Introduction to C++: Part 1

tutorial version 0.5

Brian Gregor

Research Computing Services

BOSTON UNIVERSITY

Getting started with the training room terminals

- Log on with your BU username
  - If you don’t have a BU username:
  - Username: Choose *tutm1-tutm18, tutn1-tutn18*
  - Password: on the board.

- On the desktop is a link to MobaXterm. Double click to open it.

BOSTON UNIVERSITY

Getting started on the SCC

- If you prefer to work on the SCC *and have your own account*, login using your account to the host **scc2.bu.edu**
  - On the room terminals there is a MobaXterm link on the desktop

- Load the Gnu C++ (g++) compiler and NetBeans modules:

```
module load gcc/5.3.0
module load gdb/7.11.1
module load java/1.8.0_92
module load netbeans/8.2
```

- Run to make a folder in your home directory and copy in the tutorial files:

```
/scratch/intro_to_cpp.sh
```

BOSTON UNIVERSITY

Getting started with your own laptop

- Go to:
  http://www.bu.edu/tech/support/research/training-consulting/live-tutorials/
  and download the Powerpoint or PDF copy of the unified presentation.

- Easy way to get there: Google “bu rcs tutorials” and it’s the 1st or 2nd link.

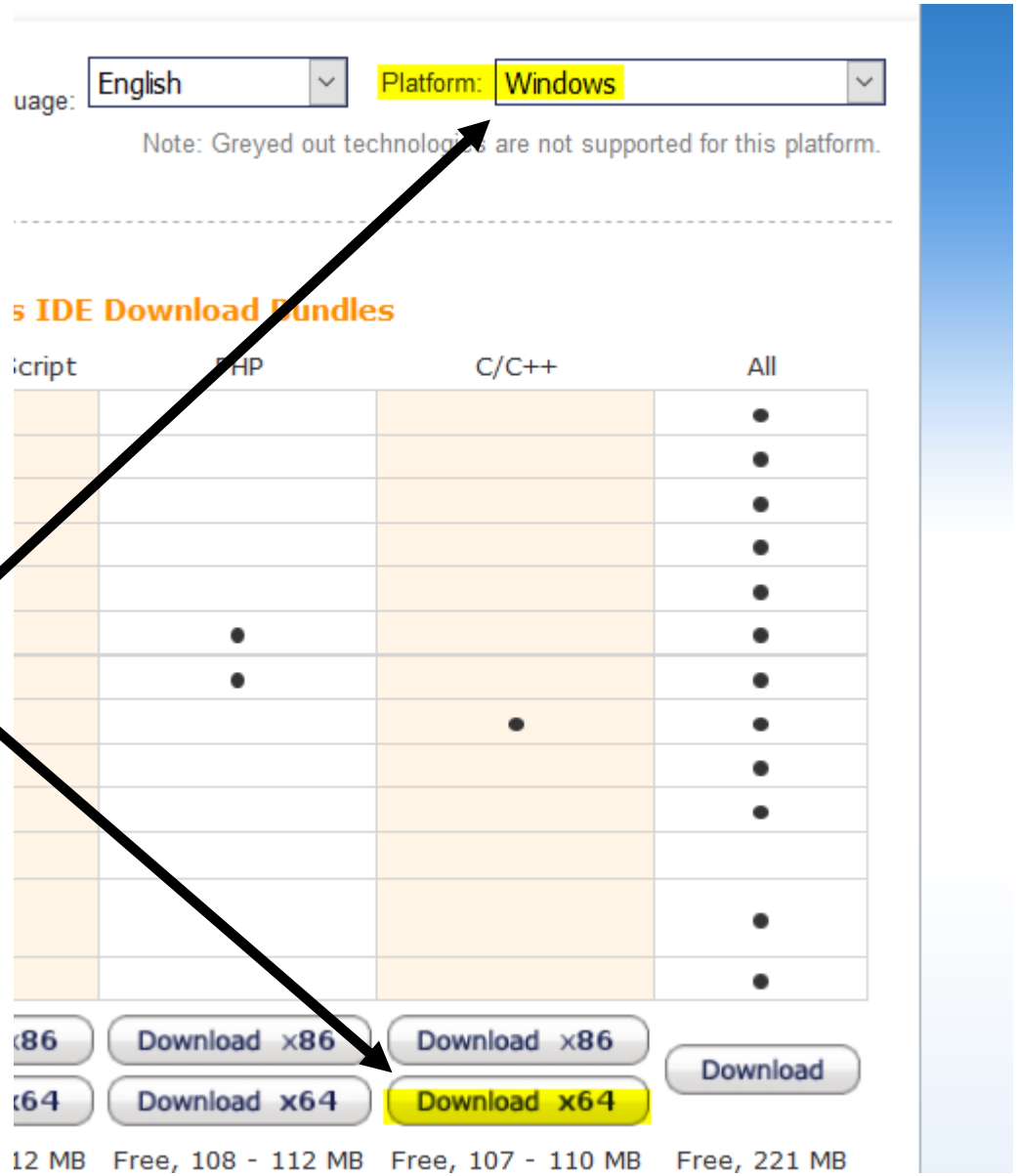
- Also download the “Additional Materials” file and unzip it to a convenient folder on your laptop.

BOSTON UNIVERSITY

Getting started with your own laptop

- Download the NetBeans 8.2 development environment:
  https://netbeans.org/downloads/

- In the upper right choose your operating system and download the link at the bottom of the C++ column.



BOSTON UNIVERSITY

Download a C/C++ compiler

- Mac OSX: You will need Apple's Xcode software with the **command line tools** installed.
  - This is the clang++ compiler, which is comparable to the g++ compiler.
- Linux: You can use the g++ compiler already installed.
- Windows: Things are a little more complicated…

gcc/g++ for Windows

- Visit: https://netbeans.org/community/releases/80/cpp-setup-instructions.html#mingw

- Follow the directions to install the Windows port of the g++ compiler.

- Skip the step about editing the PATH variable in Windows.

Tutorial Outline: All 4 Parts

- Part 1:
  - Intro to C++
  - Object oriented concepts
  - Write a first program
- Part 2:
  - Using C++ objects
  - Standard Template Library
  - Basic debugging
- Part 3:
  - Defining C++ classes
  - Look at the details of how they work
- Part 4:
  - Class inheritance
  - Virtual methods
  - Available C++ tools on the SCC

Tutorial Outline: Part 1

- Very brief history of C++
- Definition object-oriented programming
- When C++ is a good choice
- The NetBeans IDE
- Object-oriented concepts
- First program!
- Some C++ syntax
- Function calls
- Create a C++ class

