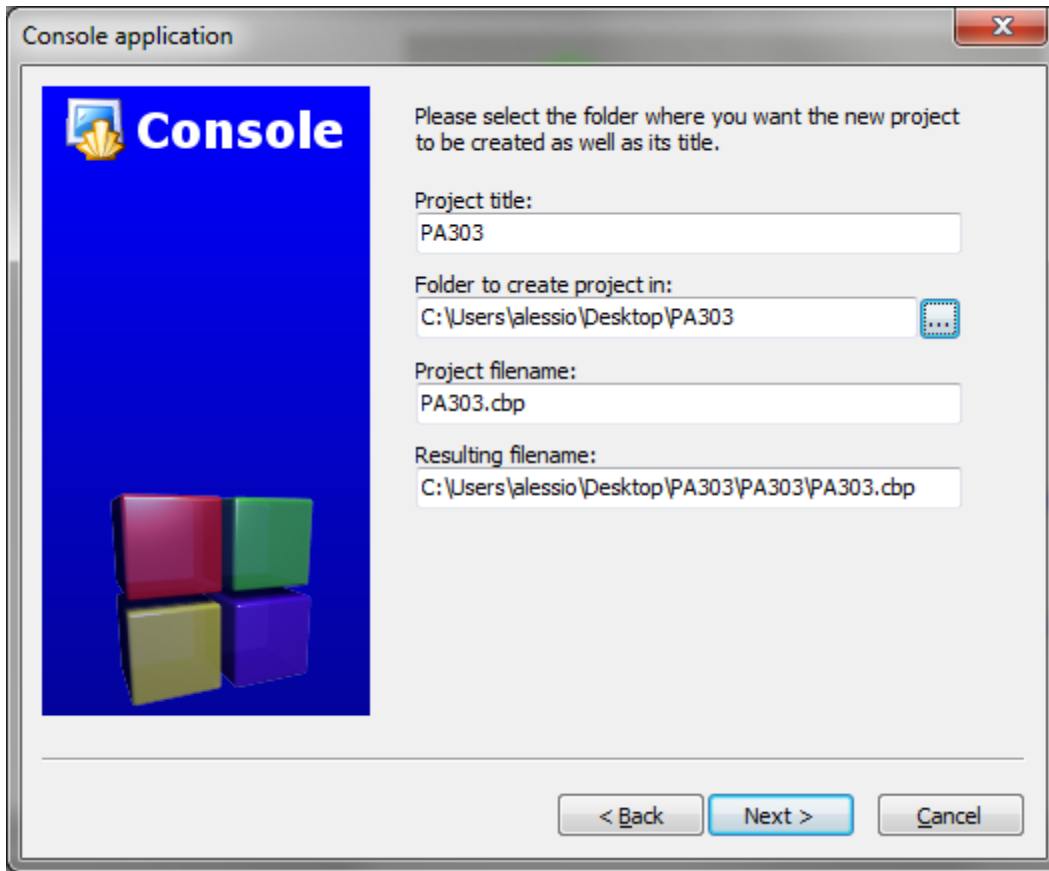


Go for **Next** and **Skip this page next time** if you want to.

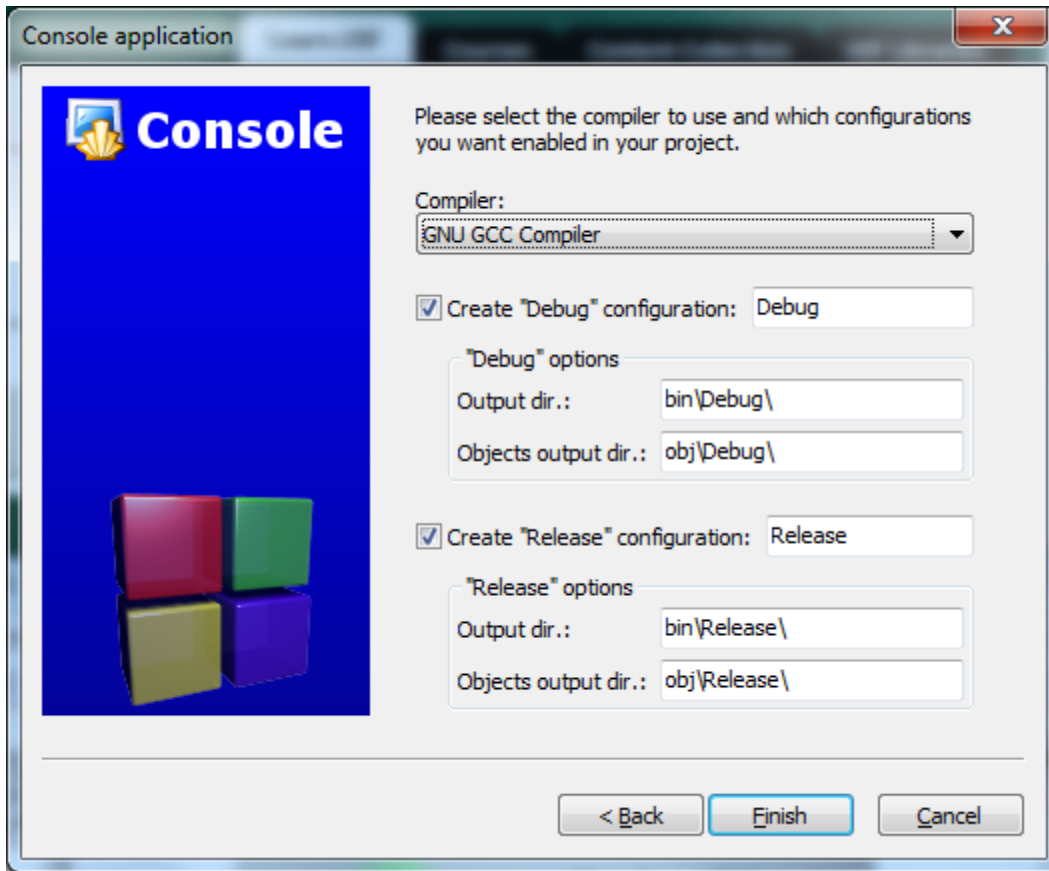


We are going to write C programs in this offering so the selection is fairly obvious.

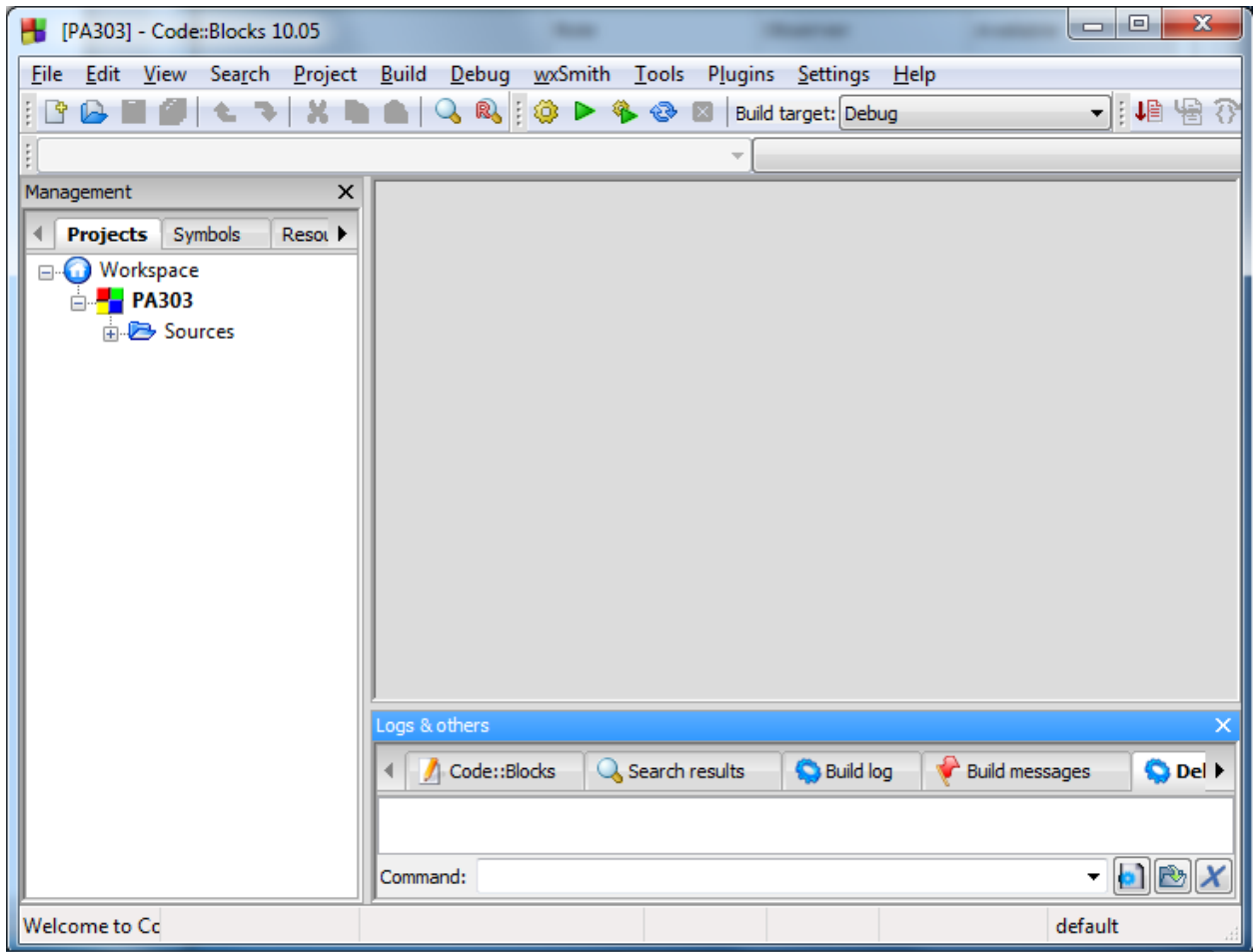
In the next panel, you will have to provide a project title. The name is up to you. In this example we'll use the name of a PA e.g. **PA303**. I'm selecting to have the project folder on my desktop. Please note that in the examples below you will see Windows paths. The process is the same with our SLB but the paths will be different.



