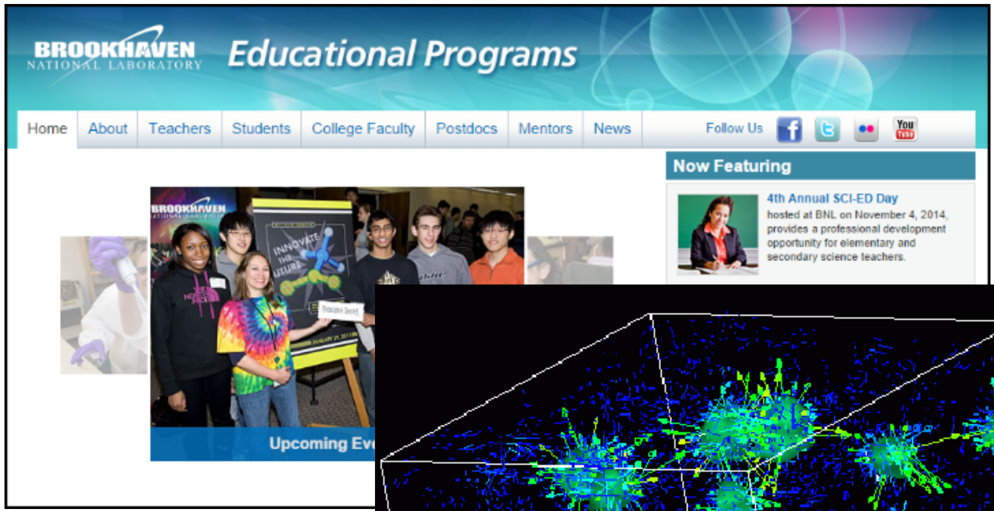
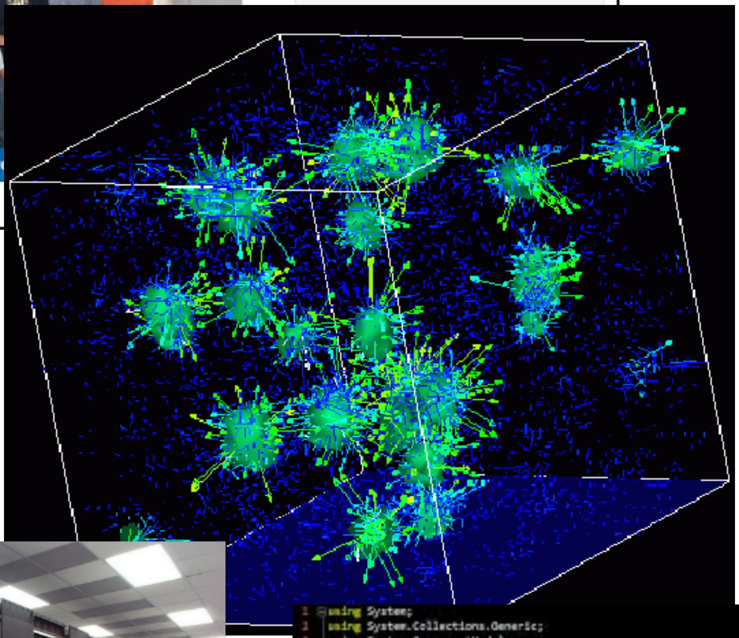
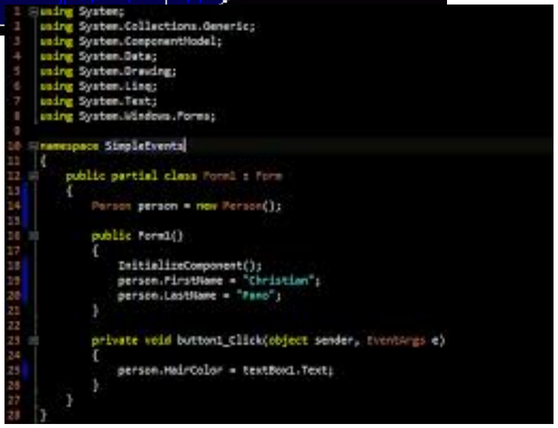