Very brief history of C++

1962
Simula I was invented by Kristen Nygaard and Ole-Johan Dahl as a *simulation language*

1967
Simula 67 developed as the first *object-oriented* language

1969-1973
The C language was invented by Dennis Ritchie at Bell Labs

1972
D. Ritchie and Ken Thompson re-write the Unix OS in C

1979
Bjarne Stroustrop began developing "C with Classes"

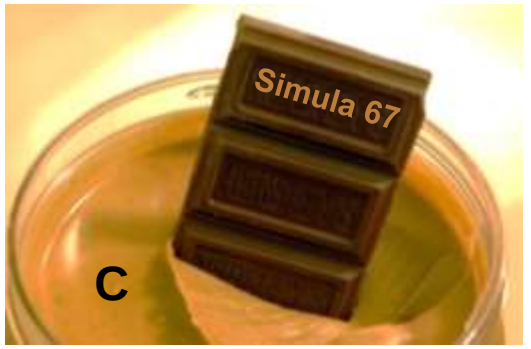
1983
"C with Classes" renamed to C++

1985
1st commercial C++ compiler, Cfront, released by AT&T

1989
C++ 2.0 standard released.

2011
Major update: C++11 standard released

2014
Minor update: C++14 released.



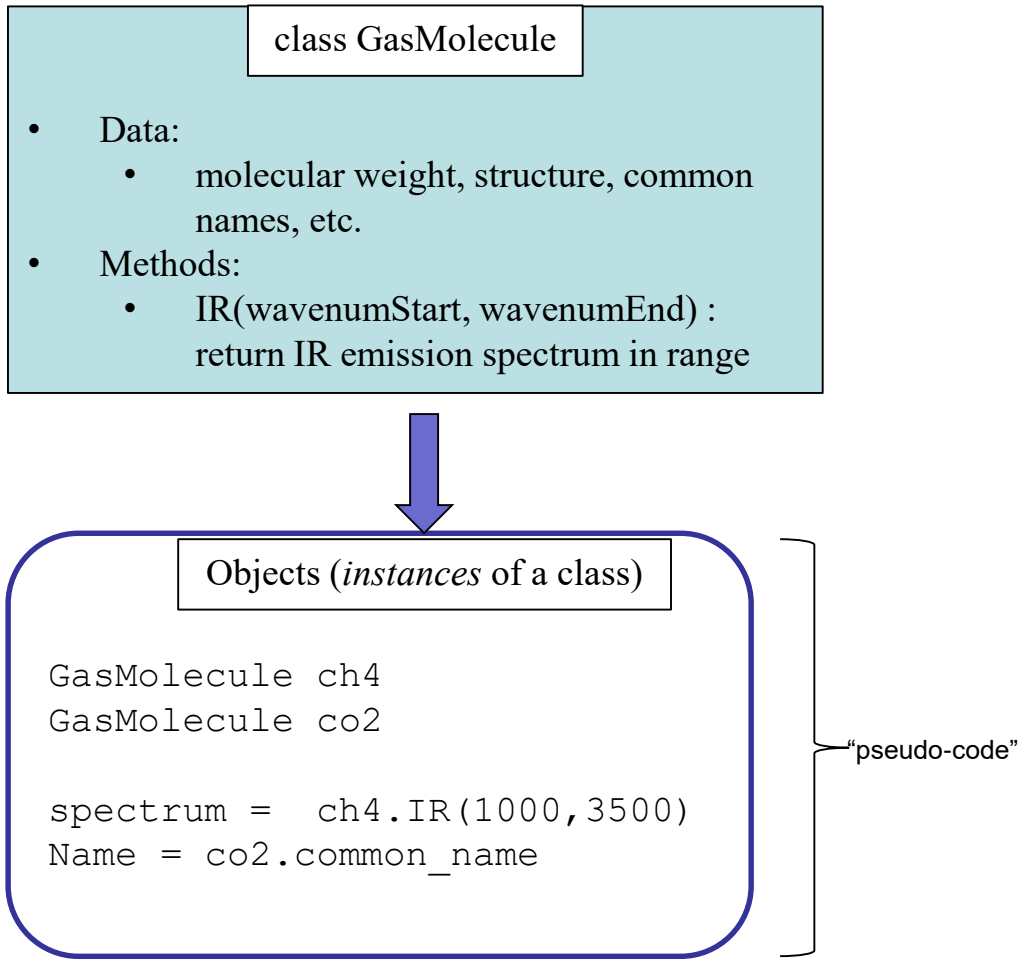
C++

BOSTON UNIVERSITY

For details more check out [A History of C++: 1979−1991]

Object-oriented programming

- Object-oriented programming (OOP) seeks to define a program in terms of the *things* in the problem:
  - files, molecules, buildings, cars, people, etc.
  - what they need to be created and used
  - what they can do

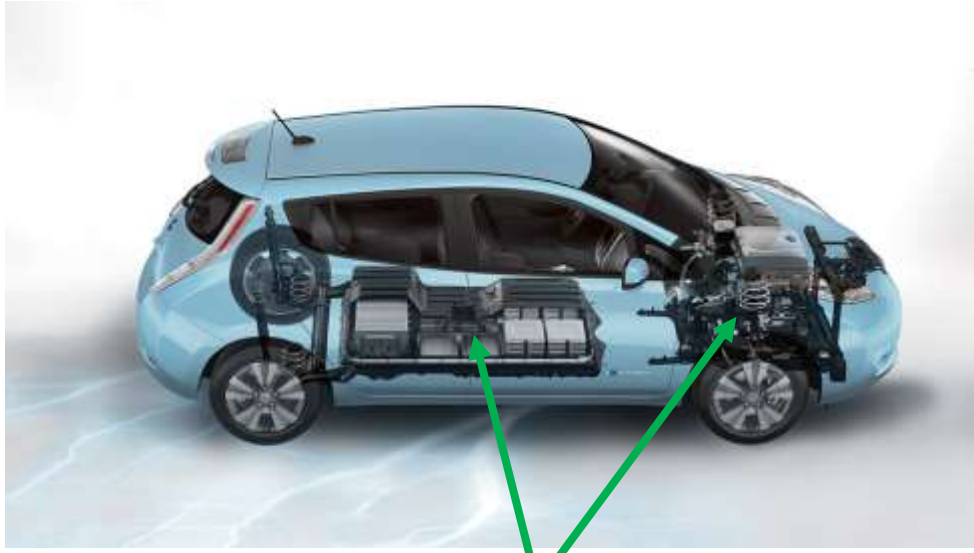


Object-oriented programming

- OOP defines *classes* to represent these things.
- Classes can contain data and methods (internal functions).
- Classes control access to internal data and methods. A *public* interface is used by external code when using the class.
- This is a highly effective way of modeling real world problems inside of a computer program.

"Class Car"

public interface



private data and methods

BOSTON UNIVERSITY

Characteristics of C++

> "Actually I made up the term 'object-oriented', and I can tell you I did not have C++ in mind."
>
> – Alan Kay (helped invent OO programming, the Smalltalk language, and the GUI)

- C++ is…
  - Compiled.
    - A separate program, the compiler, is used to turn C++ source code into a form directly executed by the CPU.
  - Strongly typed and unsafe
    - Conversions between variable types must be made by the programmer (strong typing) but can be circumvented when needed (unsafe)
  - C compatible
    - call C libraries directly and C code is nearly 100% valid C++ code.
  - Capable of very high performance
    - The programmer has a very large amount of control over the program execution
  - Object oriented
    - With support for many programming styles (procedural, functional, etc.)
- No automatic memory management (mostly)
  - The programmer is in control of memory usage

When to choose C++

> "If you're not at all interested in performance, shouldn't you be in the Python room down the hall?"
> — Scott Meyers (author of Effective Modern C++)

- Despite its many competitors C++ has remained popular for ~30 years and will continue to be so in the foreseeable future.

- Why?
  - Complex problems and programs can be effectively implemented
  - OOP works in the real world!
  - No other language quite matches C++'s combination of performance, libraries, expressiveness, and ability to handle complex programs.

- Choose C++ when:
  - Program performance matters
    - Dealing with large amounts of data, multiple CPUs, complex algorithms, etc.
  - Programmer productivity is less important
    - You'll get more code written in less time in a languages like Python, R, Matlab, etc.
  - The programming language itself can help organize your code
    - In C++ your objects can closely model elements of your problem
    - Complex data structures can be implemented
  - Access to libraries
    - Ex. Nvidia's CUDA Thrust library for GPUs
  - **Your group uses it already!**

NetBeans http://www.netbeans.org

- In this tutorial we will use the NetBeans integrated development environment (IDE) for writing and compiling C++
  - Run it right on the terminal or on the SCC (`module load netbeans/8.2.0`)
- About NetBeans
  - Originally developed at the Charles University in Prague, then by Sun Microsystems, then by Oracle, now part of the Apache Software Foundation.
  - cross-platform: supported on Mac OSX, Linux, and Windows
  - Oriented towards Java but also supports C and C++.
  - Short learning curve compared with other IDEs such as Eclipse or Visual Studio

- Generates its own *Makefiles* and builds with *make*, standard tools for building software.

IDE Advantages

- Handles build process for you
- Syntax highlighting and live error detection
- Code completion (fills in as you type)
- Creation of files via templates
- Built-in debugging
- Code refactoring (ex. Change a variable name everywhere in your code)
- Higher productivity than plain text editors!

IDEs available on the SCC

- NetBeans(used here)
- geany – a minimalist IDE, simple to use
- Eclipse – a highly configurable, adaptable IDE. Very powerful but with a long learning curve
- Spyder – Python only, part of Anaconda
- Emacs – The one and only.

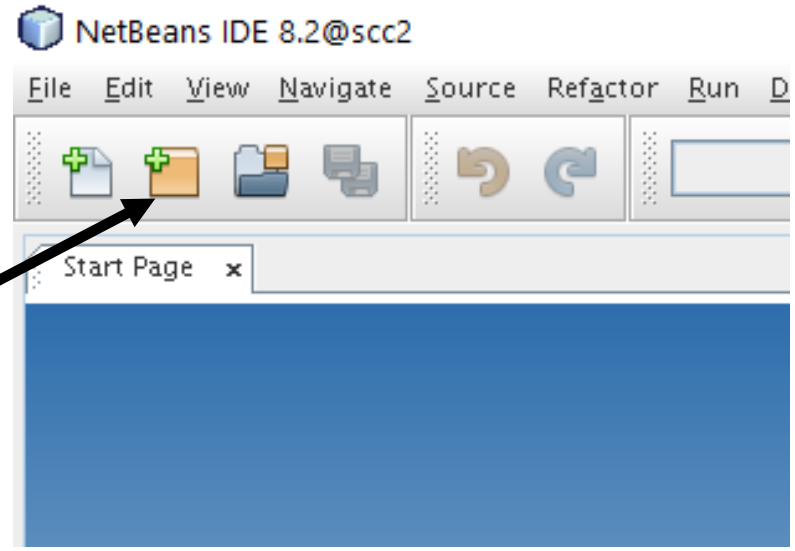
Some Others

- Xcode for Mac OSX
- Visual Studio for Windows
- Code::Blocks (cross platform)