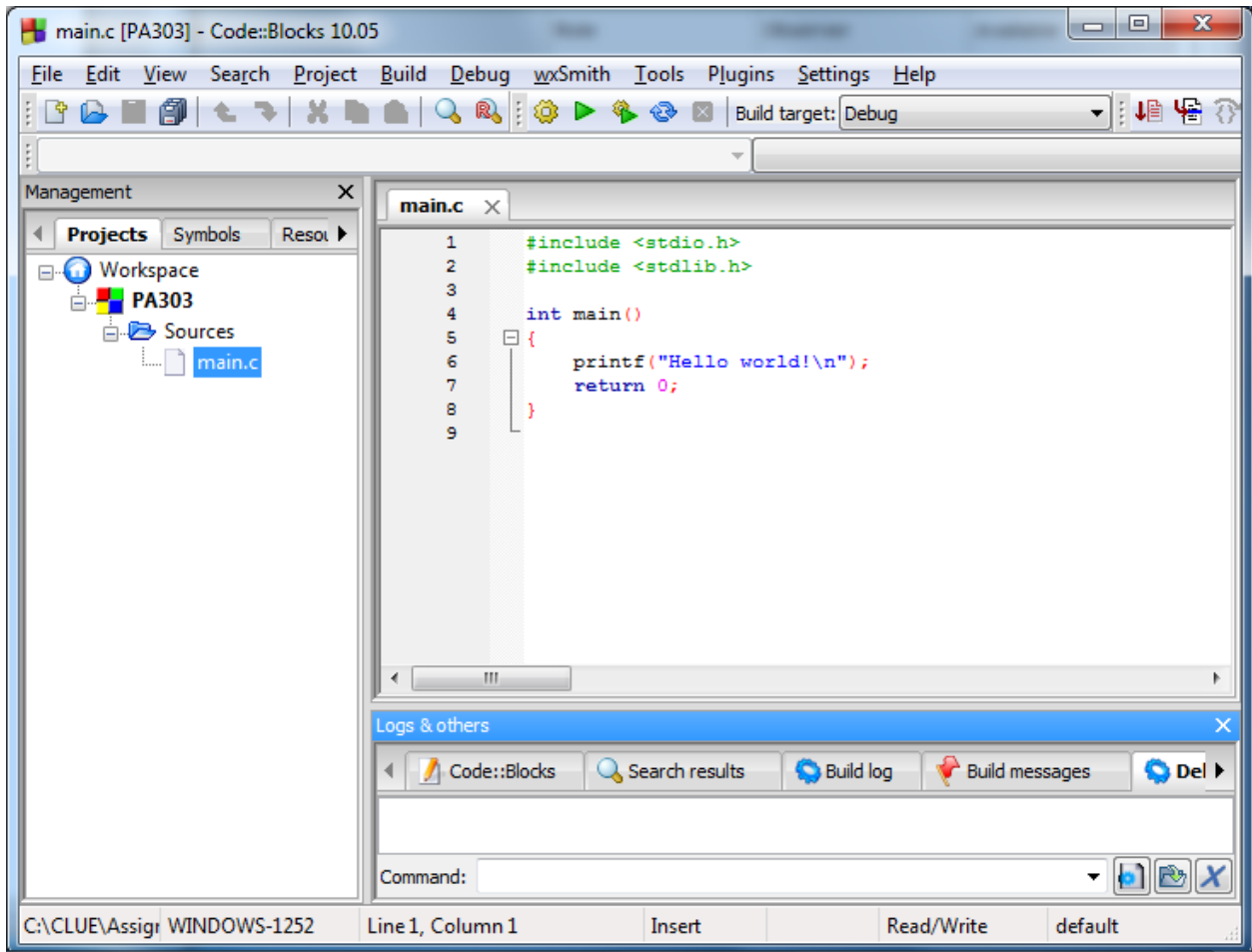
By default, Code::Block will be ready to compile two versions of your project. The **Debug** version is the one you'll work with, troubleshoot and then submit. The **Release** one won't be something we'll even use in this offering.



When you click **Finish**, you'll be back to the main window with a brand new project opened for you.



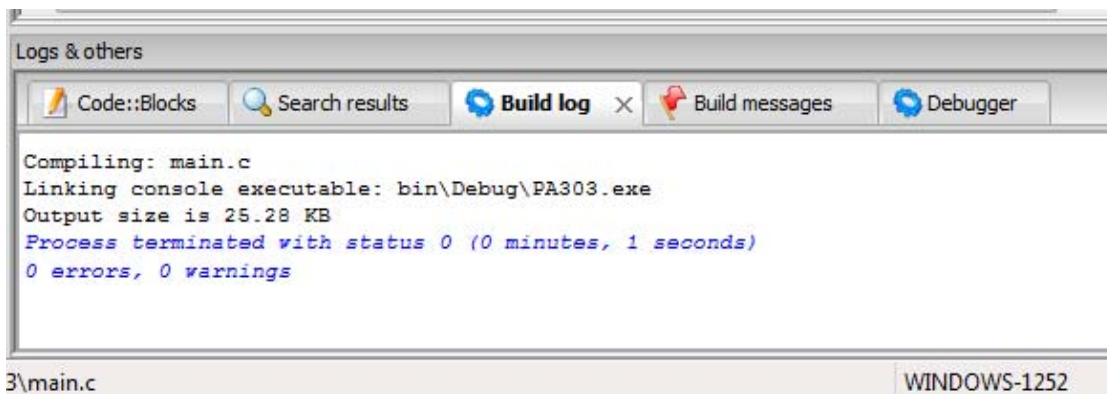
If you open the **Sources** tree on the left panel you will find a main.c already there for you.



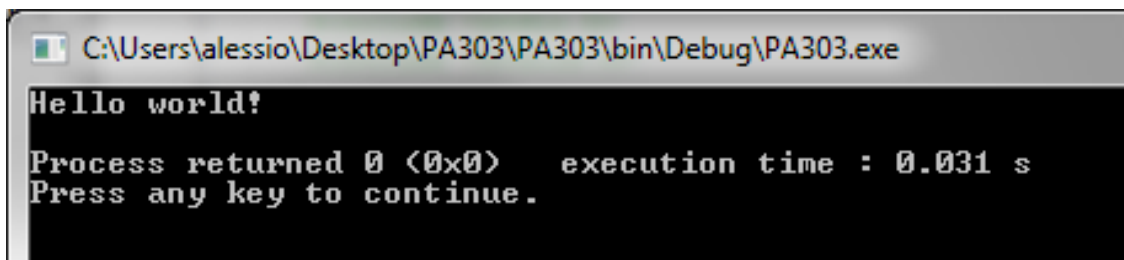
Compiling and Executing a project

In order to run your program, you must build it. This entails compiling each of the `.c` source files which are part of your project, linking them together along with any needed library with a tool named the linker. Refer to the first module's videos lectures for more information about this whole process.

