FREE eBook

LEARNING boost

Free unaffiliated eBook created from
Stack Overflow contributors.

#boost

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: boost

It is an unofficial and free boost ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official boost.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with boost

Remarks

What is Boost?

Boost is a large collection of free, high quality C++ libraries that cover a broad range of topics. It is often considered a "second standard library" for C++, since many common problems in C++ are solved by using Boost.

From boost.org:

> Boost provides free peer-reviewed portable C++ source libraries.
>
> We emphasize libraries that work well with the C++ Standard Library. Boost libraries are intended to be widely useful, and usable across a broad spectrum of applications. The Boost license encourages both commercial and non-commercial use.

Some Boost libraries have even made their way into the C++11 standard library, and some other, such as Boost.Optional and Boost.Variant, will be a part of C++17.

What can Boost do?

Boost covers most corners of programming. From the boost tag wiki here on Stack Overflow:

> It includes libraries for:
>
> - String and text processing
> - Containers
> - Iterators
> - Algorithms
> - Function objects and higher-order programming
> - Generic Programming
> - Template Metaprogramming
> - Preprocessor Metaprogramming
> - Concurrent Programming
> - Math and numerics
> - Correctness and testing
> - Data structures
> - Image processing
> - Input/Output
> - Inter-language support
> - Memory
> - Parsing
> - Programming Interfaces
> - Miscellaneous

- Broken compiler workarounds

Versions

Version	New Libraries	Release Notes	Release Date
1.10.0		notes	1999-12-14
1.11.0	Rational Number	notes	2000-02-01
1.12.0		notes	2000-02-23
1.13.0	Utility, Type Traits, Call Traits, Compressed Pair	notes	2000-02-29
1.14.0		notes	2000-03-05
1.15.0	Random Number	notes	2000-06-17
1.16.0	Functional, iterator header	notes	2000-06-28
1.17.0	Array	notes	2000-08-03
1.18.0	Graph, Regular Expression	notes	2000-09-28
1.19.0	Concept Check, Python, Static Assert, Property Map Concepts	notes	2000-12-10
1.20.0	Conversion	notes	2001-01-06
1.21.0	Iterator Adaptor, Pool, Test	notes	2001-03-09
1.22.0	CRC	notes	2001-05-25
1.23.0	Any, Function, Tokenizer, Special Functions, Octonions, Quaternions	notes	2001-07-06
1.24.0	Tuple	notes	2001-08-19
1.25.0	Thread, Bind	notes	2001-10-01
1.26.0	Common Factor, Preprocessor	notes	2001-11-30
1.27.0		notes	2002-02-05
1.28.0	Lambda, I/O State Saver	notes	2002-05-15
1.29.0	Date-Time, Dynamic Bitset, Format	notes	2002-10-10
1.30.0	Filesystem, Optional, Interval, MPL, Spirit	notes	2003-03-19

Version	New Libraries	Release Notes	Release Date
1.31.0	enable_if, Variant	notes	2004-01-26
1.32.0	Assignment, Minmax, Multi-Index, Numeric Conversion, Program Options, Range, Serialization, String, Tribool	notes	2004-11-19
1.33.0	Iostream, Hash, Parameter, Pointer Container, Wave	notes	2005-08-11
1.34.0	Foreach, Statechart, TR1, Typeof, Xpressive	notes	2007-05-12
1.35.0	Asio, Bimap, Circular Buffer, Function Types, Fusion, GIL, Interprocess, Intrusive, Math/Special Functions, Math/Statistical Distributions, MPI, System	notes	2008-03-29
1.36.0	Accumulators, Exception, Units, Unordered	notes	2008-08-14
1.37.0	Proto	notes	2008-11-03
1.38.0	Flyweight, ScopeExit, Swap	notes	2009-02-08
1.39.0	Signals2	notes	2009-05-02
1.40.0		notes	2009-08-27
1.41.0	Property Tree	notes	2009-11-17
1.42.0	Uuid	notes	2010-02-02
1.43.0	Functional/Factor, Functional/Forward	notes	2010-05-06
1.44.0	Meta State Machine, Polygon	notes	2010-08-13
1.45.0		notes	2010-11-19
1.46.0	Icl	notes	2011-02-21
1.46.1		notes	2011-03-12
1.47.0	Chrono, Geometry, Phoenix, Ratio	notes	2011-07-11
1.48.0	Container, Locale, Move	notes	2011-11-15
1.49.0	Heap	notes	2012-02-24
1.50.0	Algorithm, Functional/OverloadedFunction, LocalFunction, Utility/IdentityType	notes	2012-06-28