Opening NetBeans

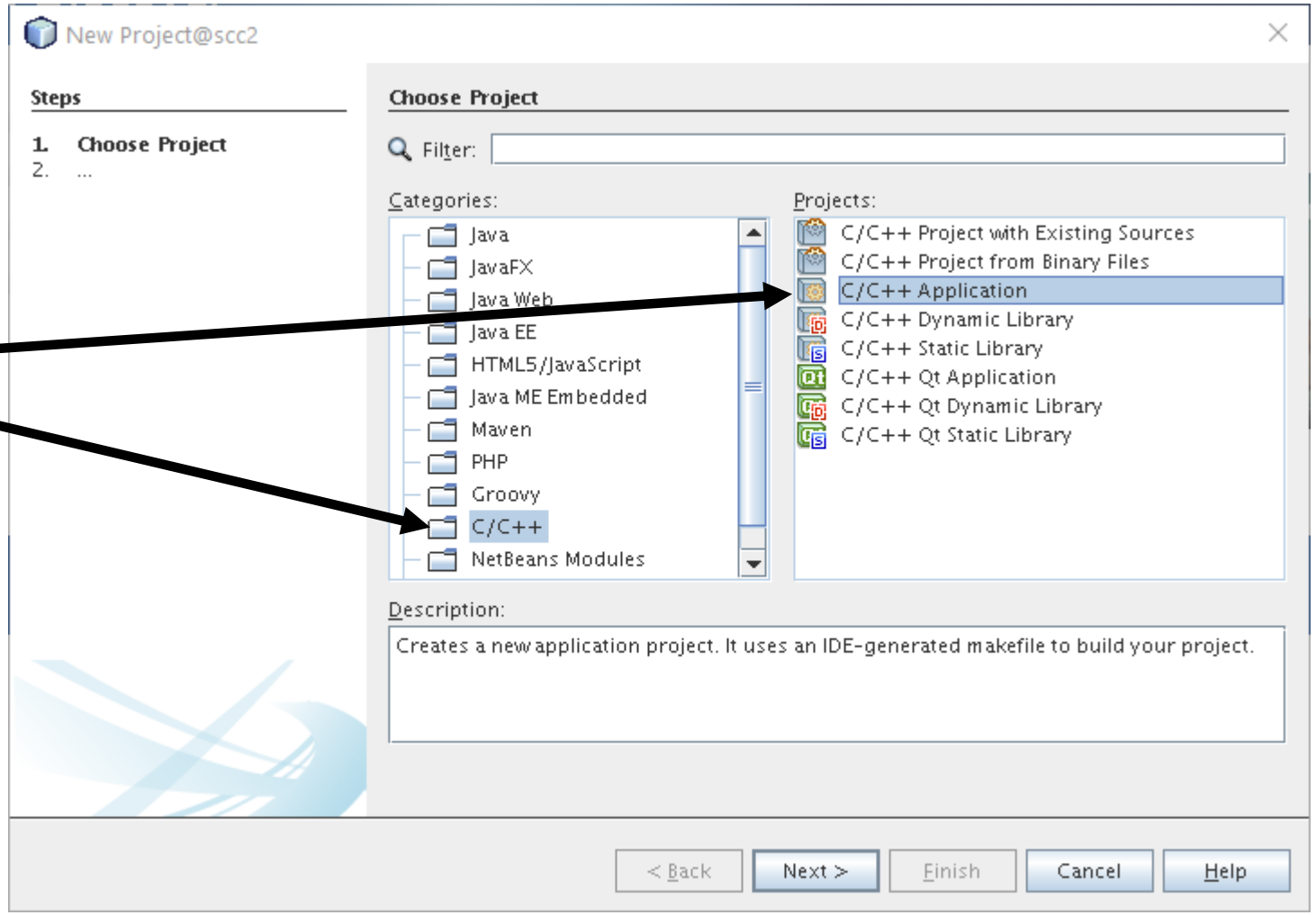
- Open NetBeans
  - click icon on OSX or Windows
  - Type *netbeans* on the SCC or Linux

- Create a new project under the File Menu or by clicking on the icon



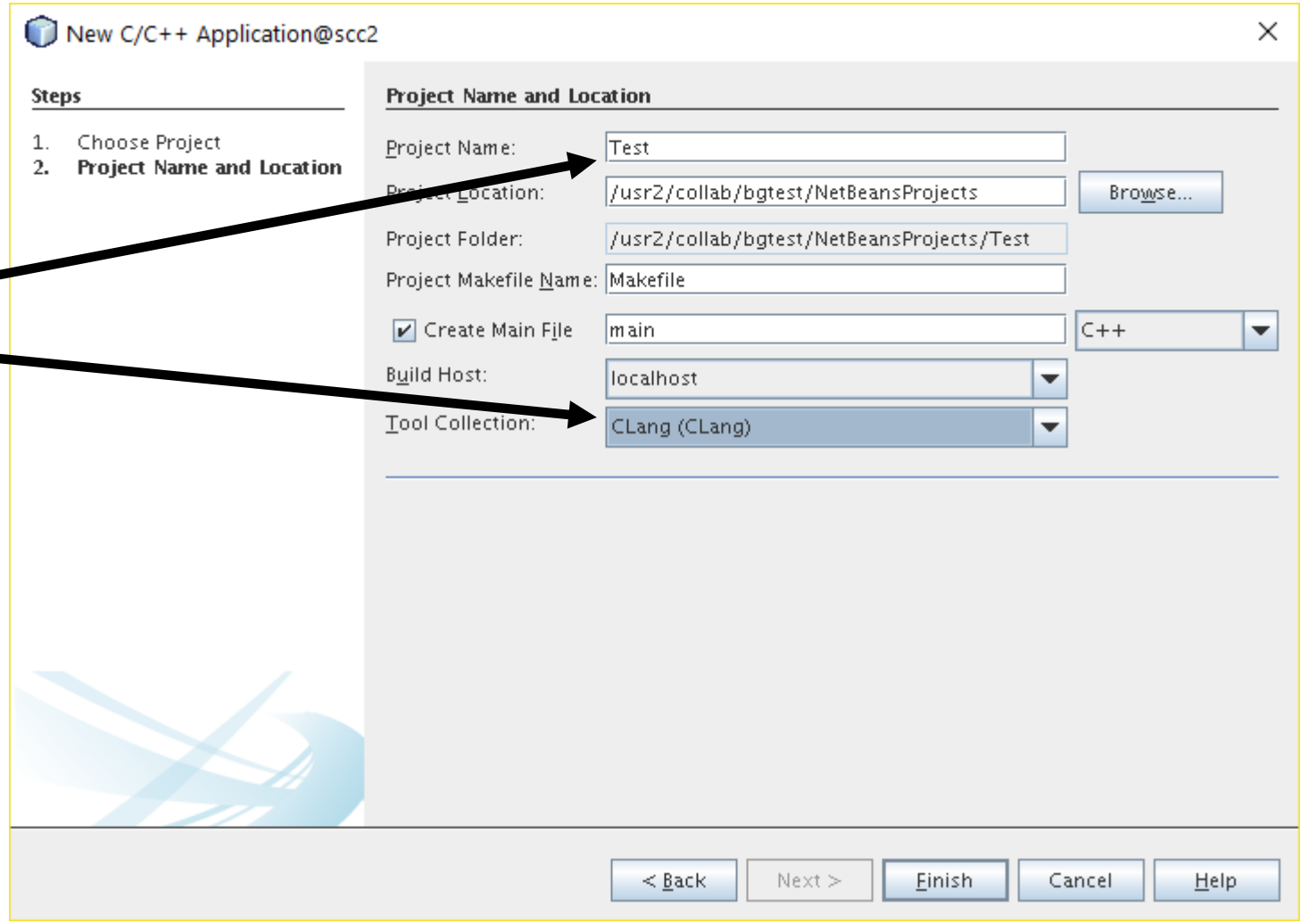
BOSTON UNIVERSITY

- Choose the *C/C++* category and *C/C++ Application*
- Click the *Next* button.