BROOKHAVEN NATIONAL LABORATORY
Educational Programs
Home
About
Teachers
Students
College Faculty
Postdocs
Mentors
News
Follow Us
Now Featuring
4th Annual SCI-ED Day
hosted at BNL on November 4, 2014, provides a professional development opportunity for elementary and secondary science teachers.
Upcoming Ev





```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace SimpleEvents
{
    public partial class Form1 : Form
    {
        Person person = new Person();

        public Form1()
        {
            InitializeComponent();
            person.FirstName = "Christian";
            person.LastName = "Fano";
        }

        private void button1_Click(object sender, EventArgs e)
        {
            person.HairColor = textBox1.Text;
        }
    }
}
```

Information adapted from cplusplus.com

Survey of Scientific Computing (SciComp)

Unit 1
Compilers

Goals

- Understand the Purpose of a Compiler and IDE
- Learning by use of Console Programs
- Get familiar the our IDE: Codeblocks
- Lab 1: Hello World!

What is a compiler?

Computers understand only one language and that language consists of sets of instructions made of ones and zeros. This computer language is appropriately called *machine language*.

A single instruction to a computer could look like this:

00000	10011110

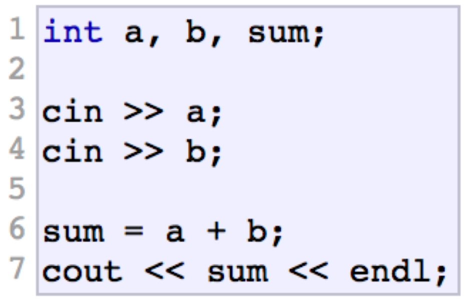
A particular computer's machine language program that allows a user to input two numbers, adds the two numbers together, and displays the total could include these machine code instructions:

00000	10011110
00001	11110100
00010	10011110
00011	11010100
00100	10111111
00101	00000000

What is a compiler? Cont.

As you can imagine, programming a computer directly in machine language using only ones and zeros is very tedious and error prone. To make programming easier, high level languages have been developed. High level programs also make it easier for programmers to inspect and understand each other's programs easier.

This is a portion of code written in C++ that accomplishes the exact same purpose:



```cpp
int a, b, sum;

cin >> a;
cin >> b;

sum = a + b;
cout << sum << endl;
```

Even if you cannot really understand the code above, you should be able to appreciate how much easier it will be to program in the C++ language as opposed to machine language.

Because a computer can only understand machine language and humans wish to write in high level languages high level languages have to be re-written (translated) into machine language at some point. This is done by special programs called compilers, interpreters, or assemblers that are built into the various programming applications.

C++ is designed to be a compiled language, meaning that it is generally translated into machine language that can be understood directly by the system, making the generated program highly efficient. For that, a set of tools are needed, known as the development toolchain, whose core are a compiler and its linker.

Console Programs

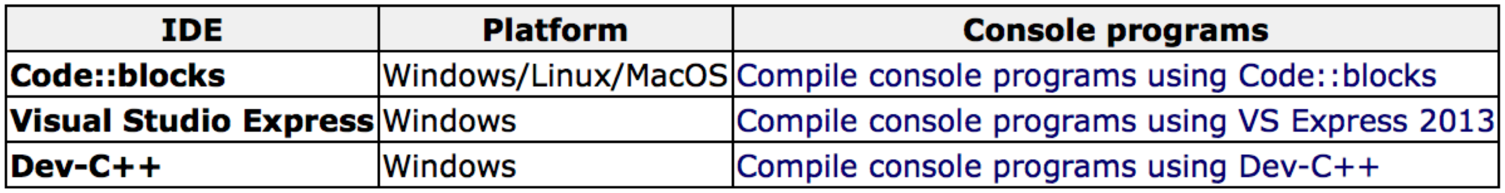
Console programs are programs that use text to communicate with the user and the environment, such as printing text to the screen or reading input from a keyboard.

Console programs are easy to interact with, and generally have a predictable behavior that is identical across all platforms. They are also simple to implement and thus are very useful to learn the basics of a programming language: The examples in these tutorials are all console programs.

The way to compile console programs depends on the particular tool you are using.