Version	New Libraries	Release Notes	Release Date
1.51.0	Context	notes	2012-08-20
1.52.0		notes	2012-11-05
1.53.0	Atomic, Coroutine, Lockfree, Multiprecision, Odeint	notes	2013-02-04
1.54.0	Log, TTI, Type Erasure	notes	2013-07-01
1.55.0	Predef	notes	2013-11-11
1.56.0	Align, TypeIndex	notes	2014-08-07
1.57.0		notes	2014-11-03
1.58.0	Endian, Sort	notes	2015-04-17
1.59.0	Convert, Coroutine2	notes	2015-08-13
1.60.0	VMD	notes	2015-12-17
1.61.0	Compute, DLL, Hana, Metaparse	notes	2016-05-13
1.62.0	Fiber, QVM	notes	2016-09-28
1.63.0		notes	2016-12-26
1.64.0	Process	notes	2017-04-19

Examples

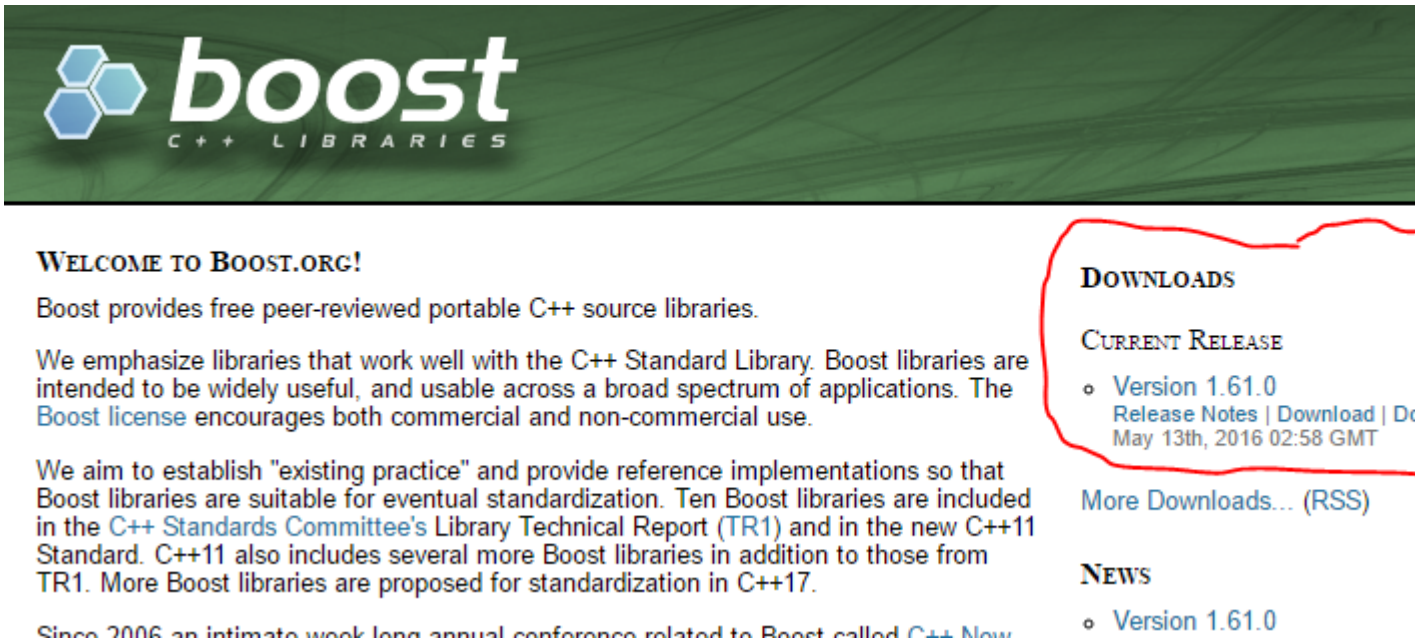
Installation or Setup

See Boost Getting Started.