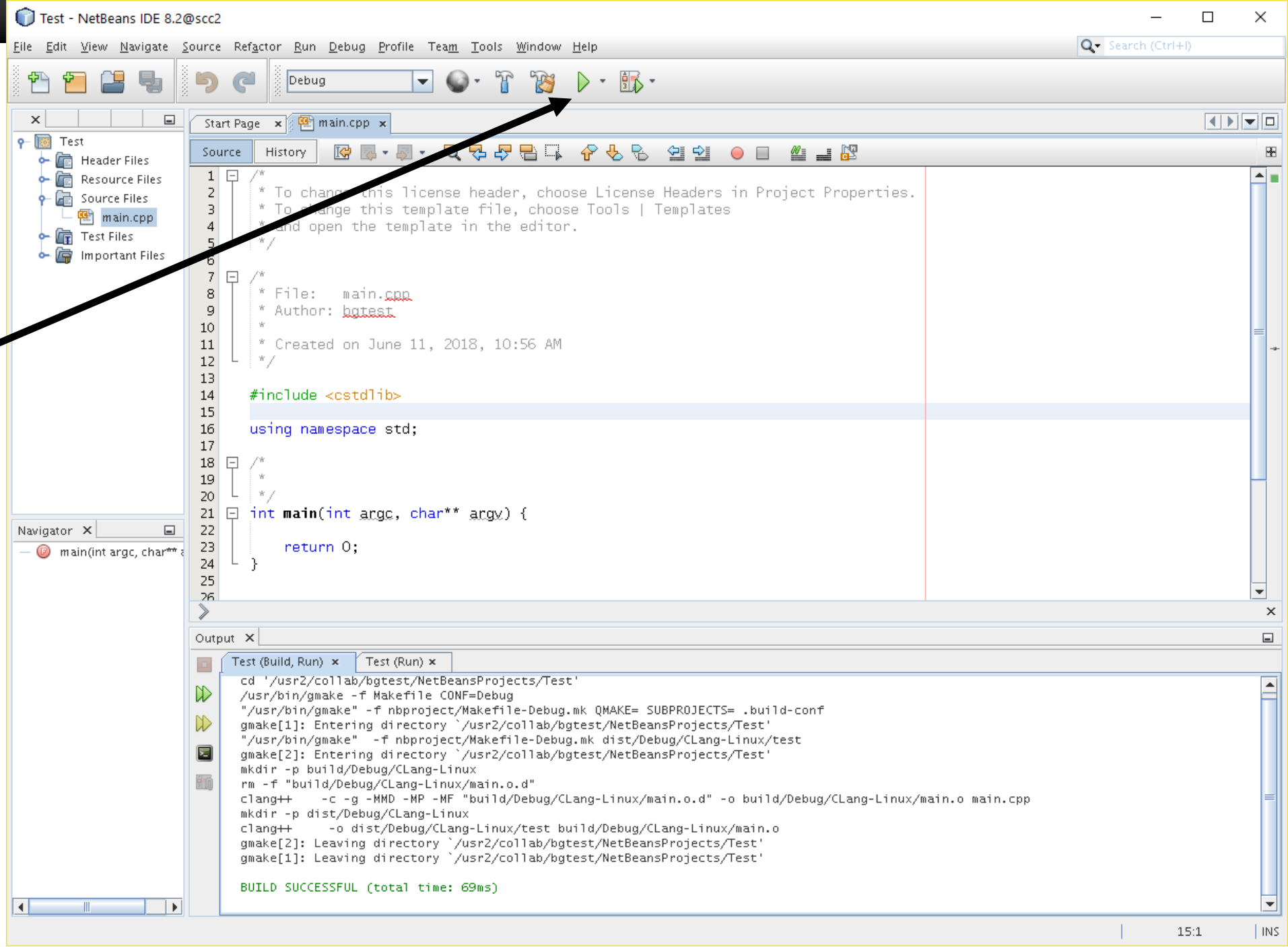
BOSTON UNIVERSITY

- Give the project the name *Test.*
- Under Tool Collection choose Clang or Gnu (depending on what is shown)
- Click *Finish*



BOSTON UNIVERSITY

- Check that everything works.
- Click the green triangle to compile & run the program.

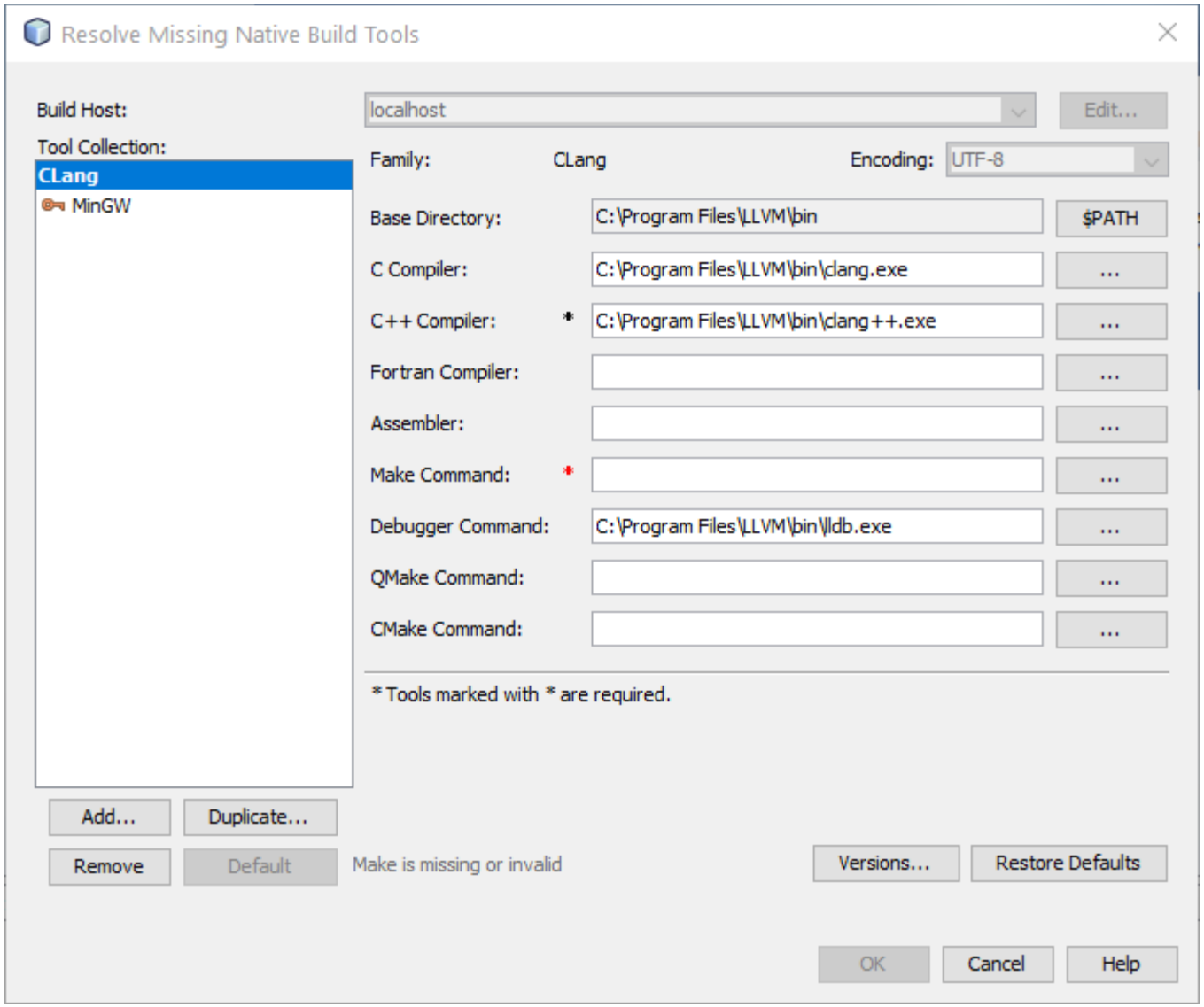


BOSTON UNIVERSITY

- Windows will probably ask to have some tools specified.

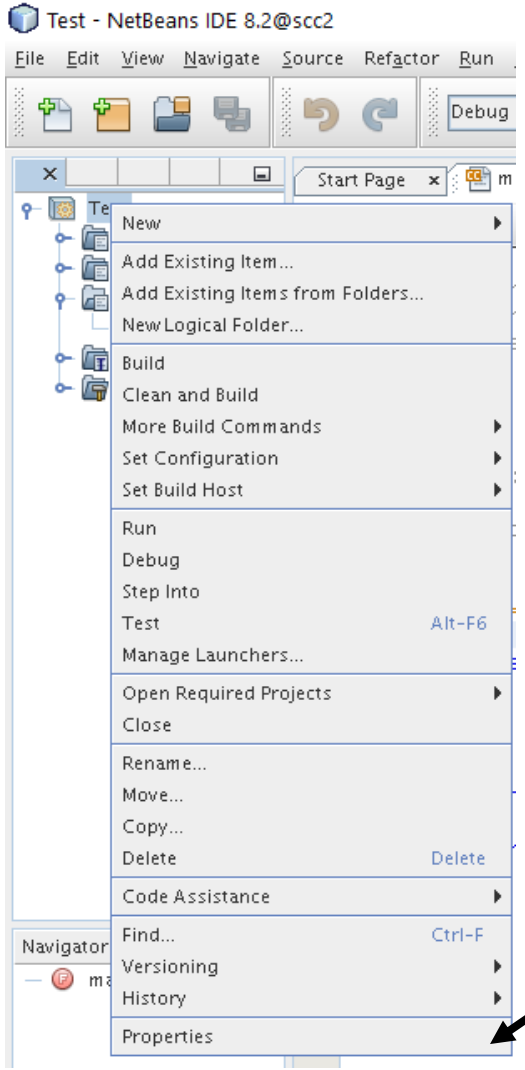
- Set the base directory for Clang to:
- C:\Program Files\LLVM\bin
- Debugger to:
- C:\Program Files\LLVM\bin\lldb.exe