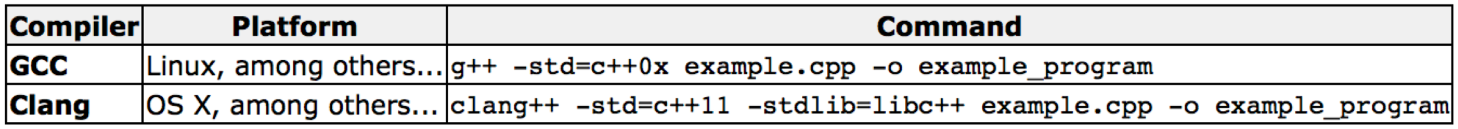
The easiest way for beginners to compile C++ programs is by using an **Integrated Development Environment (IDE)**. An IDE generally integrates several development tools, including a text editor and tools to compile programs directly from it.

Here you have instructions on how to compile and run console programs using different free Integrated Development Interfaces (IDEs):



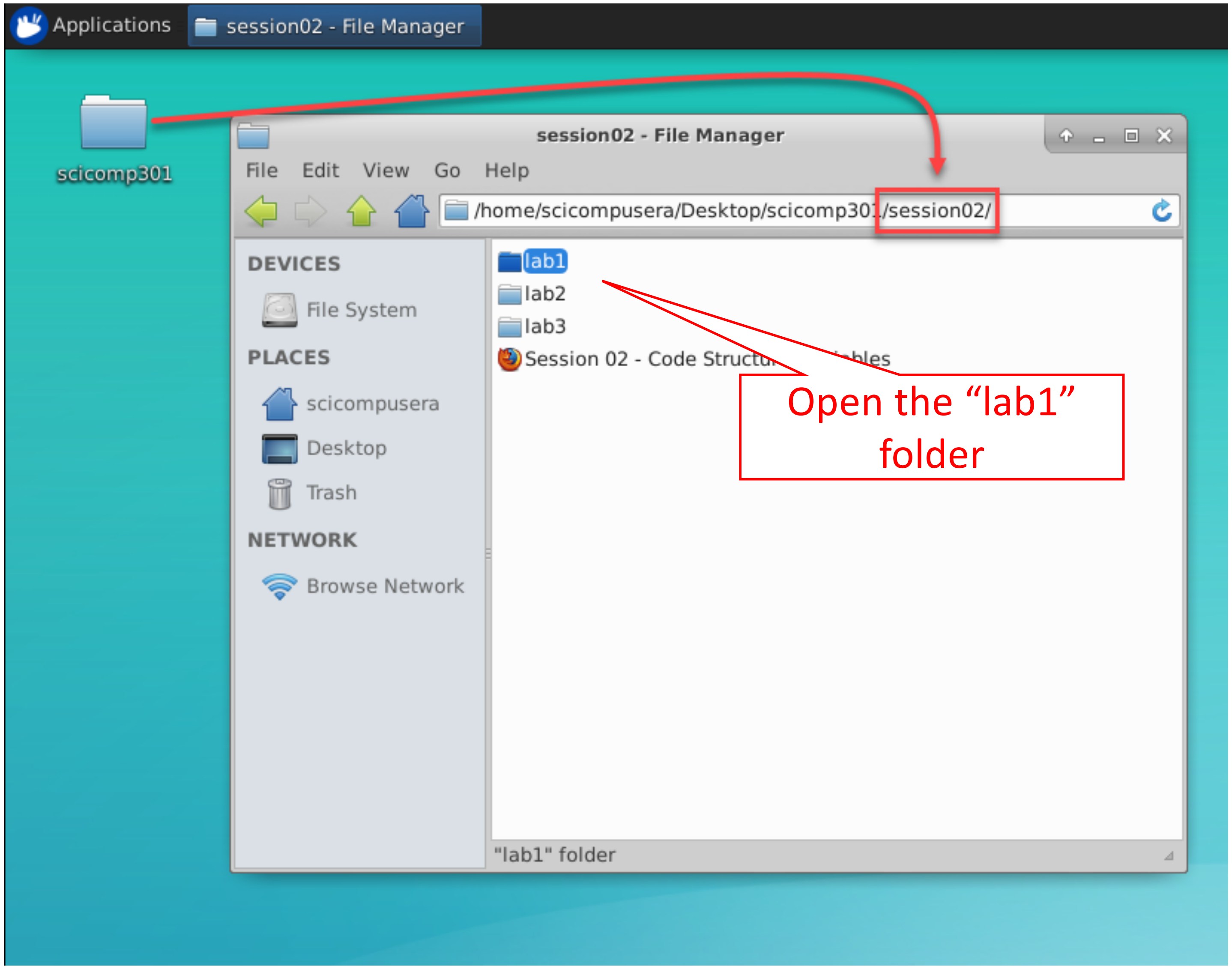
IDE	Platform	Console programs
Code::blocks	Windows/Linux/MacOS	Compile console programs using Code::blocks
Visual Studio Express	Windows	Compile console programs using VS Express 2013
Dev-C++	Windows	Compile console programs using Dev-C++