Most of the Boost libraries are header-only, meaning that there's nothing you have to compile or link to.

Make sure you are getting the most recent version of Boost:

1. Visit www.boost.org
2. Look for the Current Release download. Currently, this links here.



3. Select the appropriate archive file for your operating system, and download.

Header-only libraries can then be used by simply including the respective header files.

A few Boost libraries require compilation:

- Boost.Chrono
- Boost.Context
- Boost.Filesystem
- Boost.GraphParallel
- Boost.IOStreams
- Boost.Locale
- Boost.MPI
- Boost.ProgramOptions
- Boost.Python
- Boost.Regex
- Boost.Serialization
- Boost.Signals
- Boost.System
- Boost.Thread
- Boost.Timer
- Boost.Wave

Also, the following libraries have components which must be compiled:

- Boost.DateTime
- Boost.Graph
- Boost.Math
- Boost.Random
- Boost.Test

- Boost.Exception

The source for Boost can be obtained through the download link on the site, which will re-direct to its SourceForge page for the latest version (1.61.0 as of July 2016). This can be unzipped (or un-tared, etc) to a directory (such as C:\local\boost_1_61_0). This directory can then be added to the include path for the software you are building. After this, you can include Boost headers in C++ files with `#include <boost/header/path.hpp>`.

The majority of the libraries in Boost are header-only. If you only need these then the above source distribution is all that is needed. However, if you need to use one of the libraries that requires a compiled binary to be built, you will need that as well.

On any system, the most reliable way to get the correct binaries is to build them yourself. These directions are somewhat different for Windows or Linux/Unix/POSIX.

On Windows with Visual Studio, an alternative to building the libraries yourself is to download pre-built libraries from Boost's SourceForge page (1.61.0 as of July 2016). On that page you can select an installer that will install a version for a specific Visual Studio build or the 7-zip file (boost_X_XX_X-bin-all-32-64.7z) that contains the binaries for all the supported Visual Studio versions. Either of these options includes the source/headers as well as the binaries, so there is no need to have downloaded the source distribution above. Once you have it, extract/install to a directory (such as C:\local\boost_1_61_0) and add that directory to your include path, then add the directory containing the binaries that correspond to your version of Visual Studio (e.g. C:\local\boost_1_61_0\lib32-msvc-12.0 for Visual Studio 2013 32-bit projects) to the library path.

Installing and Running Boost (Cygwin)

(Beginner level; IDE: CLion)

First, install boost from the Cygwin mirror: open the install exe, search for boost, install the packages.

After boost is installed: it will be located in `/usr/include/boost`. This is where everything is. All `#include` statements will be a path from the boost folder, as in: `#include <boost/archive/text_oarchive.hpp>`.

Once you include the boost files of your choice in your `.cpp` files, your code will still not compile in your IDE of choice until you **link** the libraries and tell cmake to search for your downloaded boost code.

In order to get cmake to search for your boost code,

```
find_package(Boost 1.60.0 COMPONENTS components_you_want)

# for example:
find_package(Boost 1.60.0 COMPONENTS serialization)
```

Then, include the directories: `include_directories(${Boost_INCLUDE_DIRS})`

Finally, add your executable and link the libraries:

```
add_executable(your_target ${SOURCE_FILES})
target_link_libraries(your_target ${Boost_LIBRARIES} -any_missing_boostlibs)
```

Before starting your program, avoid an error dump by testing to see if boost has been found before including anything or running your code:

```
if (Boost_FOUND)
    include_directories(${Boost_INCLUDE_DIRS})
    add_executable(YourTarget ${SOURCE_FILES})
    target_link_libraries(your_target ${Boost_LIBRARIES} -missing_libs)
endif()
```

I included `-missing_libs` because an error you may run into is that some boost library or another might not have been linked, and you must manually add it--for instance, the link I referenced earlier.