You will achieve this by using the **Build** → **Build** menu or the **Ctrl-F9** shortcut. The **Build log** tab in the bottom panel will show you any potential errors.



When your project is built, you may run it with “Build” → “Run” or Ctrl-F10 shortcut. The interaction with your program will take place in DOS text Console window which will prompt you to press any key before to close when your program is done executing.

A screenshot of a DOS console window. The title bar shows the file path: C:\Users\aleccio\Desktop\PA303\PA303\bin\Debug\PA303.exe. The console output consists of three lines: "Hello world!", "Process returned 0 (0x0) execution time : 0.031 s", and "Press any key to continue.".

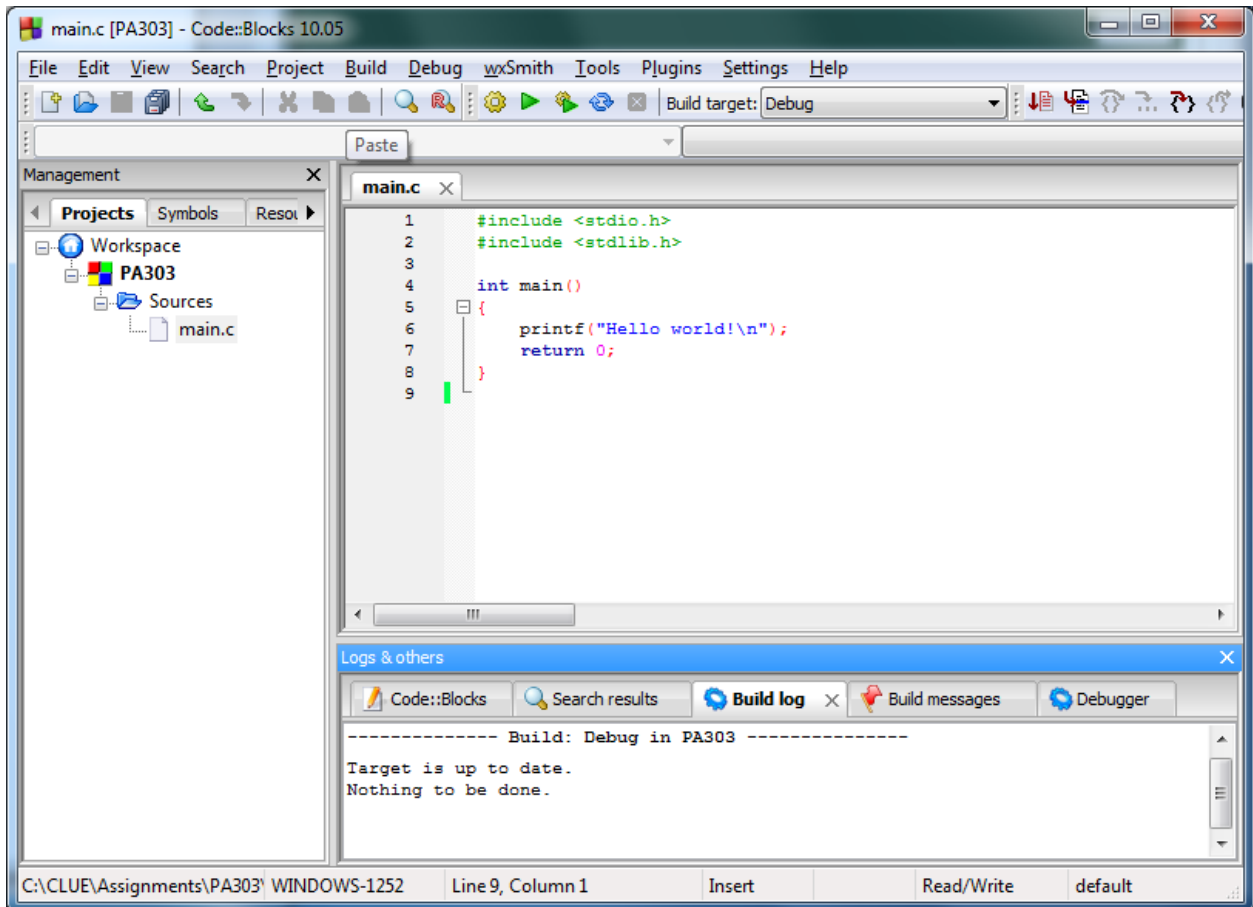
```
C:\Users\aleccio\Desktop\PA303\PA303\bin\Debug\PA303.exe
Hello world!
Process returned 0 (0x0) execution time : 0.031 s
Press any key to continue.
```

5. Debugging with Code::Blocks IDE

One of the benefits of using Code::Block is that you are able to execute step by step a program and display the value of key variables at each step. However, keep in mind you would be able to do the same thing by adding a few instructions to display the values of these same variables as your program execute. So, again, make sure you do not spend hours learning to use the debugger when you should be instead learning to program.

Preparing your program to be ran with the debugger

For the sake of demonstrating how the debugger works, we are going to use a simple “Hello World” program.

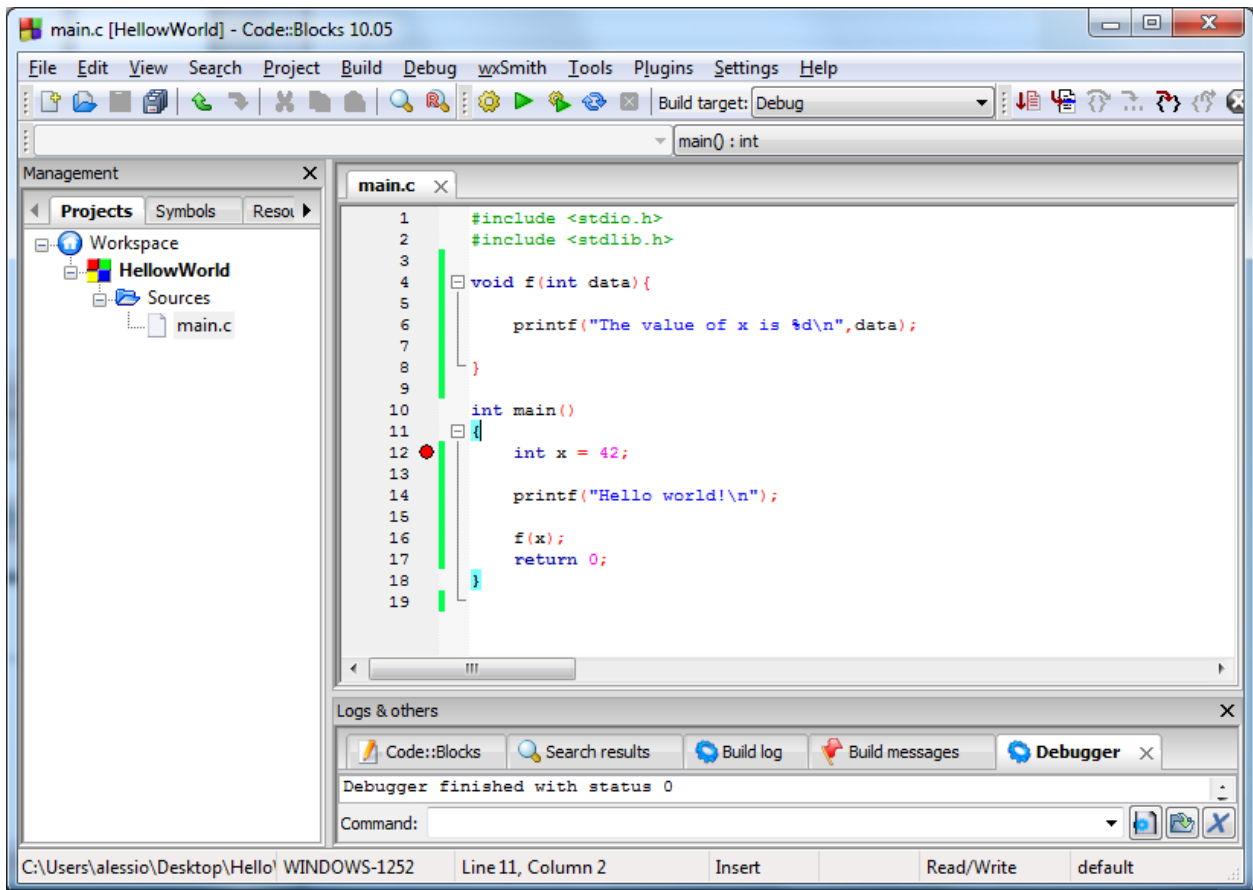


As you may notice in the above screenshot, the project is already built and ready to execute. You may not debug any project which is not already building properly & executing. If your project produces bad results or “crashes” while running, it’s ok. This is why we are going to troubleshoot it. But a debugger won’t be able to help with syntax errors.