If you happen to have a Linux or Mac environment with development features, you should be able to compile any of the examples directly from a terminal just by including C++11 flags in the command for the compiler:



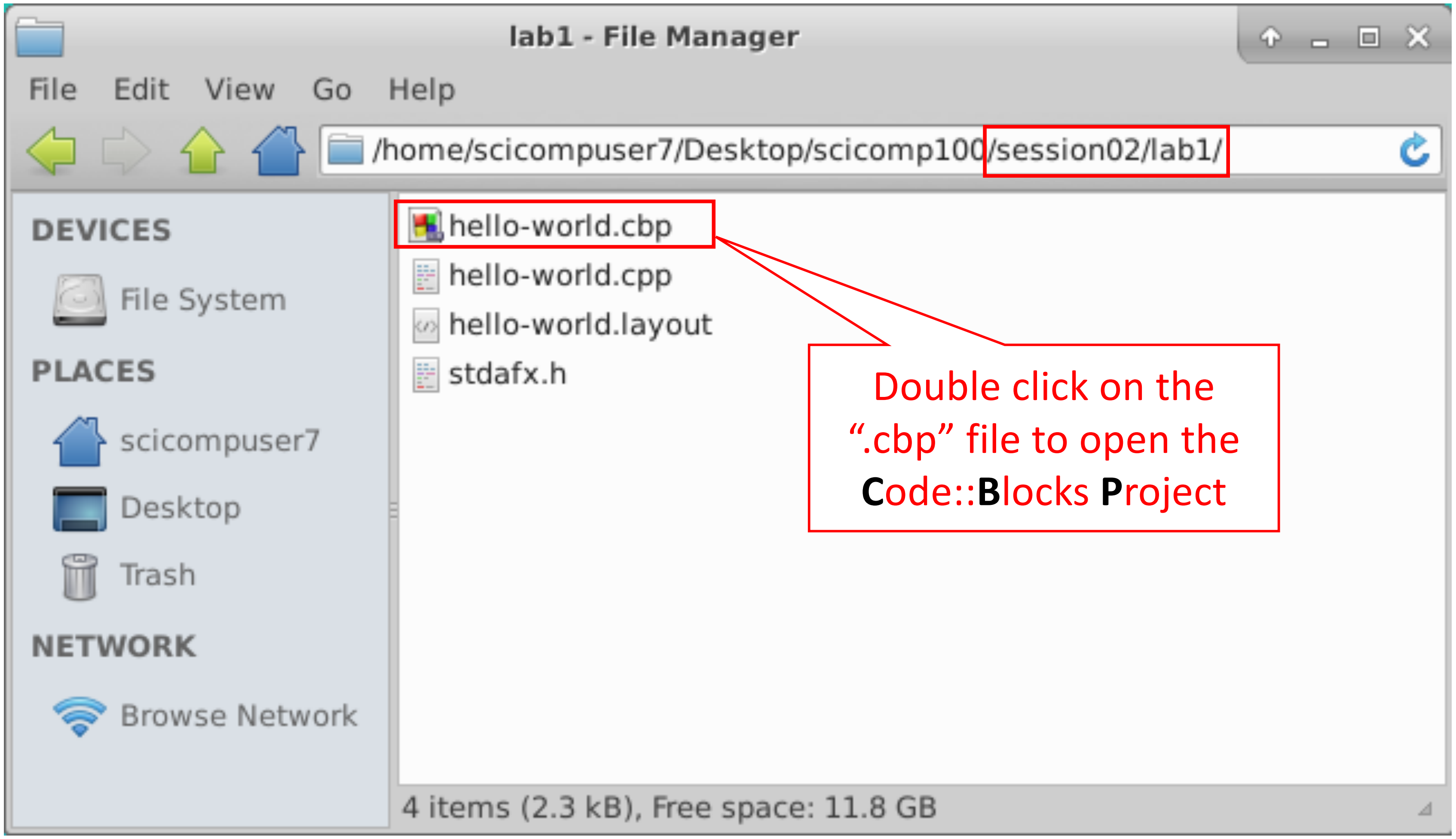
Compiler	Platform	Command
GCC	Linux, among others...	`g++ -std=c++0x example.cpp -o example_program`
Clang	OS X, among others...	`clang++ -std=c++11 -stdlib=libc++ example.cpp -o example_program`