All together, a final CMakeLists.txt file might look something like:

```
cmake_minimum_required(VERSION 3.7)
project(your_project)

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11")
set(SOURCE_FILES main.cpp tmap.cpp tmap.h)
find_package(Boost 1.60.0 COMPONENTS serialization)

if(Boost_FOUND)
    include_directories(${Boost_INCLUDE_DIRS})
    add_executable(your_project ${SOURCE_FILES})
    target_link_libraries(your_project ${Boost_LIBRARIES})
endif()
```

Read Getting started with boost online: https://riptutorial.com/boost/topic/1167/getting-started-with-boost

Chapter 2: Async boost::process

Examples

Using all 3 pipes of a child process asynchronously.

```
#include <vector>
#include <string>
#include <boost/process.hpp>
#include <boost/asio.hpp>
#include <boost/process/windows.hpp>

int Run(
    const std::string& exeName,         ///< could also be UTF-16 for Windows
    const std::string& args,            ///< could also be UTF-16 for Windows
    const std::string& input,           ///< [in] data for stdin
    std::string& output,                ///< [out] data from stdout
    std::string& error                  ///< [out] data from stderr
)
{
    using namespace boost;

    asio::io_service ios;

    // stdout setup
    //
    std::vector<char> vOut(128 << 10);          // that worked well for my decoding app.
    auto outBuffer{ asio::buffer(vOut) };
    process::async_pipe pipeOut(ios);

    std::function<void(const system::error_code & ec, std::size_t n)> onStdOut;
    onStdOut = [&](const system::error_code & ec, size_t n)
    {
        output.reserve(output.size() + n);
        output.insert(output.end(), vOut.begin(), vOut.begin() + n);
        if (!ec)
        {
            asio::async_read(pipeOut, outBuffer, onStdOut);
        }
    };

    // stderr setup
    //
    std::vector<char> vErr(128 << 10);
    auto errBuffer{ asio::buffer(vErr) };
    process::async_pipe pipeErr(ios);

    std::function<void(const system::error_code & ec, std::size_t n)> onStdErr;
    onStdErr = [&](const system::error_code & ec, size_t n)
    {
        error.reserve(error.size() + n);
        error.insert(error.end(), vErr.begin(), vErr.begin() + n);
        if (!ec)
        {
            asio::async_read(pipeErr, errBuffer, onStdErr);
        }
    };
```

```
    // stdin setup
    //
    auto inBuffer{ asio::buffer(input) };
    process::async_pipe pipeIn(ios);

    process::child c(
        exeName + " " + args,                   // exeName must be full path
        process::std_out > pipeOut,
        process::std_err > pipeErr,
        process::std_in < pipeIn
    );

    asio::async_write(pipeIn, inBuffer,
        [&](const system::error_code & ec, std::size_t n)
        {
            pipeIn.async_close();                     //  tells the child we have no more data
        });

    asio::async_read(pipeOut, outBuffer, onStdOut);
    asio::async_read(pipeErr, errBuffer, onStdErr);

    ios.run();
    c.wait();
    return c.exit_code();                           // return the process' exit code.
}
```

IMPORTANT for boost 1.64

There is a bug/fix for boost 1.64, the bug only affects Windows, apparently.

reference: https://github.com/klemens-morgenstern/boost-process/issues/90 and https://github.com/klemens-morgenstern/boost-process/commit/74814e46c1614850a8e447fd689c21cf82f36ceb

in file `boost\process\detail\windows\async_pipe.hpp, line 79`:

```
    ~async_pipe()
    {
//fix
        //if (_sink .native()  != ::boost::detail::winapi::INVALID_HANDLE_VALUE_)
        //    ::boost::detail::winapi::CloseHandle(_sink.native());
        //if (_source.native() != ::boost::detail::winapi::INVALID_HANDLE_VALUE_)
        //    ::boost::detail::winapi::CloseHandle(_source.native());
        boost::system::error_code ec;
        close(ec);
//fix
    }
```

Read Async boost::process online: https://riptutorial.com/boost/topic/10822/async-boost--process