- The *make* program is found in:
- C:\Program Files (x86)\GnuWin32\bin

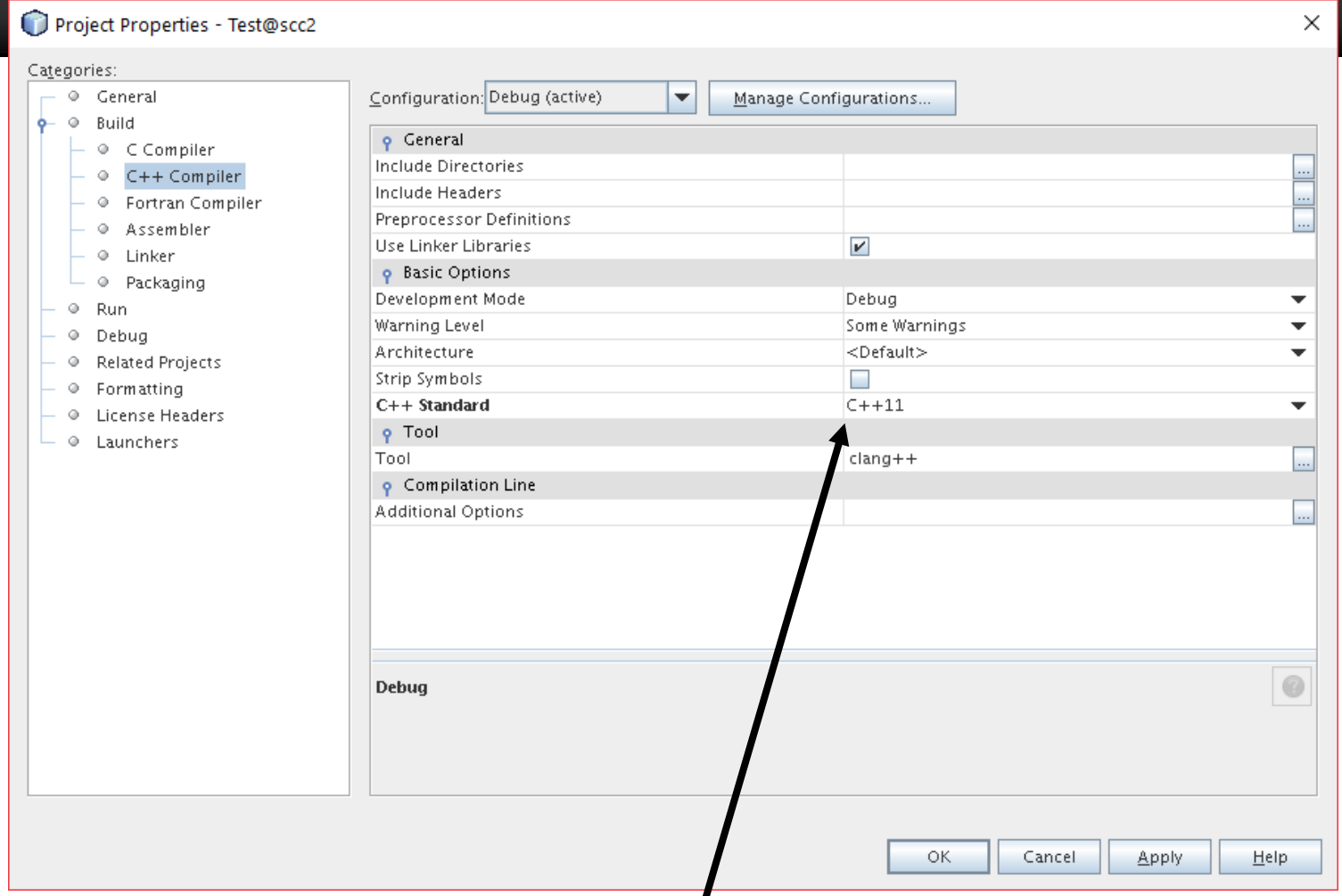


BOSTON UNIVERSITY

Enable C++11 standard



- Right-click on your project name and choose *Properties*



- Click the *C++ Compiler* category and choose *C++11* option under the *C++ Standard* menu.

Make the program do something….

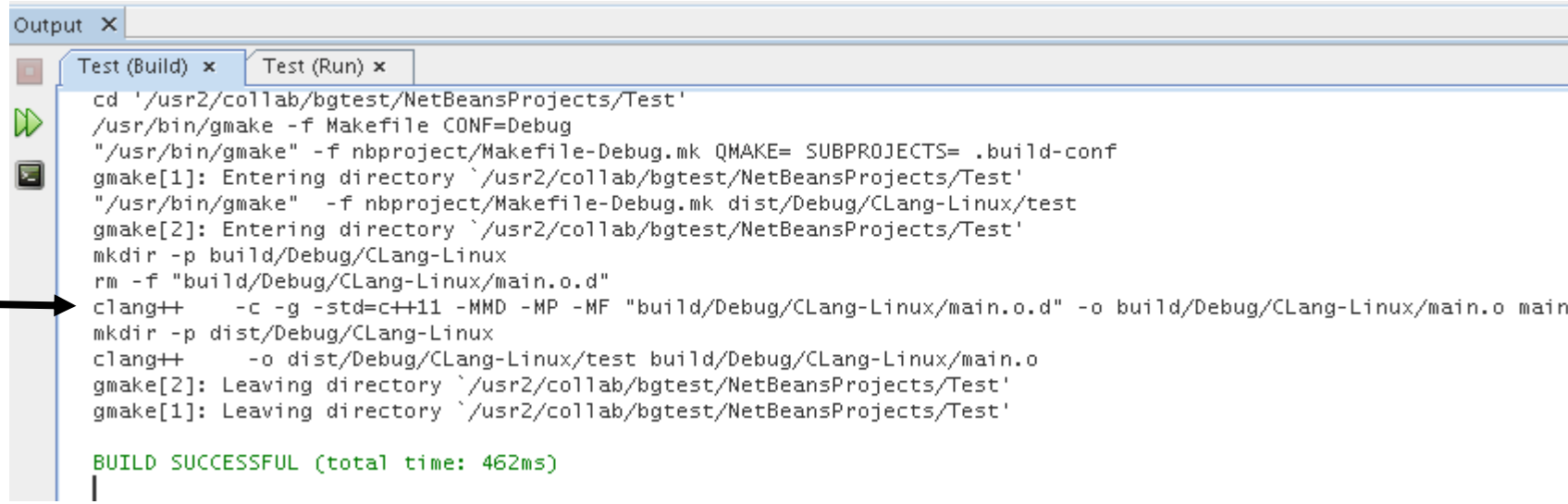
- Edit the main.cpp program in your Test project to look like the code to the right.
- This is text – it can be copied and pasted from the presentation…but it's better to type it.
- When done, click the green triangle again to run the updated program.

```
#include <iostream>
using namespace std;

int main(int argc, char** argv) {

    cout << "Hello World!" << endl ;

    return 0;
}
```

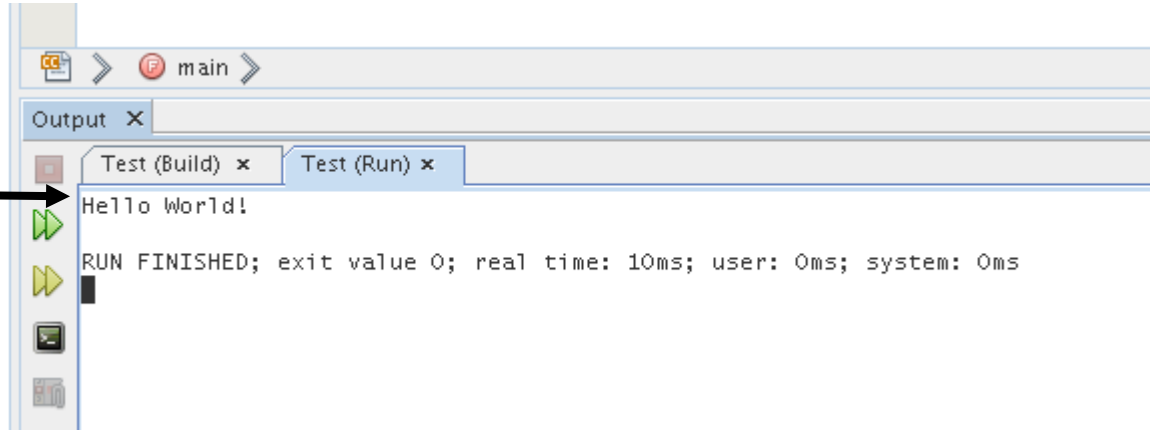
BOSTON UNIVERSITY

Hello, World!