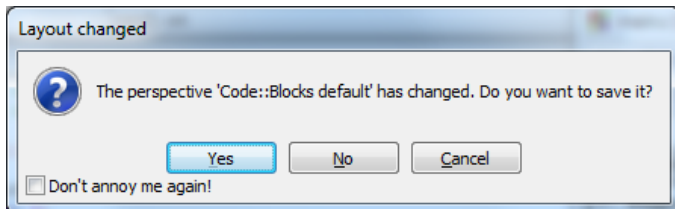
Setting up “BreakPoints”

When you use a debugger, you need to have an idea of where things are going wrong in your program. The best way to do this, is to put a breakpoint so that, when you run your program with the debugger, execution will pause at that line in the code.

Select a line of code and use **Debug** → **Toggle Breakpoint** to set or unset a breakpoint at this line. You may also use the F5 shortcut instead. Notice the red dot on the breakpoint line below;

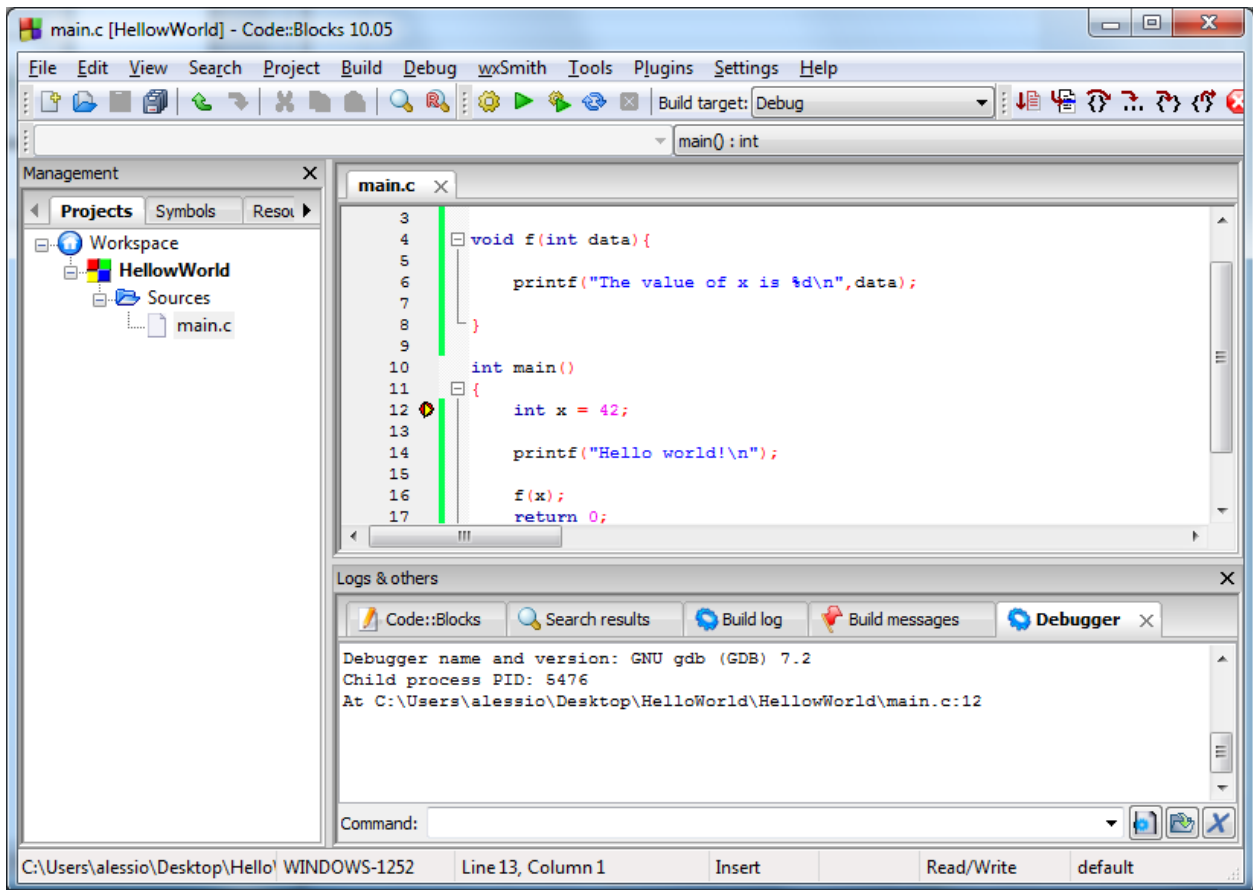


Now, we may run our program with the debugger by selecting **Debug** → **Start**. A popup will inform you that the IDE will now display your project in Debug Mode.



The, the program will start executing until it reaches your breakpoint. The display Console will be blank since we inserted a breakpoint before any of our printf.

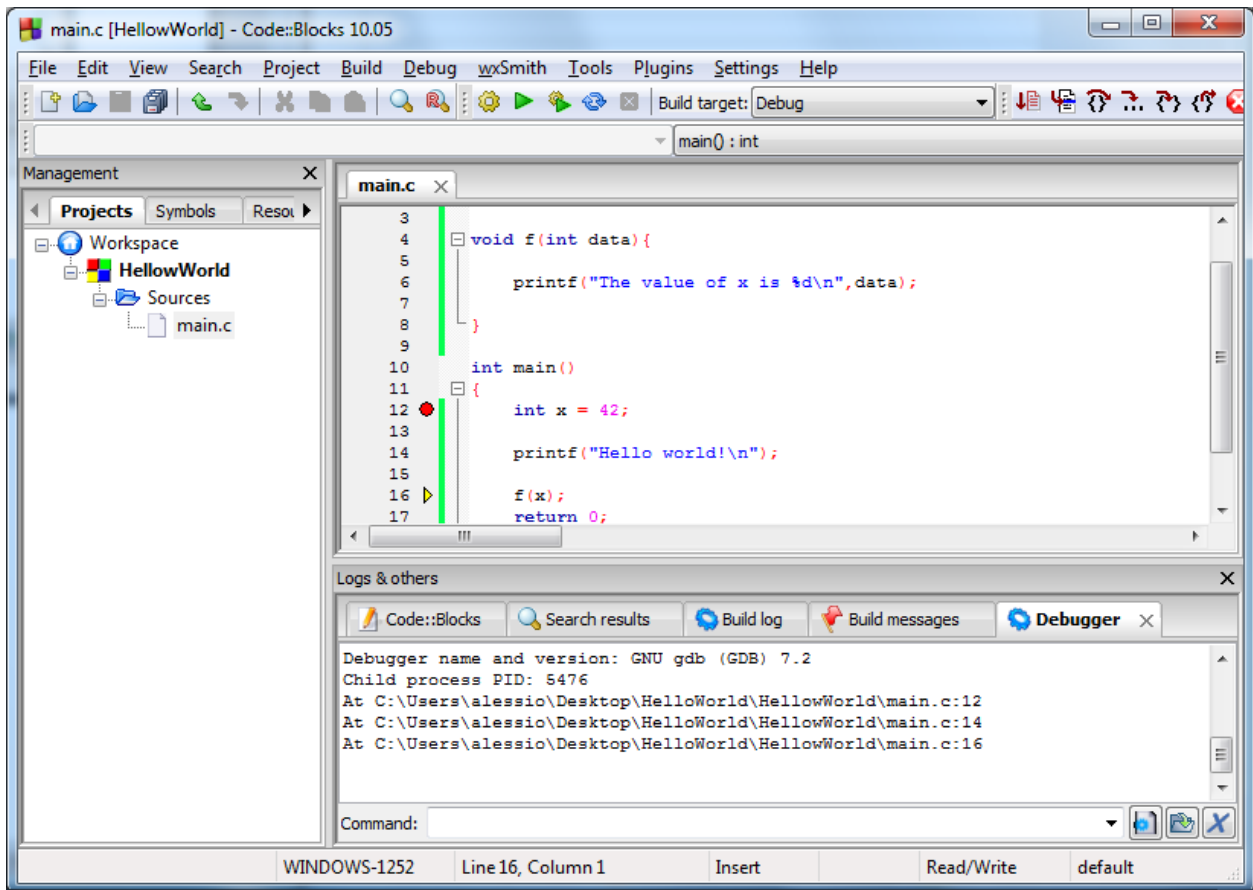
Meanwhile, the IDE window will mark that we are at the break point with a right pointing triangle.