Chapter 3: Boost Accumulators Framework

Examples

Computing mean and variance

```
#include <iostream>
#include <boost/accumulators/accumulators.hpp>
#include <boost/accumulators/statistics/stats.hpp>
#include <boost/accumulators/statistics/mean.hpp>
#include <boost/accumulators/statistics/variance.hpp>

int main()
{
  using namespace boost::accumulators;
  accumulator_set<int, stats<tag::mean, tag::variance>> acc;

  for(int i = 1; i <= 6; i++)
    acc(i);

  std::cout << "mean=" << mean(acc) << ", variance=" << variance(acc) << '\n';
  // prints "mean=3.5, variance=2.91667"

  return 0;
}
```

Read Boost Accumulators Framework online: https://riptutorial.com/boost/topic/5718/boost-accumulators-framework

Chapter 4: BOOST- Compare Images using OpevCV

Introduction

This documentation explains how an external Image can be tested and compared with the output image of OpenCV. For example, To compare two blurred images and test if they both are same, we blur an original image in an external software (I used WiT Image Processing software) or just download any blurred image online- output1. Create a Win32 OpenCV project in Visual Studio. Read the original image as an input to the OpenCV. Blur the original image in OpenCV and compare with output1.

Examples

OpenCV code to read Images and compare

#include <opencv2/opencv.hpp> #include

using namespace cv; using namespace std;

int main(int argc, char** argv) { Mat image; image = imread("C:\Users\Development\Documents\Visual Studio 2013\Projects\ImageIn.bmp", CV_LOAD_IMAGE_GRAYSCALE); // Read the file

```
if (!image.data)                              // Check for invalid input
{
    cout << "Could not open or find the image" << std::endl;
    return -1;
}



Mat witout = imread("C:\\Users\\Development\\Documents\\Visual Studio
2013\\Projects\\ImageWitOut.bmp", CV_LOAD_IMAGE_GRAYSCALE);;
Mat cvout = Mat(image.size(), image.type(), Scalar(255));

imshow("witout", witout);
imshow("cvout", cvout);

Mat diff = (witout == cvout);

namedWindow("Difference", WINDOW_AUTOSIZE);// Create a window for display.
imshow("Difference", diff);                   // Show our image inside it.

waitKey(0);                                          // Wait for a keystroke in the window
return 0;
```

}

Read BOOST- Compare Images using OpevCV online:
https://riptutorial.com/boost/topic/10703/boost--compare-images-using-opevcv

Chapter 5: Boost Program Options

Examples

Basic Usage

Boost program options provides a simple and safe way to parse and handle command line arguments.

```
#include <boost/program_options.hpp>
#include <string>
#include <iostream>

int main(int argc, char** argv) {
  namespace po = boost::program_options;

  po::variables_map vm;
  po::options_description desc("Allowed Options");

  // declare arguments
  desc.add_options()
    ("name", po::value<std::string>()->required(), "Type your name to be greeted!");

  // parse arguments and save them in the variable map (vm)
  po::store(po::parse_command_line(argc, argv, desc), vm);

  std::cout << "Hello " << vm["name"].as<std::string>() << std::endl;

  return 0;
}
```

Compile and run with:

```
 $ g++ main.cpp -lboost_program_options && ./a.out --name Batman
Hello Batman
```

You can output a `boost::program_options::options_description` object to print the expected argument format:

```
std::cout << desc << std::endl;
```

would produce:

```
Allowed Options:
  --name arg                Type your name to be greeted!
```

Error Handling

`boost::program_options::notify` can be used to report any errors in the paramters passing

```
#include <boost/program_options.hpp>
#include <string>
#include <iostream>

int main(int argc, char** argv) {
  namespace po = boost::program_options;

  po::variables_map vm;
  po::options_description desc("Allowed Options");

  // declare options
  desc.add_options()
    ("name", po::value<std::string>()->required(), "Type your name to be greeted!");

  // parse arguments
  po::store(po::parse_command_line(argc, argv, desc), vm);

  // check arguments
  try {
    po::notify(vm);
  } catch (std::exception& e) {
    std::cout << "Error: " << e.what() << std::endl;
    std::cout << desc << std::endl;
    return 1;
  }

  // program logic
  std::cout << "Hello " << vm["name"].as<std::string>() << std::endl;

  return 0;
}
```

Passing illegal arguments produces helpful errors messages

```
 $ ./a.out
Error: the option '--name' is required but missing
Allowed Options:
  --name arg            Type your name to be greeted!
```