- Build messages

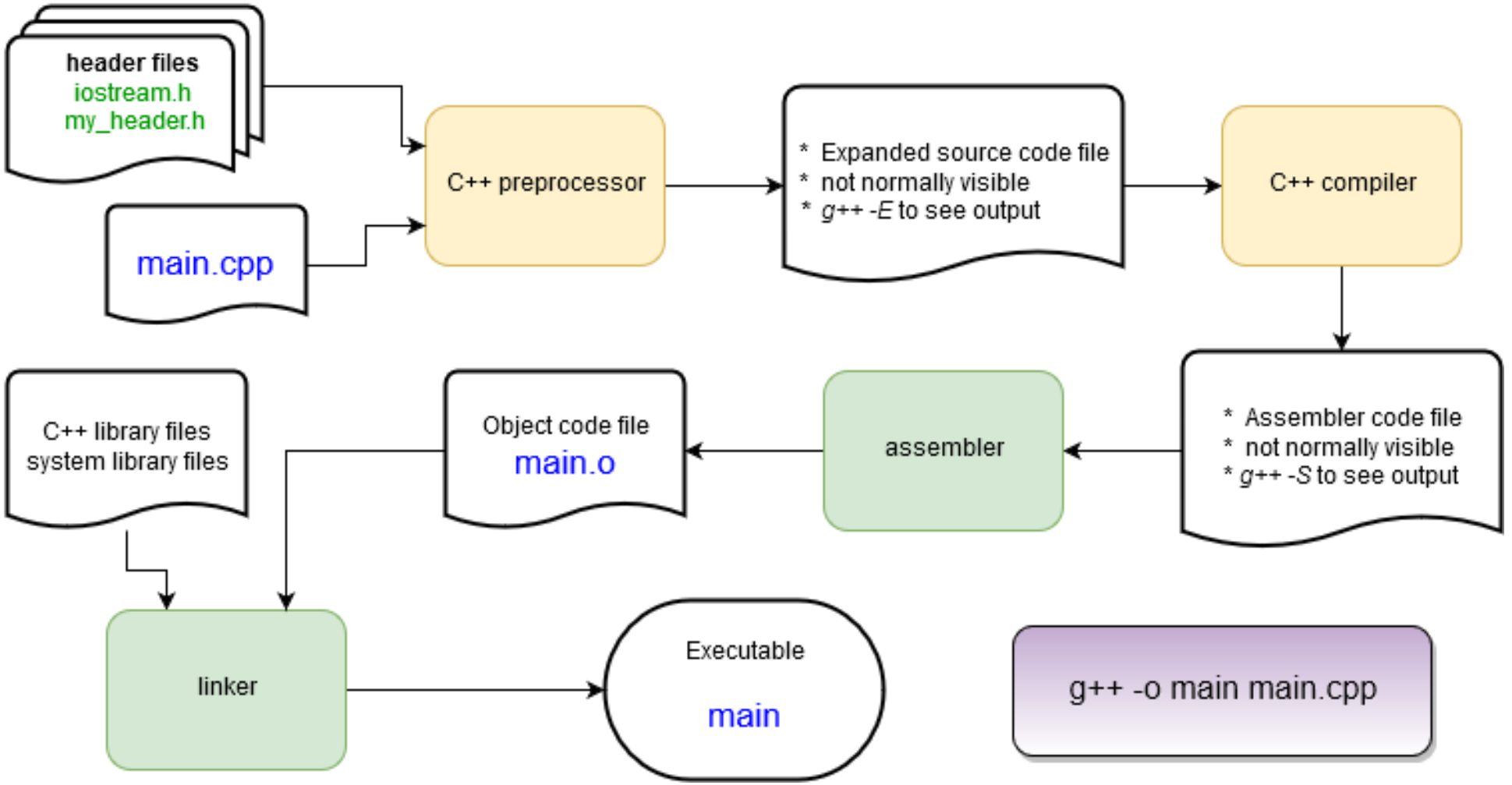


- Run output



BOSTON UNIVERSITY

Behind the Scenes: The Compilation Process



Hello, World! explained

```
#include <iostream>
using namespace std;

/*
 *
 */
int main(int argc, char** argv) {

    cout << "Hello World!" << endl ;

    return 0;
}
```

The *main* routine – the start of **every** C++ program! It returns an integer value to the operating system and (in this case) takes arguments to allow access to command line arguments.

The **return** statement returns an integer value to the operating system after completion. 0 means “no error”. C++ programs **must** return an integer value.

Hello, World! explained

```
#include <iostream>
using namespace std;

/*
 *
 */
int main(int argc, char** argv) {

    cout << "Hello World!" << endl ;

    return 0;
}
```

- loads a *header* file containing function and class definitions
- Loads a *namespace* called *std*.
- Namespaces are used to separate sections of code for programmer convenience. To save typing we'll always use this line in this tutorial.
- *cout* is the *object* that writes to the stdout device, i.e. the console window.
- It is part of the C++ standard library.
- Without the "using namespace std;" line this would have been called as *std::cout*. It is defined in the *iostream* header file.
- << is the C++ *insertion operator*. It is used to pass characters from the right to the object on the left.
- *endl* is the C++ newline character.

Header Files

- C++ (along with C) uses *header files* as to hold definitions for the compiler to use while compiling.
- A source file (file.cpp) contains the code that is compiled into an object file (file.o).
- The header (file.h) is used to tell the compiler what to expect when it assembles the program in the linking stage from the object files.

- Source files and header files can refer to any number of other header files.

- When compiling the *linker* connects all of the object (.o) files together into the executable.

Make some changes

- Let's put the message into some variables of type *string* and print some numbers.
- Things to note:
  - Strings can be concatenated with a + operator.
  - No messing with null terminators or *strcat()* as in C
- Some string notes:
  - Access a string character by brackets or function:
    - msg[0] → "H" or msg.at(0) → "H"
    - C++ strings are *mutable* – they can be changed in place.
- Re-run and check out the output.

```
#include <iostream>

using namespace std;

int main()
{
    string hello = "Hello";
    string world = "world!";
    string msg = hello + " " + world ;
    cout <<  msg << endl;
    msg[0] = 'h';
    cout <<  msg << endl;
    return 0;
}
```

BOSTON UNIVERSITY

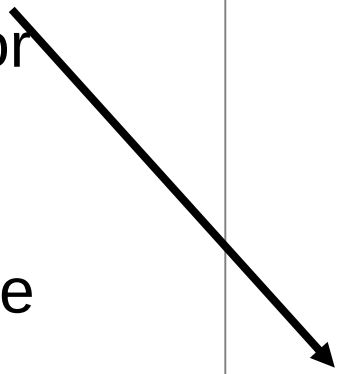
A first C++ class: *string*

- *string* is not a basic type (more on those later), it is a class.
- `string hello` creates an *instance* of a string called “hello”.
- `hello` is an object.
- Remember that a class defines some data and a set of functions (methods) that operate on that data.
- Let’s use NetBeans to see what some of these methods are….

```
#include <iostream>

using namespace std;

int main()
{
    string hello = "Hello";
    string world = "world!";
    string msg = hello + " " + world ;
    cout <<  msg << endl;
    msg[0] = 'h';
    cout <<  msg << endl;
    return 0;
}
```

A first C++ class: *string*

- Update the code as you see here.
- After the last character is entered NetBeans will display a large number of *methods* defined for the msg object.
- If you click or type something else just delete and re-type the last character.
- Ctrl-space will force the list to appear.



```
#include <iostream>

using namespace std;

int main()
{
    string hello = "Hello";
    string world = "world!";
    string msg = hello + " " + world ;
    cout <<  msg << endl;
    msg[0] = 'h';
    cout <<  msg << endl;

    msg.

    return 0;
}
```

BOSTON UNIVERSITY

A first C++ class: *string*

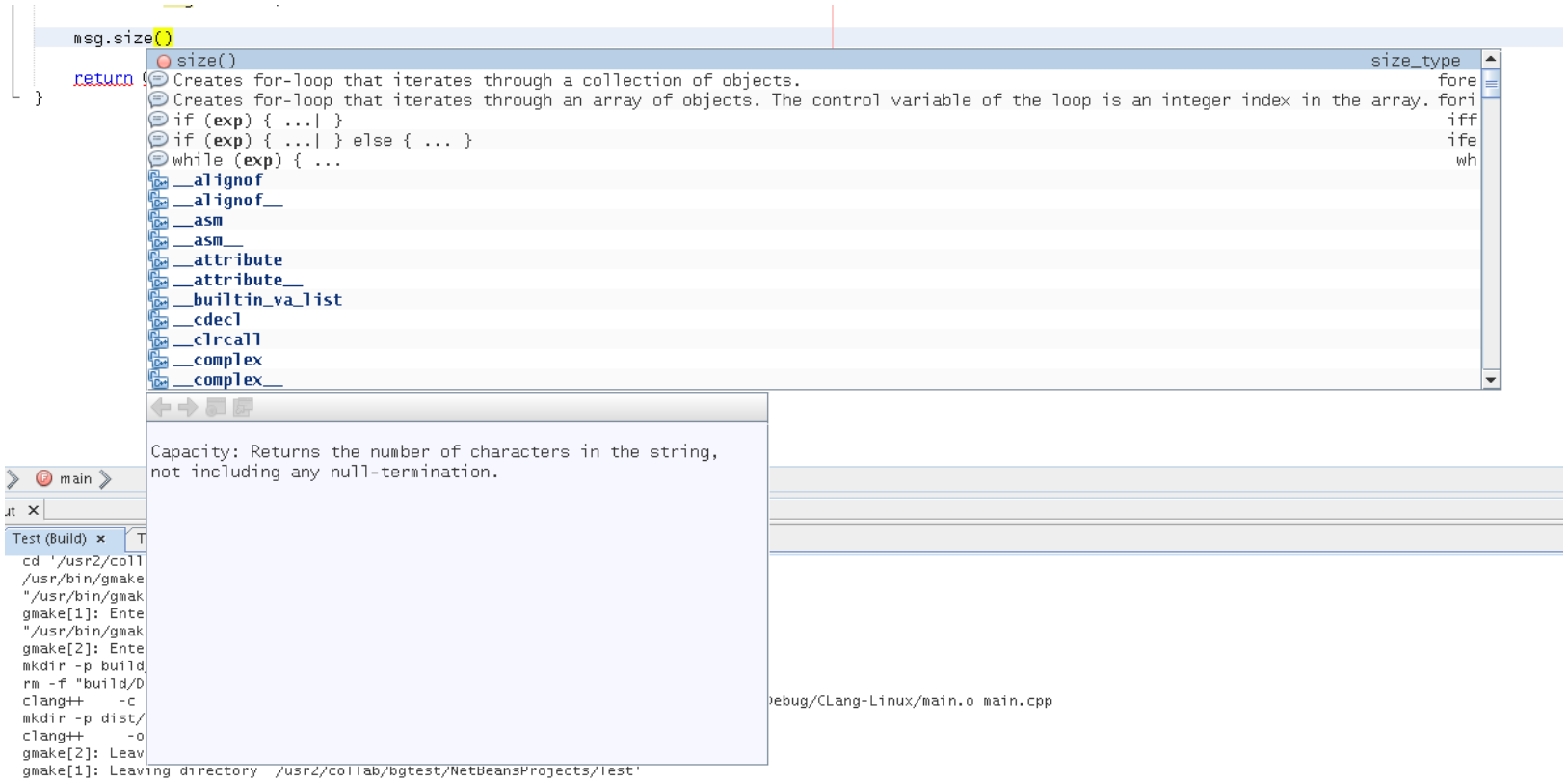
```
    cout <<  msg << endl;
    msg[0] = 'h';
    cout <<  msg << endl;

    msg.
}
```

```
npos                                                                                                                                const size_type
append(const basic_string& __str)                                                                                                     basic_string&
append(const basic_string& __str, unsigned char __pos, unsigned char __n)                                                             basic_string&
append(const char* __s, unsigned char __n)                                                                                            basic_string&
append(const char* __s)                                                                                                               basic_string&
append(unsigned char __n, char __c)                                                                                                   basic_string&
append(initializer_list<char> __l)                                                                                                    basic_string&
append<class _InputIterator,typename=std::_RequireInputIter<_InputIterator>>(_InputIterator __first, _InputIterator __last)          basic_string&
assign(const basic_string& __str)                                                                                                     basic_string&
assign(basic_string&& __str)                                                                                                          basic_string&
assign(const basic_string& __str, unsigned char __pos, unsigned char __n)                                                             basic_string&
assign(const char* __s, unsigned char __n)                                                                                            basic_string&
assign(const char* __s)                                                                                                               basic_string&
assign(unsigned char __n, char __c)                                                                                                   basic_string&
assign<class _InputIterator,typename=std::_RequireInputIter<_InputIterator>>(_InputIterator __first, _InputIterator __last)          basic_string&
assign(initializer_list<char> __l)                                                                                                    basic_string&
at(unsigned char __n)                                                                                                          _Alloc::value_type
```

```
Value returned by various member functions when they fail.
```