Stepping through your programs

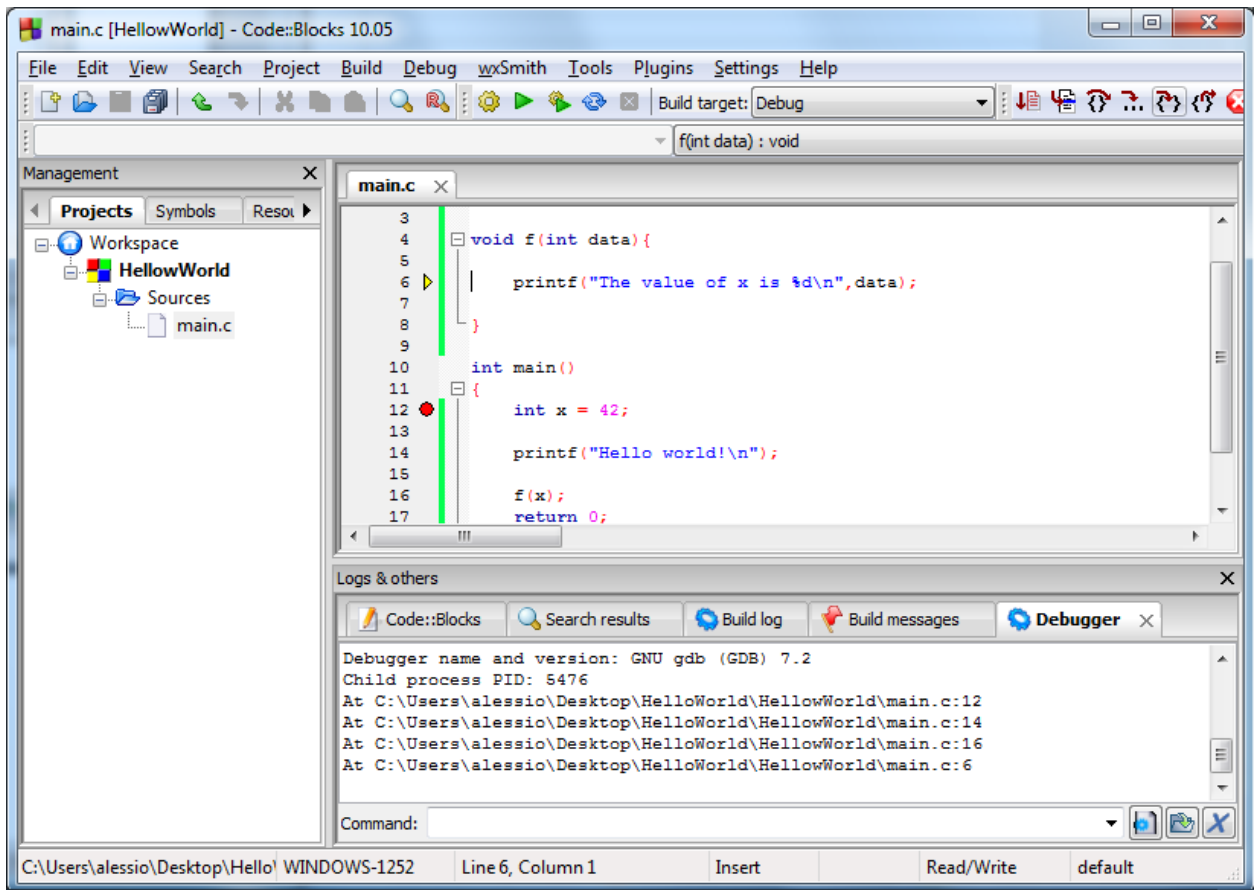
Right now, we might just select **Debug** → **Continue** and the debugger would just run until the program is done. Generally though, we don't want to do this. We want to “step” line per line through the program. There are two ways of stepping

Next Line or Next Instruction will do pretty much what it sounds it'd do. Here is the result after using it twice;

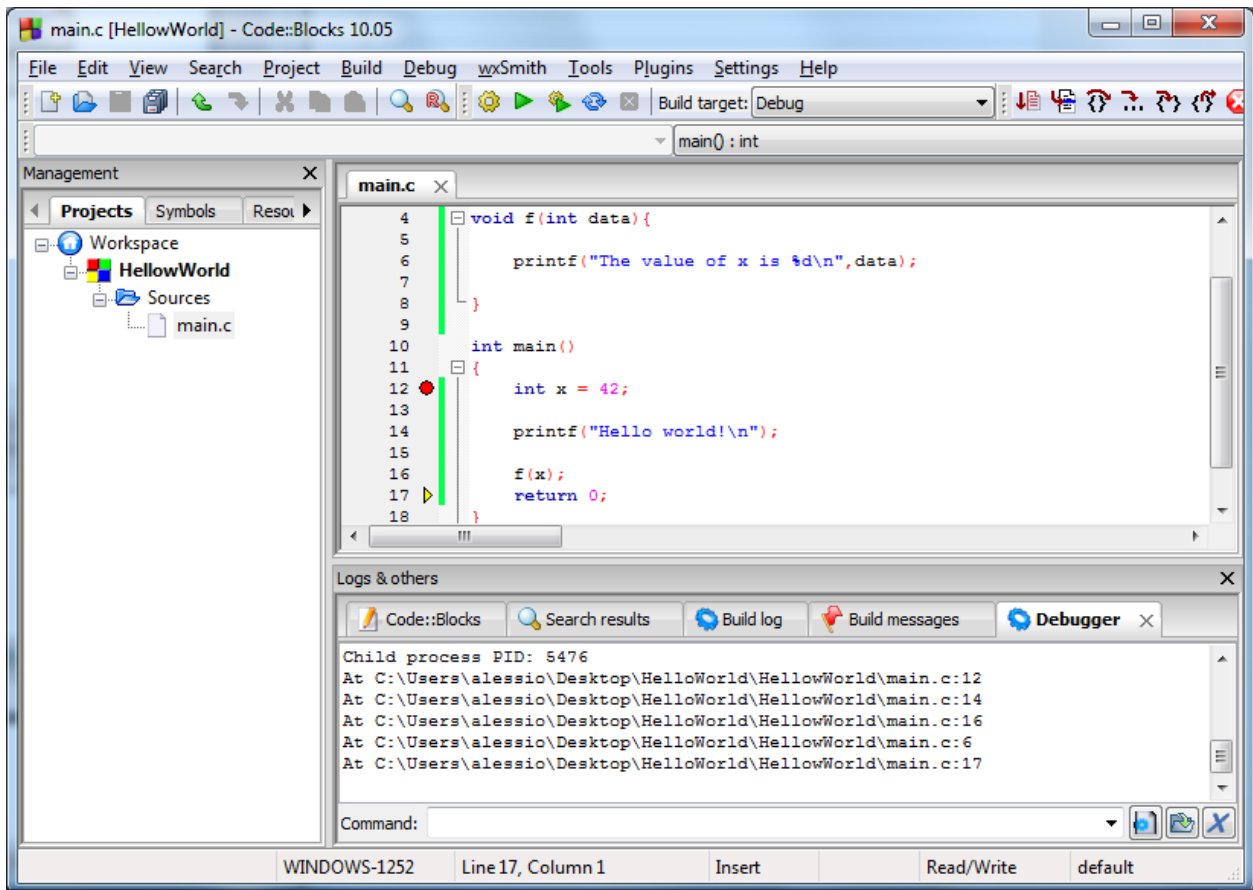


Now that we are on a function call, things are different. If we use the same way of stepping, the whole function call will be executed and we'll find ourselves on line #17 without having a possibility to step through the code of the function.

This is when you might want to use **Step into** instead which will allow you to get into the function's code and step through it.

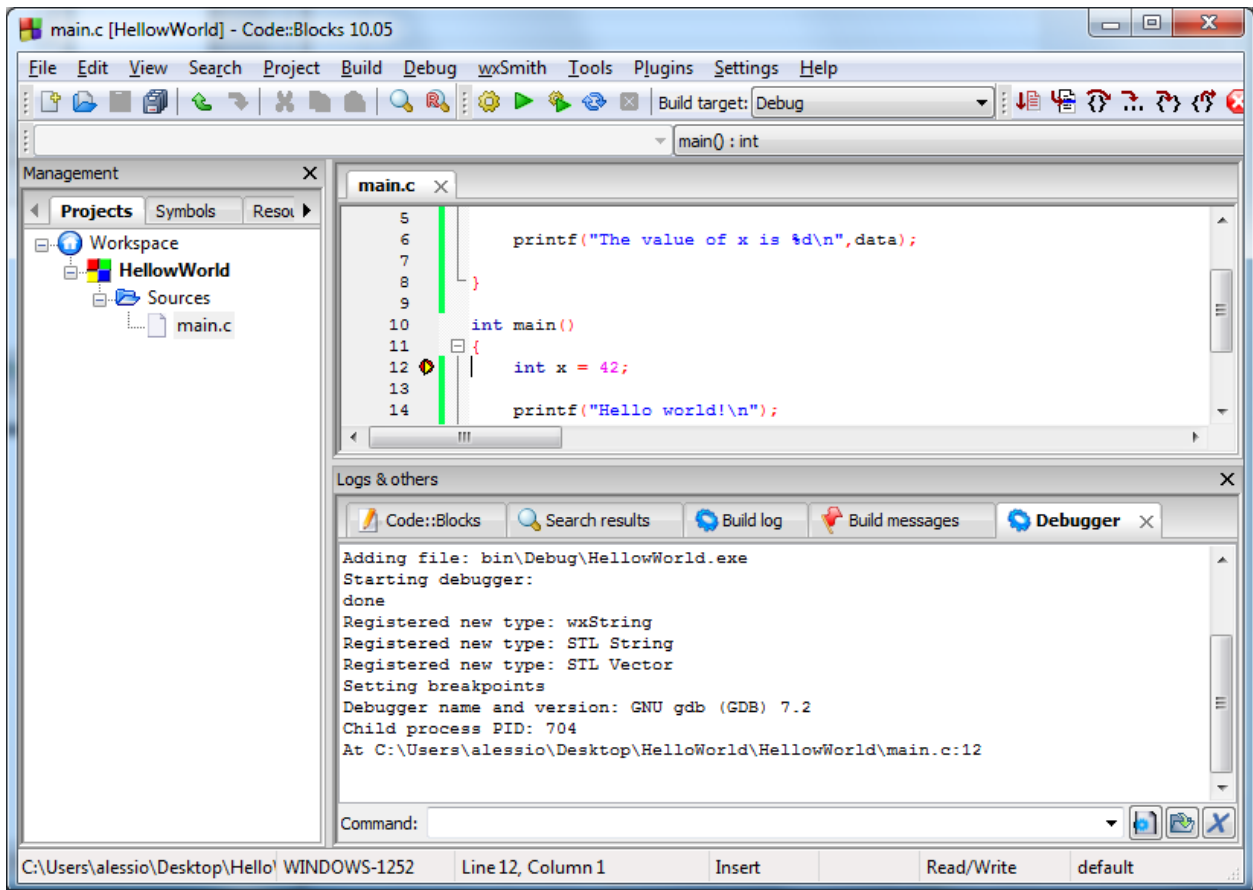


If you want to step through all its code, go for it. If at some point you just want to run until the function returns, then use the **Step Out** option.

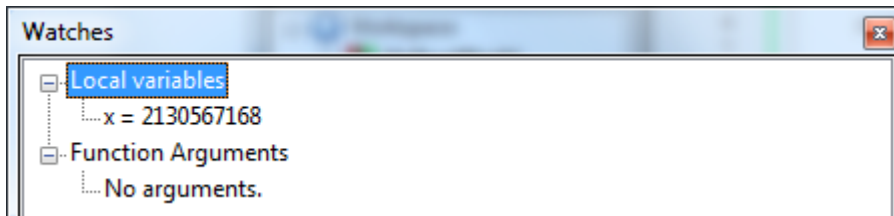


Inspecting Variables Values on the go

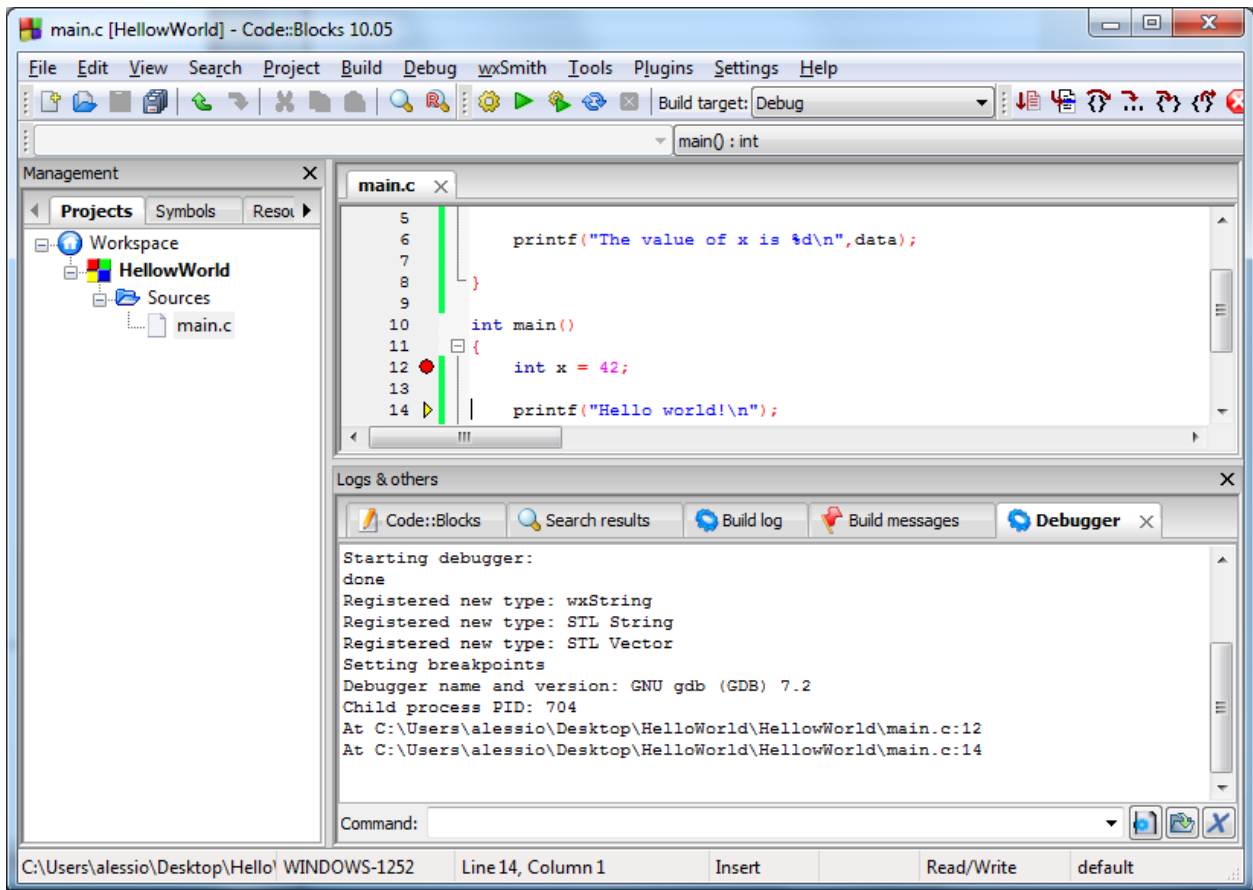
The other main feature of a debugger is to allow you to keep an eye on a list of variables as you step through the program. Let's finish the previous run and start a new one with **Debug** → **Start**.



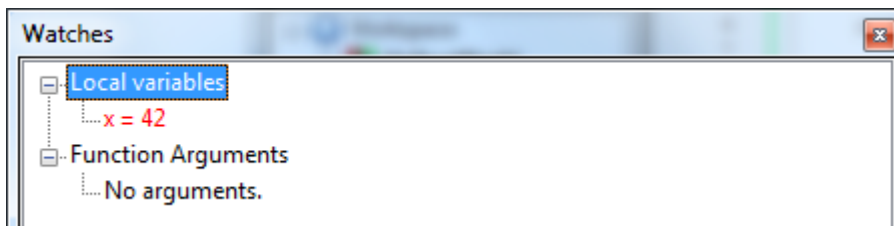
We are going to open the window holding all our variables monitoring by selecting **Debug** → **Debugging Windows** → **Watches**.