Default Values

A default valued command line argument can be specified easily:

```
// declare options
desc.add_options()
  ("name", po::value<std::string>()->required(), "Type your name to be greeted!")
  ("rank", po::value<std::string>()->default_value("Dark Knight"), "Your rank");
```

Its value is also added to the variable map:

```
  std::cout << "Hello " << vm["name"].as<std::string>() << " " << vm["rank"].as<std::string>()
<< std::endl;
```

The default value is shown in the description...

```
$ ./a.out
Error: the option '--name' is required but missing
Allowed Options:
  --name arg                Type your name to be greeted!
  --rank arg (=Dark Knight) Your rank
```

... and used if not specified...

```
$ ./a.out --name Batman
Hello Batman Dark Knight
```

... but can be overwritten at command line:

```
 $ ./a.out --name Batman --rank FlyingSquirrel
Hello Batman FlyingSquirrel
```

Switches

A switch is a command line argument which takes no value. It can be specified with:

```
desc.add_options()
  ("hidden", po::bool_switch()->default_value(false), "Hide your name");
```

And used with:

```
if (vm["hidden"].as<bool>())
   std::cout << "Hello *****" << std::endl;
```

from the command line:

```
 $ ./a.out --name Batman --hidden
Hello *****
```

and in the description it shows as:

```
Allowed Options:
  --name arg                Type your name to be greeted!
  --rank arg (=Dark Knight) Your rank
  --hidden                  Hide your name
```

Read Boost Program Options online: https://riptutorial.com/boost/topic/5359/boost-program-options

Chapter 6: boost String Algorithms Library

Remarks

Boost Documention on String Algrorithms

Examples

boost::split()

```
#include <iostream>
#include <vector>
#include <string>
#include <boost/algorithm/string.hpp>

using namespace std;

int main()
{

    // String to split

    string str = "You're supposed to see this!|NOT THIS!!!!!!";

    // Line container

    vector<string> lines;

    // Splits string

    boost::split(lines, str, boost::is_any_of("|"), boost::token_compress_on);

    // Outputs 1 half of the split string

    cout << lines.at(0).c_str() << endl;

    // Waits for input before program exits

    cin.get();

    return 0;
}
```

The following is the program in *psuedocode*:

Declare string *str* and **set** it to *"You're supposed to see this!|NOT THIS!!!!!!"*.

Declare vector *lines* of **type** string.

Split string *str* into vector *lines* if regex *"|"* is found.

Print object at index 0 in lines.

Get input.

Replace Algrorithms

boost::replace_all():

```
#include <iostream>
#include <string>
#include <boost/algorithm/string.hpp>

using namespace std;

int main()
{

    // String to replace characters in

    string str = "Darn you, Darn you to the 5th power!!!";

    // Replace "Darn" with "*CENSORED*"

    boost::replace_all(str, "Darn", "*CENSORED*");

    // Print str

    cout << str.c_str() << endl;

    // Waits for program to exit

    cin.get();

    return 0;
}
```

boost::replace_first():

```
#include <iostream>
#include <string>
#include <boost/algorithm/string.hpp>

using namespace std;

int main()
{

    // String to replace characters in

    string str = "Darn you, Darn you to the 5th power!!!";

    // Replace the first instance of "Darn" with "*CENSORED*"

    boost::replace_first(str, "Darn", "*CENSORED*");

    // Print str
```

```
    cout << str.c_str() << endl;

    // Waits for program to exit

    cin.get();

    return 0;
}
```

boost_replace_last():

```
#include <iostream>
#include <string>
#include <boost/algorithm/string.hpp>

using namespace std;

int main()
{

    // String to replace characters in

    string str = "Darn you, Darn you to the 5th power!!!";

    // Replace the last instance of "Darn" with "*CENSORED*"

    boost::replace_last(str, "Darn", "*CENSORED*");

    // Print str

    cout << str.c_str() << endl;

    // Waits for program to exit

   cin.get();

   return 0;
}
```