Lab 1 – Hello World!

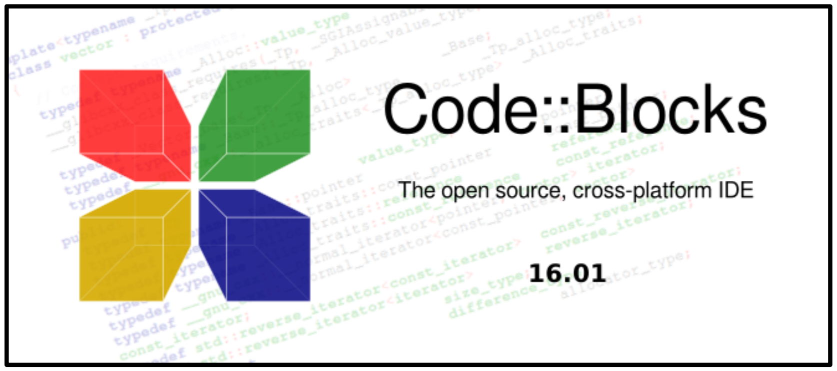


Open the "lab1" folder

Lab 1 – Hello World!



Double click on the ".cbp" file to open the **C**ode::**B**locks **P**roject



http://www.codeblocks.org

Highlights:

- **Open Source!** GPLv3, no hidden costs.
- **Cross-platform**. Runs on Linux, Mac, Windows (uses wxWidgets).
- Written in C++. No interpreted languages or proprietary libs needed.
- Extensible through plugins

Compiler:

- **Multiple compiler support**:
  - GCC (MingW / GNU GCC)
  - MSVC++
  - clang
  - Digital Mars
  - Borland C++ 5.5
  - Open Watcom
  - ...and more
- **Very fast** custom build system (no makefiles needed)
- Support for **parallel builds** (utilizing your CPU's extra cores)
- Multi-target projects
- Workspaces to combine multiple projects
- Inter-project dependencies inside workspace
- Imports MSVC projects and workspaces (NOTE: assembly code not supported yet)
- Imports Dev-C++ projects