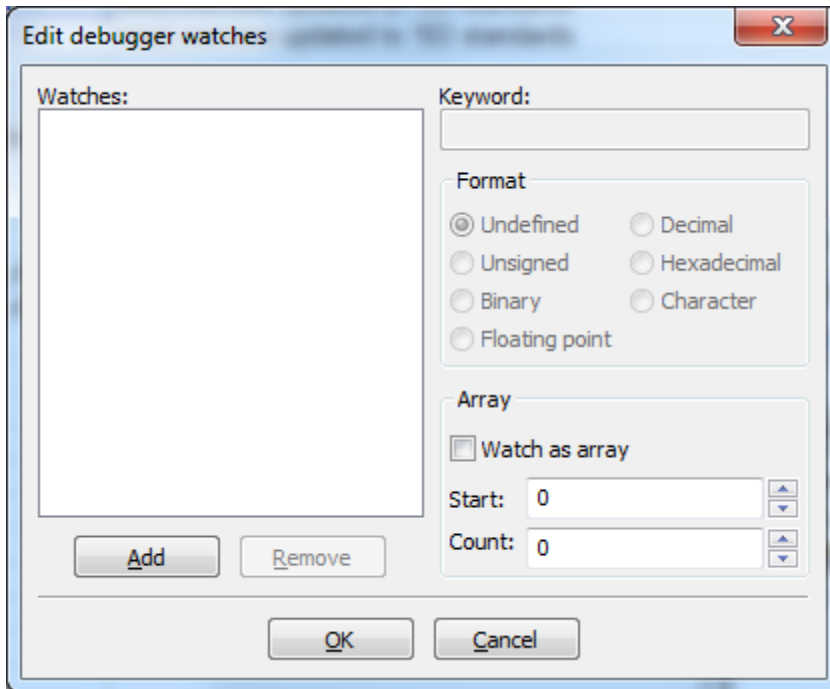
This panel shows all the local variables, here x only, and functions arguments if we are in a function accepting parameters, here none. We haven't stepped through the line of our program initializing x so it is right now holding a random value.



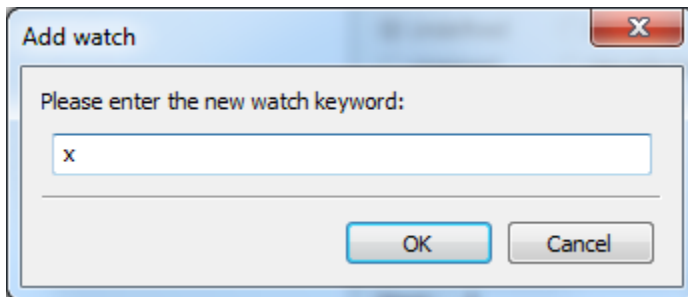
Now we've stepped through the initialization, look at your watch window now.



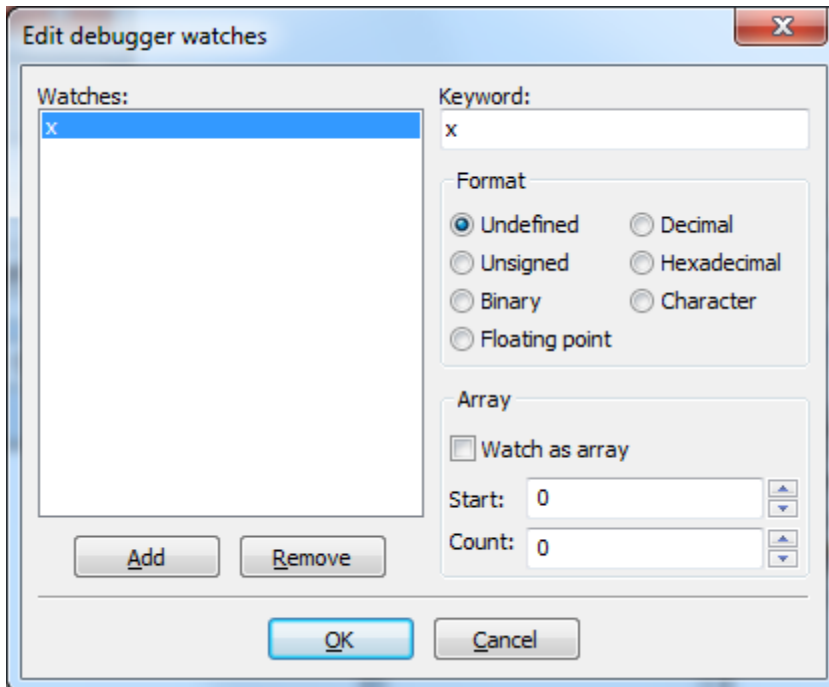
The red indicates what was modified since the last line which was executed. In addition to watching the variables the debugger finds for you, you may also add your own watches by selecting **Debug** → **Edit Watches**.



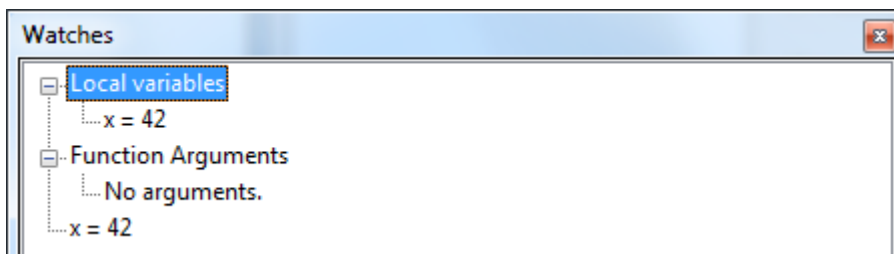
Now we are able to click the **Add** button and say we want to monitor x. Pointless, agreed, but it will show you how it's done.



Select **ok** when you're done.



The variables we specify by hand are listed at the bottom of our watch window



6. Using “NED” to track down novice errors

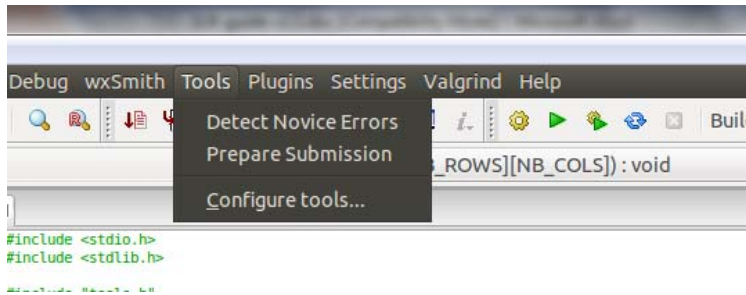
NED, the Novice Errors Detector web application, has a double objective;

- Allow students to run a few analyzers on their program to warn them about potential errors.
- Run the compiler on their programs with a set of options making it more likely to warn them about anything even remotely suspicious
- Provide students with tutorials on the most difficult to understand messages to help them understand their meaning & determine whether or not the warnings apply to their programs.

The web application may be accessed via a web browser or invoked directly from within your IDE.

Using NED from Code::Blocks

You may apply NED on the files in your current project by going in the **Tools** menu then selecting **Detect Novice Errors**.



When you do so, a Firefox window will open with a report on the analysis performed by NED on the files of your current project. You need your virtual machine to be on the internet for the IDE to be able to reach NED.

As illustrated below, the analyzer report will show the reports from different tools in a tabbed window; GCC with specific options, splint, CPPCheck, & the CLUE syntax analyzer. The last tab shows the uploaded files.

Analyzer Report

Analyzed 07/08/2013 - 06:33:38

A screenshot of the Analyzer Report interface. It features a tabbed window with tabs for 'Compilation Warnings', 'splint', 'cppcheck', 'CLUE', and 'Uploaded Files'. The 'Compilation Warnings' tab is active. Below the tabs, there is a section titled 'Results' with the text 'Nothing to report..'. Below that is a section titled 'Report generated by...' which lists details for GCC: Path: /usr/bin/gcc, Options Used: -S -std=gnu99 -Wall -pedantic -W -Wextra -Wconversion -Wuninitialized, Version: 4.4.7, Version Details: gcc (GCC) 4.4.7 20120313 (Red Hat 4.4.7-3), and More Information: http://gcc.gnu.org/.

Using NED from a browser

You may use NED by pointing Firefox to <http://CEReAL.forest.usf.edu/clue/ned/>. The interface is simple to use;

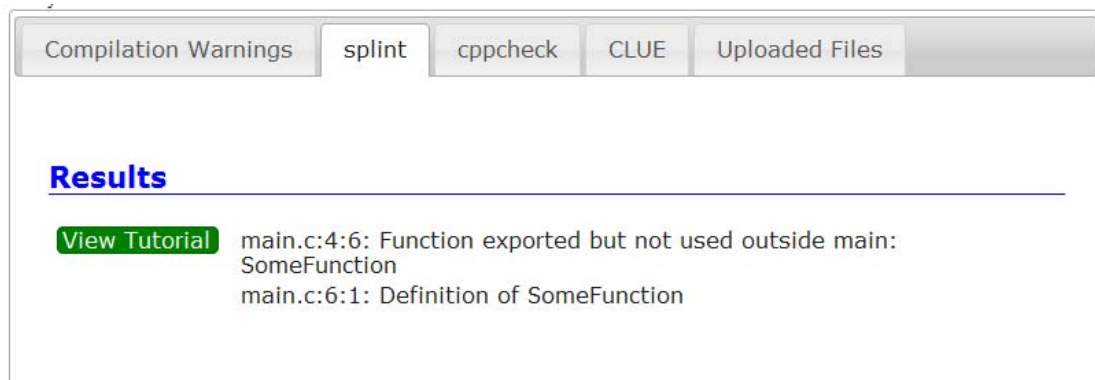
- There is a default **Browse...** button allowing you to find the first file you want to upload. If your project has a single .c file, this will suffice.

- If you have more .c or .h file, it is important you add them all. To do so, use the **add file** to add another **Browse...** button to the page. Use it to select another file & repeat as needed
- If you added too many, just leave the extra blank or use the **Remove File** button to get rid of the superfluous **Browse...** buttons & their selected files

When you are done selecting all the files in your project, use the **Analyze** button to upload them to our server & have them analyzed. The report page you will get in return is the same as the one you would get by using NED from within the IDE.

Using the reports

When warnings are available from any of the tools, NED displays a green **View Tutorial** button in front of them.



The screenshot shows a web interface with a navigation bar at the top containing buttons for 'Compilation Warnings', 'splint', 'cppcheck', 'CLUE', and 'Uploaded Files'. Below the navigation bar, the 'Results' section is displayed. A green 'View Tutorial' button is positioned to the left of a warning message. The warning message reads: 'main.c:4:6: Function exported but not used outside main: SomeFunction' and 'main.c:6:1: Definition of SomeFunction'.

Using this button will open the tutorial page on this specific warning;

Warning

Function should not be exported

What does this error / warning mean?

You have a non-static function defined in a file. This entails it might be used from another file. However, no other files are using it.

Troubleshooting

Finding the problem

The analyzer will refer you to the function.

Fixing the problem

Adding the "static" keyword in front of the function's definition will solve the problem

If you are uploading a single file to be validated, you will get this error for every defined function. This is simply due to the fact the analyzer was working only on your file & therefore assumed no other files were using the function even if it's true in your project folder.

Examples

```
splint-fn-exports-1.c [ Download ] [ Analyze ]  
1. /*  
2. e      All functions not used in other files should be static.  
3.      It's silly if this is your only file in the project.  
4.      If you are using some functions in other files but you
```

Each tutorial will have a “mini-survey” at the bottom of its page allowing students to let us know whether it was useful or not. This feedback is important to help us improve tutorials during the semester so they may be more useful.

Was this tutorial useful?

Your feedback is essential in helping us identify both the tutorials which need improvement & the tutorials which we should be using as "models". So, please take a second to let us know whether this tutorial was useful to you.

- [Yes](#), this tutorial was helpful to me
- [No](#), this tutorial was not helpful to me

Don't hesitate to [Drop us an email](#) to let us know in more details how we may improve this tutorial. You may even attach your file or provide screenshots to supplement your explanation.

Some warnings do not have yet a tutorial. However, if you have trouble with them, you will find a [request tutorial](#) button which will allow you to inform the authors you could have used a tutorial to help you with this error message. Our goal is not to provide tutorials on warnings or error messages which are trivial to understand but rather identify the ones which our students struggle to find useful so we may attempt to explain them better. To this end, your feedback is essential in helping us identify both missing tutorials & those which are useful as is or need improvement.

What does it mean to validate your program with NED?

NED uses different tools to identify well-known novice errors in your programs. Most static code analysis tools will generate a fair amount of “false positive” meaning that they are at best able to produce warnings about anything which look suspect.

The tutorial pages should help you better understand the meaning of each warning, learn about the potential bugs they describe then decide whether or not your program features it. Whether you had a false positive or not, the end result should be an improvement of your programming skills by simply learning about yet another possible type of bug. However, we hope the tools we are using will also bring to your attention actual bugs.

7. Validating your tests & solutions

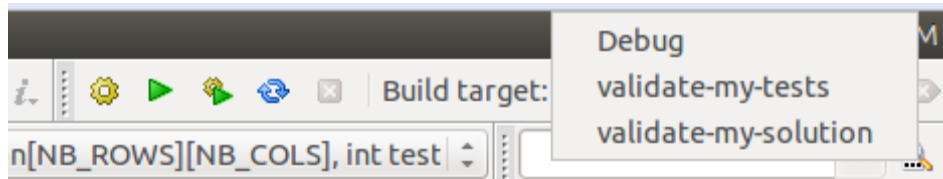
As part of our integration of the Programming Assignments” with the IDE, we have added “build targets” allowing you to perform two complementary forms of validation of your programs before to submit them for grading.

Validating your solution against the instructor’s tests

All but the first two PAs will require you to write functions to address a given problem & functions to ensure the former are working properly. This is meant to get you started thinking about testing your programs automatically using other functions.

Each PAs has a compiled version of the instructor’s tests. These are not exhaustive tests but are a sample of tests your program should be able to pass. They are provided only to help you understand the requirements & develop your own comprehensive suite of tests.

In order to run the functions you implemented to solve the problem at hand against the instructor’s tests, you need to use the drop down menu labeled **Build Target**. It is generally located toward the top-right of the IDE window.



Select **validate-my-solution** from the drop down menu then **Build** → **Build**. The tests you developed in your project will be set aside & replaced by the instructor’s. The output produced by running your functions against these tests will be displayed.

This form of validation will provide you with partial hints as to what your own tests should be doing. This, in turn, will shed some light on the requirements & the correctness of your solutions.

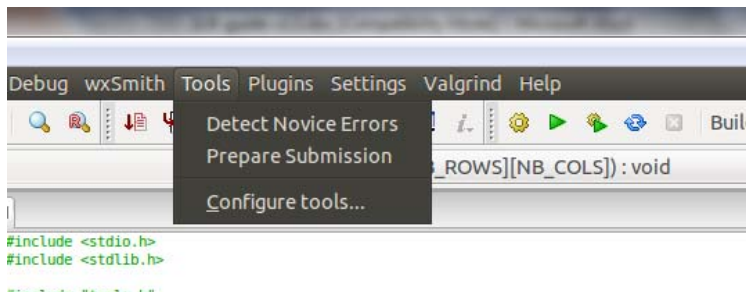
Validating your solution against the instructor’s tests

Similarly, you may select the **validate-my-tests** build target then **Build** → **Build** in order to run your tests against the instructor’s solution.

This form of validation will provide you with an idea of whether some of your tests are incorrect. If one of your test fails with the instructor's solution, make sure to email immediately to rule out the possibility of your test being based on an inaccurate understanding of the requirements.

8. Preparing your submissions

Another tool has been added to your IDE. By going to the **Tools** menu & selecting **Prepare Submission**, you will be able to generate an archive of your project which you may then submit to your instructor as he or she instructed you to; e.g. transfer it to the host's shared folder in order to email it, upload or FTP it to a specific site...



After select this tool, you will find a new file on your desktop. The file name will be always based on the number of the PA you are working on, e.g. PA201, followed by a data & time stamp.

Please note that the material being packaged also includes the entire development history of your project; i.e. how & when you modified the files in between every compilation or build of the project. This extra information might be required by your instructor to allow him or her to not only grade you on the final result but also on how you reached it. This will allow you to also receive feedback on how you worked on your project, therefore potentially improving your technique.

9. Other tools

A plethora of other tools have been pre-installed on your SLB virtual appliance. We invite you to spend some time researching & leveraging them.

- <http://valgrind.org/>
- [http://elinux.org/Electric Fence](http://elinux.org/Electric_Fence)
- <http://www.gnu.org/software/make/>
- <http://subversion.apache.org/>
- <http://git-scm.com/>
- <http://bazaar.canonical.com/en/>
- <http://www.fossil-scm.org/index.html/doc/trunk/www/index.wiki>
- <http://www.splint.org/>
- <http://cppcheck.sourceforge.net/>