A first C++ class: *string*

- Start typing “msg.size()” until it appears in the list.
- Once it’s highlighted (or you scroll to it) press the Tab key to auto-enter it.
- An explanation appears below.
- “Returns the number of characters in a string not including any null-termination.”



BOSTON UNIVERSITY

A first C++ class: *string*

- Tweak the code to print the number of characters in the string, build, and run it.
- size() is a *public* method, usable by code that creates the object.
- The internal tracking of the size and the storage itself is *private*, visible only inside the string class source code.

```
#include <iostream>

using namespace std;

int main()
{
    string hello = "Hello" ;
    string world = "world!" ;
    string msg = hello + " " + world ;
    cout <<  msg << endl ;
    msg[0] = 'h';
    cout <<  msg << endl ;

    cout << msg.size() << endl ;

    return 0;
}
```

- *cout* prints integers without any modification!

BOSTON UNIVERSITY

- Note: while the string class has a **huge** number of methods your typical C++ class has far fewer!

Break your code.

- Remove a semi-colon. Re-compile. What messages do you get from the compiler and NetBeans?
- Fix that and break something else. Capitalize *string* → *String*
- C++ can have elaborate error messages when compiling. Experience is the only way to learn to interpret them!
- Fix your code so it still compiles and then we'll move on…

Basic Syntax

- C++ syntax is very similar to C, Java, or C#. Here's a few things up front and we'll cover more as we go along.
- Curly braces are used to denote a *code block* (like the main() function):

```
{ … some code … }
```

- Statements end with a semicolon:

```
int a ;
a = 1 + 3 ;
```

- Comments are marked for a single line with a ***//*** or for multilines with a pair of ***/**** and ****/*** :

```
// this is a comment.
/* everything in here
        is a comment */
```

- Variables can be declared at any time in a code block.

```
void my_function() {
    int a ;
    a=1 ;
    int b;
}
```

- Functions are sections of code that are called from other code. Functions always have a return argument type, a function name, and then a list of arguments separated by commas:

```
int add(int x, int y) {
    int z = x + y ;
    return z ;
}
```

```
// No arguments? Still need ()
void my_function() {
        /* do something...
           but a void value means the
           return statement can be skipped.*/
}
```

- A *void* type means the function does not return a value.

- Variables are declared with a type and a name:

```
// Specify the type
int x = 100;
float y;
vector<string> vec ;
// Sometimes types can be inferred
auto z = x;
```

- A sampling of arithmetic operators:
  - Arithmetic: + - * / % ++ --
  - Logical: && (AND) ||(OR) !(NOT)
  - Comparison: == > < >= <= !=

- Sometimes these can have special meanings beyond arithmetic, for example the “+” is used to concatenate strings.

- What happens when a syntax error is made?
  - The compiler will complain and refuse to compile the file.
  - The error message *usually* directs you to the error but sometimes the error occurs before the compiler discovers syntax errors so you hunt a little bit.

Built-in (aka primitive or intrinsic) Types

- "primitive" or "intrinsic" means these types are not objects.
  - They have no methods or internal hidden data.
- Here are the most commonly used types.
- Note: The exact bit ranges here are **platform and compiler dependent**!
  - Typical usage with PCs, Macs, Linux, etc. use these values
  - Variations from this table are found in specialized applications like embedded system processors.

Name	Name	Value
char	unsigned char	8-bit integer
short	unsigned short	16-bit integer
int	unsigned int	32-bit integer
long	unsigned long	64-bit integer
bool		true or false

Name	Value
float	32-bit floating point
double	64-bit floating point
long long	128-bit integer
long double	128-bit floating point

http://www.cplusplus.com/doc/tutorial/variables/

Need to be sure of integer sizes?

- In the same spirit as using *integer(kind=8)* type notation in Fortran, there are type definitions that exactly specify exactly the bits used. These were added in C++11.
- These can be useful if you are planning to port code across CPU architectures (ex. Intel 64-bit CPUs to a 32-bit ARM on an embedded board) or when doing particular types of integer math.
- For a full list and description see: http://www.cplusplus.com/reference/cstdint/

#include <cstdint>

Name	**Name**	**Value**
int8_t	uint8_t	8-bit integer
int16_t	uint16_t	16-bit integer
int32_t	uint32_t	32-bit integer
int64_t	uint64_t	64-bit integer

Reference and Pointer Variables

```
string hello = "Hello";

string *hello_ptr = &hello;

string &hello_ref = hello;
```

The object *hello* occupies some computer memory.

A **pointer** to the hello object string. *hello_ptr* is assigned the memory address of object *hello* which is accessed with the “&” syntax.

hello_ref is a **reference** to a string. The *hello_ref* variable is assigned the memory address of object *hello* automatically.

- Variable and object values are stored in particular locations in the computer’s memory.
- Reference and pointer variables **store the memory location of other variables.**
- Pointers are found in C. References are a C++ variation that makes pointers easier and safer to use.
- More on this topic later in the tutorial.

Type Casting

- C++ is strongly typed. It will auto-convert a variable of one type to another where it can.

```
short x = 1 ;
int y = x ;    // OK
string z = y ; // NO
```

- Conversions that don't change value work as expected:
  - increasing precision (float → double) or integer → floating point of at least the same precision.

- Loss of precision usually works fine:
  - 64-bit double precision → 32-bit single precision.
  - But…be careful with this, if the larger precision value is too large the result might not be what you expect!

Type Casting

- C++ allows for C-style type casting with the syntax: `(new type) expression`

```
double x = 1.0 ;
int y = (int) x ;
float z = (float) (x / y) ;
```

- But when using C++ it's best to stick with deliberate type casting using the **4** different ways that are offered…

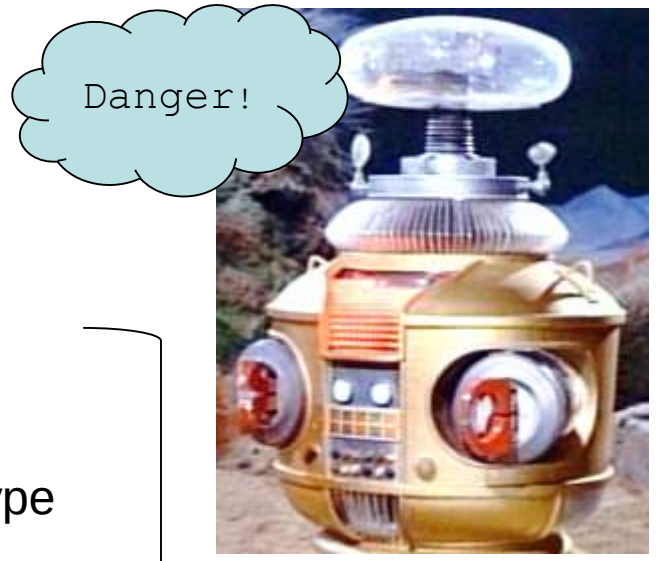
Type Casting

```
double d = 1234.56 ;
float f = static_cast<float>(d) ;
// same as
float g = (float) d ;
// same as
float h = d ;
```

- `static_cast<new type>( expression )`
  - This is exactly equivalent to the C style cast.
  - This identifies a cast **at compile time**.
  - This makes it clear to another programmer that you really intended a cast that reduces precision (ex. double → float) and make it
  - ~99% of all your casts in C++ will be of this type.