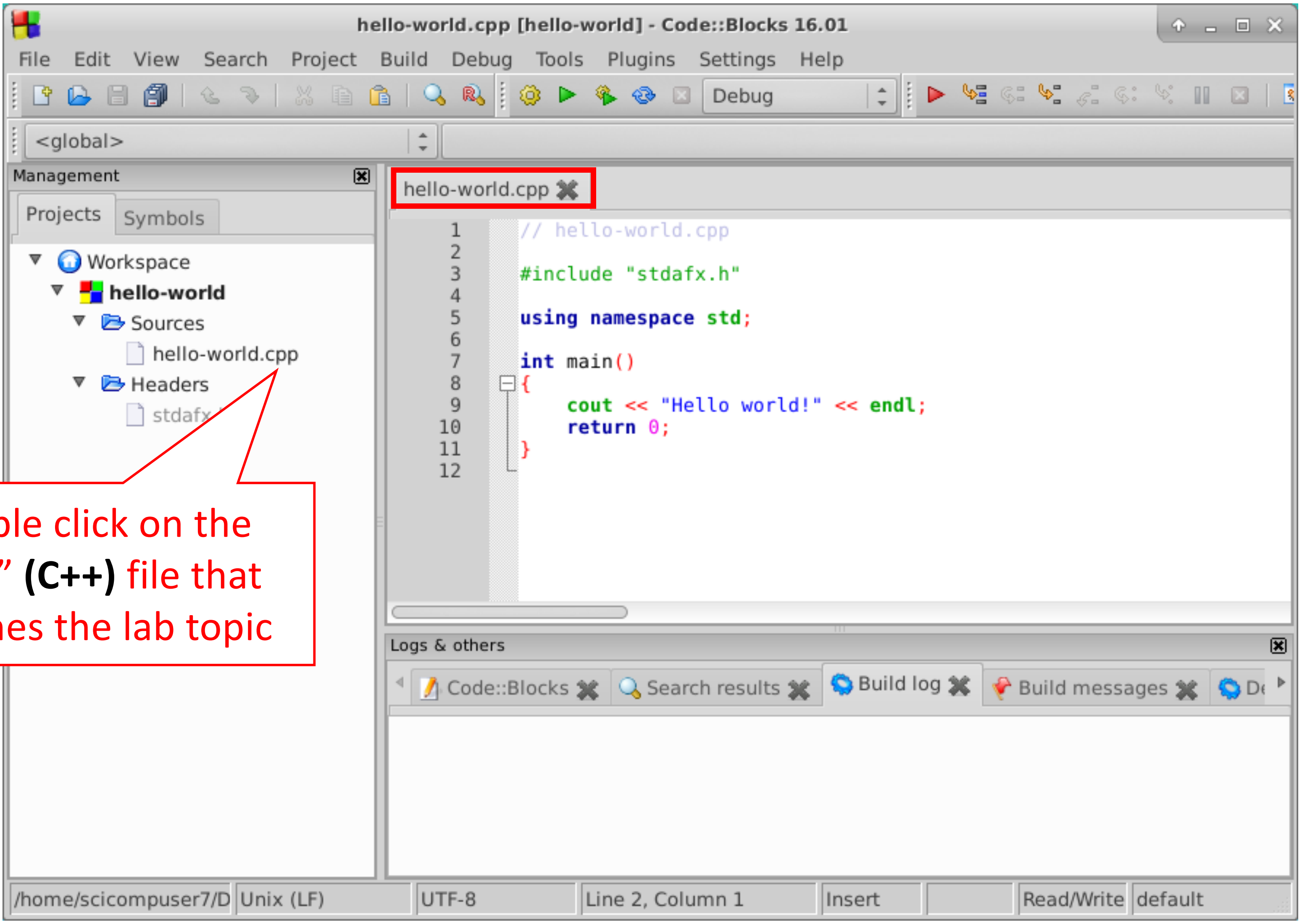
Debugger:

- Interfaces GNU GDB
- Also supports MS CDB (not fully featured)
- **Full breakpoints support**:
  - Code breakpoints
  - Data breakpoints (read, write and read/write)
  - Breakpoint conditions (break only when an expression is true)
  - Breakpoint ignore counts (break only after certain number of hits)
- Display local function symbols and arguments
- User-defined watches (support for watching user-defined types through scripting)
- Call stack
- Disassembly
- Custom memory dump
- Switch between threads
- View CPU registers

Interface:

- Syntax highlighting, customizable and extensible
- Code folding for C, C++, Fortran, XML and many more files.
- Tabbed interface
- Code completion
- Class Browser
- Smart indent
- One-key swap between .h and .c/.cpp files
- Open files list for quick switching between files (optional)
- External customizable "Tools"
- To-do list management with different users

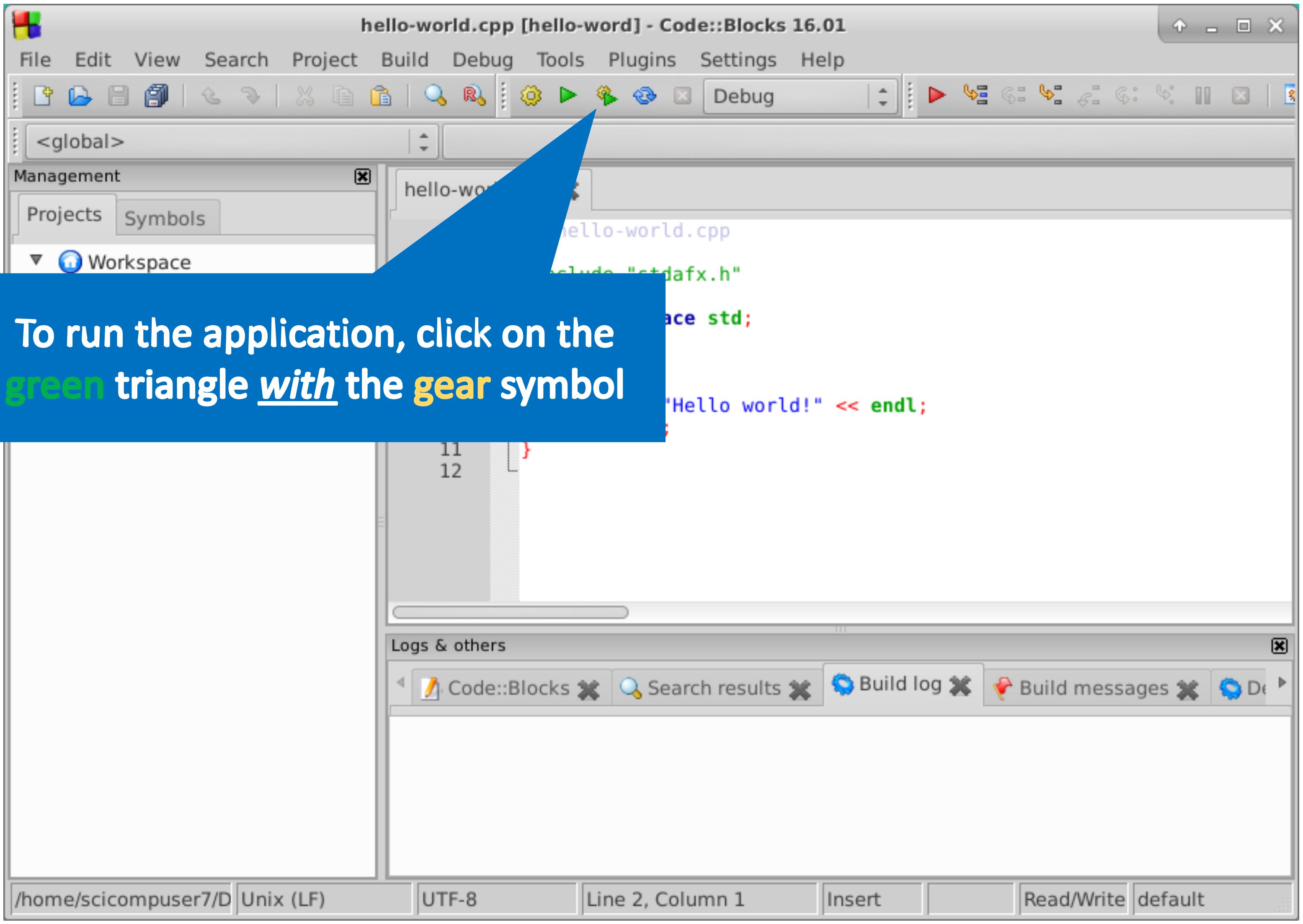
Lab 1 – Hello World!



```
// hello-world.cpp

#include "stdafx.h"

using namespace std;

int main()
{
    cout << "Hello world!" << endl;
    return 0;
}
```

Double click on the ".cpp" **(C++)** file that matches the lab topic

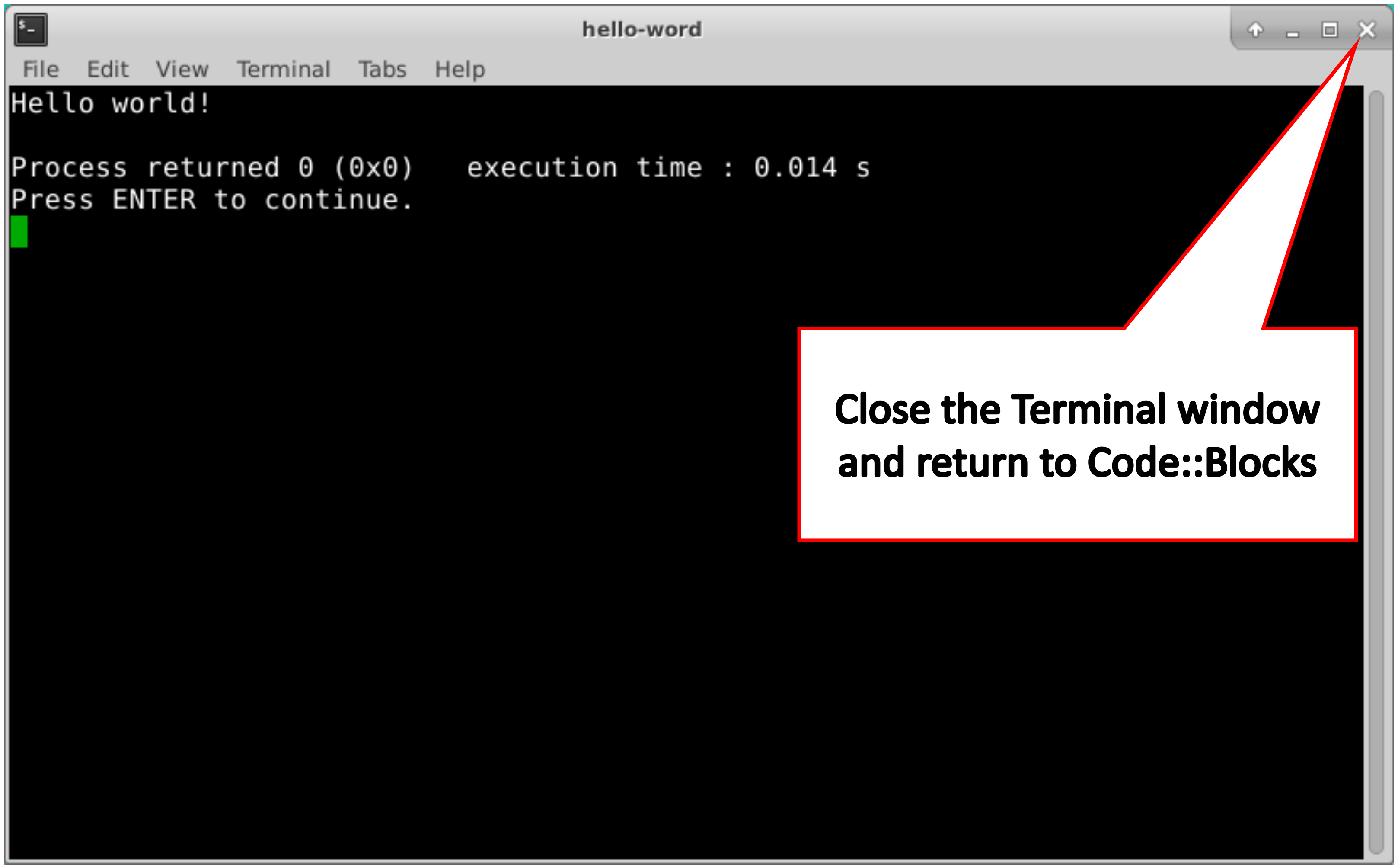
Lab 1 – Hello World!



```
hello-world.cpp
#include "stdafx.h"
ace std;
"Hello world!" << endl;
}
```

To run the application, click on the green triangle *with* the gear symbol

Lab 1 – Hello World!



```
Hello world!

Process returned 0 (0x0)   execution time : 0.014 s
Press ENTER to continue.
```

Close the Terminal window and return to Code::Blocks

Hints on Using Xfce & Code::Blocks

- Terminal = Shell = Console = Command Line
  - The shell is used to interact with character mode applications
  - Linux programmers mostly use the command line
  - Shell commands and file names are **case-sensitive**
  - Don't accumulate open **terminal windows** – close them!
- Only keep one (1) instance of **Code::Blocks** open
  - After each lab, close all open instances of Code::Blocks
  - To open a lab, be sure to double-click on the **.cbp** file (.~~cpp~~!)
- Your session will auto-logout after an hour of of inactivity

Now you know…

- Understand the Purpose of a Compiler and IDE
- Learning by use of Console Programs
- Get familiar the our IDE: Codeblocks
- Basic understanding of the program layout-
  Lab 1: Hello World!