Case conversion methods

to_upper():

```
#include <iostream>
#include <string>
#include <boost/algorithm/string.hpp>

using namespace std;

int main()
{

    // String to convert characters to uppercase
```

```
    string str = "ThIS iS SUpPoSEd tO Be UpPeR CAsE.";

    // Convert characters in str to upper case

    boost::to_upper(str);

    // Print str

    cout << str.c_str() << endl;

    // Waits for program to exit

    cin.get();

    return 0;
}
```

to_lower():

```
#include <iostream>
#include <string>
#include <boost/algorithm/string.hpp>

using namespace std;

int main()
{

    // String to convert characters to lowercase

    string str = "ThIS iS SUpPoSEd tO Be LoWer CAsE.";

    // Convert characters in str to lower case

    boost::to_lower(str);

    // Print str

    cout << str.c_str() << endl;

    // Waits for program to exit

    cin.get();

    return 0;
}
```

Read boost String Algorithms Library online: https://riptutorial.com/boost/topic/7239/boost-string-algorithms-library

Chapter 7: Using boost.python

Examples

Introductory Example on Boost.Python

Things are easy when you have to use a C++ library in a Python project. Just you can use Boost.

First of all here is a list of components you need:

- A CMakeList.txt file, because you're going to use CMake.
- The C++ files of the C++ project.
- The python file - this is your python project.

Let's start with a small C++ file. Our C++ project has only one method which returns some string "This is the first try". Call it *CppProject.cpp*

```
char const *firstMethod() {
    return "This is the first try.";
}

BOOST_PYTHON_MODULE(CppProject) {
    boost::python::def("getTryString", firstMethod); // boost::python is the namespace
}
```

Have a CMakeLists.txt file a below:

```
cmake_minimum_required(VERSION 2.8.3)
FIND_PACKAGE(PythonInterp)
FIND_PACKAGE(PythonLibs)
FIND_PACKAGE(Boost COMPONENTS python)

INCLUDE_DIRECTORIES(${Boost_INCLUDE_DIRS} ${PYTHON_INCLUDE_DIRS})

PYTHON_ADD_MODULE(NativeLib CppProject)
FILE(COPY MyProject.py DESTINATION .) # See the whole tutorial to understand this line
```

By this part of the tutorial everything is so easy. you can import the library and call method in your python project. Call your python project *MyProject.py*.

```
import NativeLib
print (NativeLib.getTryString)
```

In order to run your project follow the instructions below:

- Create a directory with the name *build*.
- Enter into that directory.
- Give the command `cmake -DCMAKE_BUILD_TYPE=Release ..`
- `make`

- `python MyProject.py`. Now, you have to see the string which the method in your C++ project returns.

Wrapping std::vector in boost.python

If a function returns a `std::vector` type, and it is exposed to Python directly like

```
std::vector<float> secondMethod() {
    return std::vector<float>();
}

BOOST_PYTHON_MODULE(CppProject) {
    boost::python::def("getEmptyVec", secondMethod);
}
```

then when the functions gets called Python will tell you `No to_python (by-value) converter found for C++ type: std::vector<float, std::allocator<float> >`, because Python needs to know how to deal with `std::vector`.

Fortunately boost.python has provided a wrapper funciton for us in `vector_indexing_suite.hpp`. The returning value can be handled as a `FloatVec` object whose element can be accessed by the `[]` operator, by exposing the corresponding wrapper function as following.

```
std::vector<float> secondMethod() {
    return std::vector<float>();
}

BOOST_PYTHON_MODULE(CppProject) {
    // wrapper function
    class_<std::vector<float> >("FloatVec")
        .def(vector_indexing_suite<std::vector<float> >());
    boost::python::def("getEmptyVec", secondMethod);
}
```

The result can be further converted into a Python list or Numpy array simply by calling `list()` and `numpy.asarray()`.

Read Using boost.python online: https://riptutorial.com/boost/topic/6433/using-boost-python

Credits

S. No	Chapters	Contributors
1	Getting started with boost	bordeo, Community, Igor R., ildjarn, Justin, Null, teeks99
2	Async boost::process	Michaël Roy
3	Boost Accumulators Framework	Philipp Claßen
4	BOOST- Compare Images using OpevCV	DivyaMaheswaran
5	Boost Program Options	paul-g
6	boost String Algorithms Library	Zopesconk
7	Using boost.python	dontloo, Onur Tuna