- `dynamic_cast<new type>( expression)`
  - Special version where type casting is performed at runtime, only works on reference or pointer type variables.
  - Usually created automatically by the compiler where needed, rarely done by the programmer.

Type Casting cont'd



- `const_cast<new type>( expression )`
  - Variables labeled as *const* can't have their value changed.
  - const_cast lets the programmer remove or add *const* to reference or pointer type variables.
  - If you need to do this, you probably want to re-think your code!
- `reinterpret_cast<new type>( expression )`
  - Takes the bits in the expression and re-uses them **unconverted** as a new type. Also only works on reference or pointer type variables.
  - Sometimes useful when reading or writing binary files or when dealing with hardware devices like serial or USB ports.

"unsafe": the compiler will not protect you here!

The programmer must make sure everything is correct!

Functions

- Open the project “FunctionExample” in the Part 1 NetBeans project file.
  - Compile and run it!
- Open main.cpp
- 4 function calls are listed.
- The 1st and 2nd functions are identical in their behavior.
  - The values of L and W are sent to the function, multiplied, and the product is returned.
- RectangleArea2 uses *const* arguments
  - The compiler **will not** let you modify their values in the function.
  - Try it! Uncomment the line and see what happens when you recompile.
- The 3rd and 4th versions pass the arguments by *reference* with an added *&*

The return type is *float*.

The function arguments L and W are sent as type *float*.

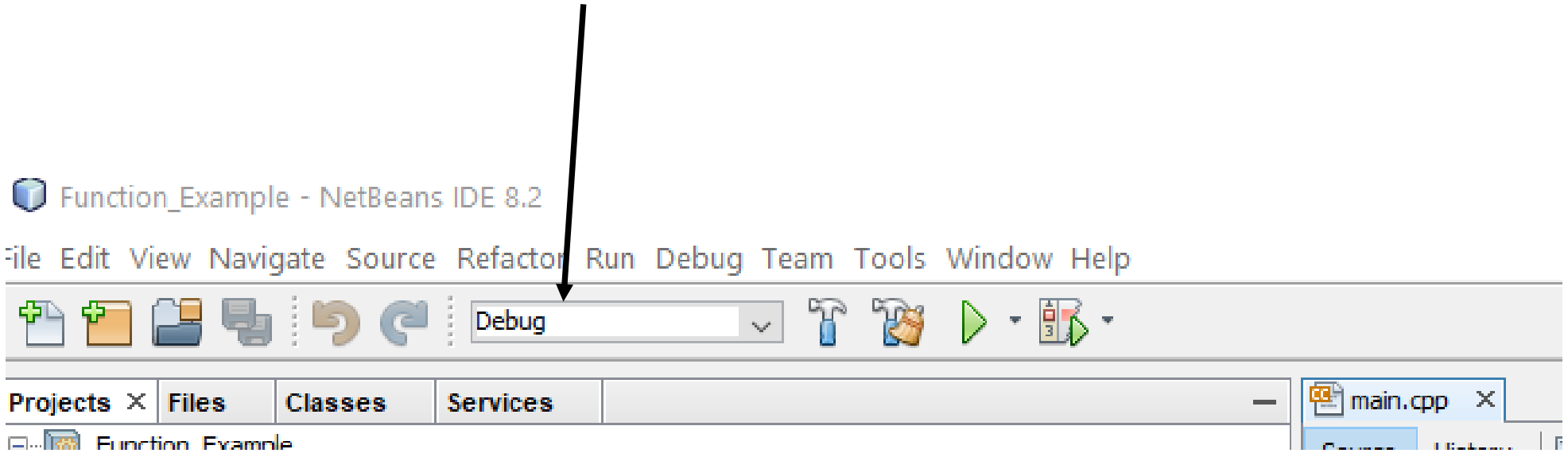
```
float RectangleArea1(float L, float W) {
    return L*W ;
}


float RectangleArea2(const float L, const float W) {
    // L=2.0 ;
    return L*W ;
}


float RectangleArea3(const float& L, const float& W) {
    return L*W ;
}


void RectangleArea4(const float& L, const float& W, float& area) {
    area= L*W ;
}
```

Product is computed

BOSTON UNIVERSITY

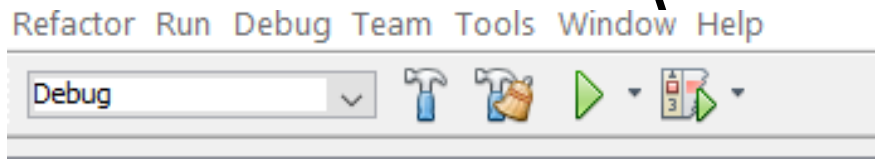
Using the NetBeans Debugger

- To show how this works we will use the NetBeans interactive debugger to step through the program line-by-line to follow the function calls.
- Make sure you are running in *Debug* mode. This turns off compiler optimizations and has the compiler include information in the compiled code for effective debugging.



Add Breakpoints

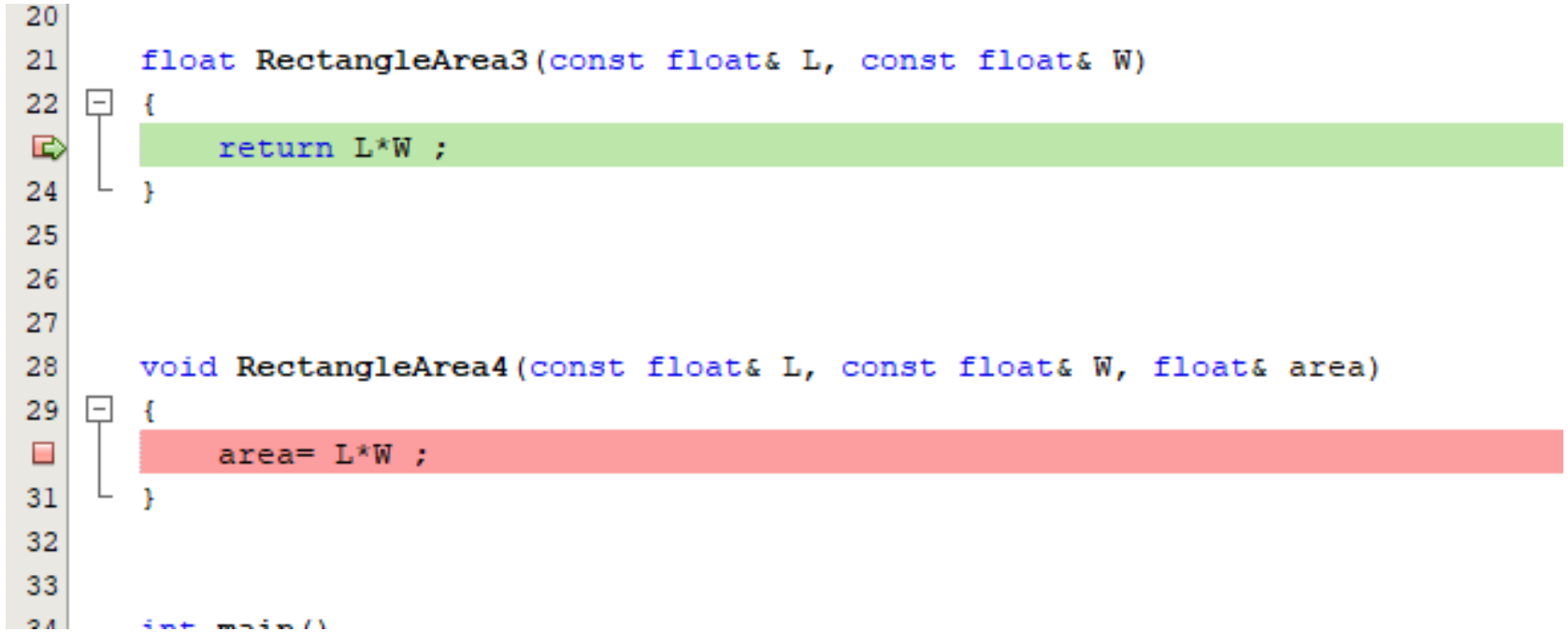
- Breakpoints tell the debugger to halt at a particular line so that the state of the program can be inspected.
- In main.cpp, click to the left of the lines in the functions to set a pair of breakpoints. A red square will appear.
- Click the this arrow to start the code in the debugger.




```
20
21  float RectangleArea3(const float& L, const float& W)
22  {
        return L*W ;
24  }
25
26
27
28  void RectangleArea4(const float& L, const float& W, floa
29  {
        area= L*W ;
31  }
32
```

BOSTON UNIVERSITY

- The debugger will pause the program at the first breakpoint.



```
20
21  float RectangleArea3(const float& L, const float& W)
22  {
        return L*W ;
24  }
25
26
27
28  void RectangleArea4(const float& L, const float& W, float& area)
29  {
        area= L*W ;
31  }
32
33
```

- Controls (hover mouse over for help):

- At the bottom of the window there are several tabs showing the state of the program: