Tutorial T02
Partha Pratim Das
Tutorial Recap
Objectives & Outline
Build Pipeline
Compilers
gcc and g++
Build with GCC
C/C++ Dialects
C Dialects
C++ Dialects
Standard Library
C Std. Lib.
C++ Std. Lib.
std
Header Conventions
Tutorial Summary

Programming in Modern C++

Tutorial T02: How to build a C/C++ program?: Part 2: Build Pipeline

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ac.in

All url's in this module have been accessed in September, 2021 and found to be functional

Tutorial Recap

Tutorial T02
Partha Pratim Das
Tutorial Recap
Objectives & Outline
Build Pipeline
Compilers
gcc and g++
Build with GCC
C/C++ Dialects
C Dialects
C++ Dialects
Standard Library
C Std. Lib.
C++ Std. Lib.
std
Header Conventions
Tutorial Summary

- Understood the differences and relationships between source and header files
- Understood how CPP can be harnessed to manage code during build

Tutorial Objective

Tutorial T02
Partha Pratim Das
Tutorial Recap
Objectives & Outline
Build Pipeline
Compilers
gcc and g++
Build with GCC
C/C++ Dialects
C Dialects
C++ Dialects
Standard Library
C Std. Lib.
C++ Std. Lib.
std
Header Conventions
Tutorial Summary

- What is the build pipelines? Especially with reference to GCC
- How to work with C/C++ dialects during build?
- Understanding C/C++ Standard Libraries

Tutorial Outline

Tutorial T02

Partha Pratim Das

Tutorial Recap

Objectives & Outline

Build Pipeline

Compilers

gcc and g++

Build with GCC

C/C++ Dialects

C Dialects

C++ Dialects

Standard Library

C Std. Lib.

C++ Std. Lib.

std

Header Conventions

Tutorial Summary

1. Tutorial Recap
2. Build Pipeline
   - Compilers, IDE, and Debuggers
   - gcc and g++
   - Build with GCC
3. C/C++ Dialects
   - C Dialects
   - C++ Dialects
4. Standard Library
   - C Standard Library
   - C++ Standard Library
     - `std`
     - Header Conventions
5. Tutorial Summary

Tutorial T02
Partha Pratim Das
Tutorial Recap
Objectives & Outline
Build Pipeline
Compilers
gcc and g++
Build with GCC
C/C++ Dialects
C Dialects
C++ Dialects
Standard Library
C Std. Lib.
C++ Std. Lib.
std
Header Conventions
Tutorial Summary

Build Pipeline

Build Pipeline

Tutorial T02

Partha Pratim Das

Tutorial Recap

Objectives & Outline

Build Pipeline

Compilers

gcc and g++

Build with GCC

C/C++ Dialects

C Dialects

C++ Dialects

Standard Library

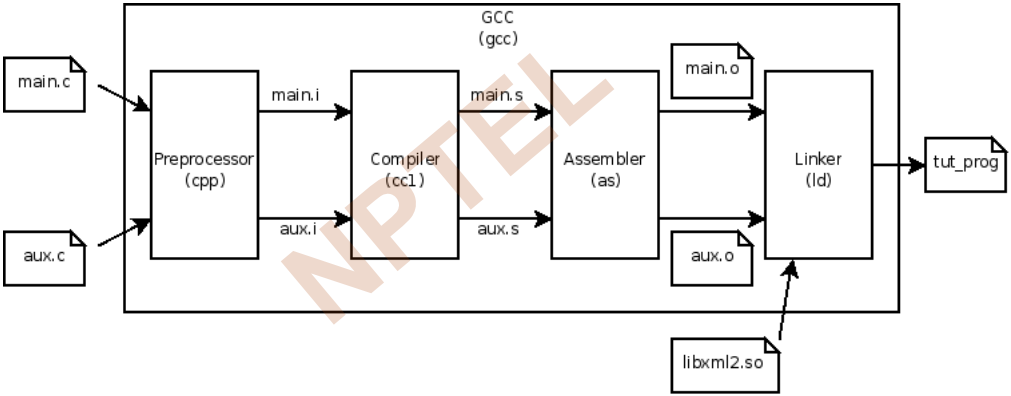
C Std. Lib.

C++ Std. Lib.

std

Header Conventions

Tutorial Summary



Source: GNU Compiler Collection, Wikiwand Accessed 13-Sep-21

Build Pipeline

Tutorial T02
Partha Pratim Das
Tutorial Recap
Objectives & Outline
Build Pipeline
Compilers
gcc and g++
Build with GCC
C/C++ Dialects
C Dialects
C++ Dialects
Standard Library
C Std. Lib.
C++ Std. Lib.
std
Header Conventions
Tutorial Summary

- The **C preprocessor (CPP)** has the ability for the inclusion of header files, macro expansions, conditional compilation, and line control. It works on `.c`, `.cpp`, and `.h` files and produces `.i` files
- The **Compiler** translates the pre-processed C/C++ code into assembly language, which is a machine level code in text that contains instructions that manipulate the memory and processor directly. It works on `.i` files and produces `.s` files
- The **Assembler** translates the assembly program to binary machine language or object code. It works on `.s` files and produces `.o` files
- The **Linker** links our program with the pre-compiled libraries for using their functions and generates the executable binary. It works on `.o` (static library), `.so` (shared library or dynamically linked library), and `.a` (library archive) files and produces `a.out` file

File extensions mentioned here are for GCC running on Linux. These may vary on other OSs and for other compilers. Check the respective documentation for details. The build pipeline, however, would be the same.

Compilers

- **The recommended compiler for the course is GCC, the GNU Compiler Collection - GNU Project. To install it (with gdb, the debugger) on your system, follow**:
  - **Windows**: How to install gdb in windows 10 on YoutTube
  - **Linux**: Usually comes bundled in Linux distribution. Check manual
- You may also use **online versions** for quick tasks
  - GNU Online Compiler
    - From Language Drop-down, choose C (**C99**), C++ (**C++11**), **C++14**, **C++17**, or **C++20**
    - To mark the language for gcc compilation, set `-std=<compiler_tag>`
      - Tags for C are: `ansi`, `c89`, `c90`, `c11`, `c17`, `c18`, etc.
      - Tags for C++ are: `ansi`, `c++98`, `c++03`, `c++11`, `c++14`, `c++17`, `c++20`, etc.
      - Check Options Controlling C Dialect and Language Standards Supported by GCC
  - Code::Blocks is a free, open source cross-platform IDE that supports multiple compilers including GCC, Clang and Visual C++
  - Programiz Online Compiler: **C18** and **C++14**
  - OneCompiler: **C18** and **C++17**
  - JDOODLE Online Compiler And Editor: **C99**, **C11** & **C18** and **C++98**, **C++14** & **C++17**
- *For a compiler, you must know the language version you are compiling for - check to confirm*

What is GCC?

Tutorial T02
Partha Pratim Das
Tutorial Recap
Objectives & Outline
Build Pipeline
Compilers
gcc and g++
Build with GCC
C/C++ Dialects
C Dialects
C++ Dialects
Standard Library
C Std. Lib.
C++ Std. Lib.
std
Header Conventions
Tutorial Summary

- **GCC** stands for **GNU Compiler Collections** which is used to compile mainly C and C++ language
- It can also be used to compile Objective C, Objective C++, Fortran, Ada, Go, and D
- The most important option required while compiling a source code file is the name of the source program, rest every argument is optional like a warning, debugging, linking libraries, object file, etc.
- The different options of GCC command allow the user to stop the compilation process at different stages.
- **g++** command is a GNU C++ compiler invocation command, which is used for preprocessing, compilation, assembly and linking of source code to generate an executable file. The different "options" of g++ command allow us to stop this process at the intermediate stage.

What are the differences between gcc and g++?

g++	gcc
g++ is used to compile C++ program	gcc is used to compile C program
g++ can compile any .c or .cpp files but they will be treated as C++ files only	gcc can compile any .c or .cpp files but they will be treated as C and C++ respectively
Command to compile C++ program by g++ is: `g++ fileName.cpp -o binary`	Command to compile C program by gcc is: `gcc fileName.c -o binary -lstdc++`
Using g++ to link the object files, files automatically links in the std C++ libraries.	gcc does not do this and we need to specify `-lstdc++` in the command line
g++ compiling .c/.cpp files has a few extra macros	gcc compiling .c files has less predefined macros. gcc compiling .cpp files has a few extra macros

```
#define __GXX_WEAK__ 1
#define __cplusplus 1
#define __DEPRECATED 1
#define __GNUG__ 4
#define __EXCEPTIONS 1
#define __private_extern__ extern
```

Build with GCC: Options

Tutorial T02
Partha Pratim Das
Tutorial Recap
Objectives & Outline
Build Pipeline
Compilers
gcc and g++
Build with GCC
C/C++ Dialects
C Dialects
C++ Dialects
Standard Library
C Std. Lib.
C++ Std. Lib.
std
Header Conventions
Tutorial Summary

[1] **Place the source (`.c`) and header (`.h`) files in current directory**

```
11-09-2021  10:46               157 fact.c
11-09-2021  10:47               124 fact.h
11-09-2021  10:47               263 main.c
```

[2] **Compile source files (`.c`) and generate object (`.o`) files using option "`-c`". Note additions of files to directory**

```
$ gcc -c fact.c
$ gcc -c main.c

11-09-2021  11:02               670 fact.o
11-09-2021  11:02             1,004 main.o
```

[3] **Link object (`.o`) files and generate executable (`.exe`) file of preferred name (`fact`) using option "`-o`". Note added file to directory**

```
$ gcc fact.o main.o -o fact

11-09-2021  11:03            42,729 fact.exe
```

[4] **Execute**

```
$ fact
Input n
5
fact(5) = 120
```

[5] **We can combine steps [2] and [3] to generate executable directly by compiling and linking source files in one command**

```
$ gcc fact.c main.c -o fact
```

Build with GCC: Options

Tutorial T02
Partha Pratim Das
Tutorial Recap
Objectives & Outline
Build Pipeline
Compilers
gcc and g++
Build with GCC
C/C++ Dialects
C Dialects
C++ Dialects
Standard Library
C Std. Lib.
C++ Std. Lib.
std
Header Conventions
Tutorial Summary

[6] **We can only compile and generate assembly language (`.s`) file using option "`-S`"**

```
$ gcc -S fact.c main.c

11-09-2021  11:34               519 fact.s
11-09-2021  11:34             1,023 main.s
```

[7] **To stop after prepossessing use option "`-E`". The output is generated in `stdout` (redirected here to `cppout.c`).**

```
$ gcc -E fact.c main.c >cppout.c

11-09-2021  11:32            21,155 cppout.c
```

Note that CPP:

- Produces a single file containing the source from all `.c` files
- Includes all required header files (like `fact.h`, `stdio.h`) and strips off unnecessary codes present there
- Strips off all comments
- Textually replaces all manifest constants and expands all macros

[8] **We can know the version of the compiler**

```
$ gcc --version

gcc (MinGW.org GCC-6.3.0-1) 6.3.0
Copyright (C) 2016 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Build with GCC: Options

Tutorial T02
Partha Pratim Das
Tutorial Recap
Objectives & Outline
Build Pipeline
Compilers
gcc and g++
Build with GCC
C/C++ Dialects
C Dialects
C++ Dialects
Standard Library
C Std. Lib.
C++ Std. Lib.
std
Header Conventions
Tutorial Summary

[9] **When we intend to debug our code with `gdb` we need to use "`-g`" option to tell GCC to emit extra information for use by a debugger**

```
$ gcc -g fact.c main.c -o fact
```

[10] **We should always compile keeping it clean of all warnings. This can be done by "`-Wall`" flag. For example if we comment out `f = fact(n);` and try to build we get warning, w/o "`-Wall`", it is silent**

```
$ gcc -Wall main.c

main.c: In function 'main':
main.c:14:5: warning: 'f' is used uninitialized in this function [-Wuninitialized]
    printf("fact(%d) = %d\n", n, f);
    ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

$ gcc main.c
```

With "`-Werror`", all warnings are treated as errors and no output will be produced

Build with GCC: Options

Tutorial T02
Partha Pratim Das
Tutorial Recap
Objectives & Outline
Build Pipeline
Compilers
gcc and g++
Build with GCC
C/C++ Dialects
C Dialects
C++ Dialects
Standard Library
C Std. Lib.
C++ Std. Lib.
std
Header Conventions
Tutorial Summary

[11] **We can trace the commands being used by the compiler using option "-v", that is, verbose mode**

```
$ gcc -v fact.c main.c -o fact

Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=c:/mingw/bin/../libexec/gcc/mingw32/6.3.0/lto-wrapper.exe
Target: mingw32
[truncated]
Thread model: win32
gcc version 6.3.0 (MinGW.org GCC-6.3.0-1)
[truncated]
```

Build with GCC: Summary of Options and Extensions

- **`gcc` options and file extensions. Note that `.c` is shown as a placeholder for user provided source files. A detailed list of source file extensions are given in the next point**

Option	Behaviour	Input Extension	Output Extension
`-c`	Compile or assemble the source files, but do not link	`.c, .s, .i`	`.o`
`-S`	Stop after the stage of compilation proper; do not assemble	`.c, .i`	`.s`
`-E`	Stop after the preprocessing stage	`.c`	To `stdout`
`-o` *file*	Place the primary output in file *file* (`a.out` w/o `-o`)	`.c, .s, .i`	Default for OS
`-v`	Print the commands executed to run the stages of compilation	`.c, .s, .i`	To `stdout`

- **Source file (user provided) extensions**

Extension	File Type	Extension	File Type
.c	C source code that must be preprocessed	**.cpp**, `.cc`, `.cp`, `.cxx` `.CPP`, `.c++`, `.C`	C++ source code that must be preprocessed
.h	C / C++ header file	`.H`, `.hp`, `.hxx`, `.hpp` `.HPP`, `.h++`, `.tcc`	C++ header file
`.s`	Assembler code	`.S`, `.sx`	Assembler code that must be preprocessed

* Varied extensions for C++ happened during its evolution due various adoption practices

* We are going to follow the extensions marked in red

Source: 3.1 Option Summary and 3.2 Options Controlling the Kind of Output Accessed 13-Sep-21

Tutorial T02
Partha Pratim Das
Tutorial Recap
Objectives & Outline
Build Pipeline
Compilers
gcc and g++
Build with GCC
C/C++ Dialects
C Dialects
C++ Dialects
Standard Library
C Std. Lib.
C++ Std. Lib.
std
Header Conventions
Tutorial Summary

C / C++ Dialects

C Dialects

K&R C	C89/C90	C95	C99	C11	C18
1978	1989/90	1995	1999	2011	2011
Created by Dennis Ritchie in early 1970s augmenting Ken Thompson's B	ANSI Std. in 1989	ISO Published Amendment	New built-in data types: `long long`, `_Bool`, `_Complex`, and `_Imaginary`	type generic macros	ISO Published Amendment
Brian Kernighan wrote the first C tutorial	ISO Std. in 1990	Errors corrected	Headers: `<stdint.h>`, `<tgmath.h>`, `<fenv.h>`, `<complex.h>`	Anonymous structures	Errors corrected
K & R published The C Programming Language in 1978. It worked as a defacto standard for a decade		Better multi-byte & wide character support in the library, with `<wchar.h>`, `<wctype.h>` and multi-byte I/O	Static array indices, designated initializers, compound literals, variable-length arrays, flexible array members, variadic macros, and restrict keyword	Improved Unicode support	
ANSI C was covered in second edition in 1988		digraphs added	Compatibility with C++ like inline functions, single-line comments, mixing declarations and code, universal character names in identifiers	Atomic operations	
		Alternative specs. of operators, like 'and' for '&&'	Removed C89 language features like implicit function declarations and	Multi-threading	
		Std. macro `__STDC_VERSION__` with value `199409L` for C99 support		Std. macro `__STDC_VERSION__` defined as `201112L` for C11 support	Std. macro `__STDC_VERSION__` defined as `201710L` for C18 support
				Bounds-checked functions	
The C Programming Language, 1978	**ANSI X3.159-1989 ISO/IEC 9899:1990**	**ISO/IEC 9899/ AMD1:1995**	**ISO/IEC 9899:1999**	**ISO/IEC 9899:2011**	**ISO/IEC 9899:2018**

Latest Version as of Sep-21: C18: ISO/IEC 9899:2018, 2018

C Dialects: Checking for a dialect

Tutorial T02
Partha Pratim Das
Tutorial Recap
Objectives & Outline
Build Pipeline
Compilers
gcc and g++
Build with GCC
C/C++ Dialects
C Dialects
C++ Dialects
Standard Library
C Std. Lib.
C++ Std. Lib.
std
Header Conventions
Tutorial Summary

- **We check the language version (dialect) of C being used by GCC in compilation using the following code**

```
/* File Check C Version.c */
#include <stdio.h>
int main() {
    if (__STDC_VERSION__ == 201710L) printf("C18\n");       /* C11 with bug fixes */
    else if (__STDC_VERSION__ == 201112L) printf("C11\n");
    else if (__STDC_VERSION__ == 199901L) printf("C99\n");
    else if (__STDC_VERSION__ == 199409L) printf("C89\n");
    else printf("Unrecognized version of C\n");

    return 0;
}
```

- **We can ask GCC to use a specific dialect by using `-std` flag and check with the above code for three cases**

```
$ gcc -std=c99 "Check C Version.c"
C99

$ gcc "Check C Version.c"
C11

$ gcc -std=c11 "Check C Version.c"
C11
```

Default for this `gcc` is `C11`

C++ Standards

C++98	C++11	C++14	C++17	C++20
1998	2011	2014	2017	2020
Templates	Move Semantics	Reader-Writer Locks	Fold Expressions	Coroutines
STL with Containers and Algorithms	Unified Initialization	Generic Lambda Functions	constexpr if	Modules
Strings	auto and decltype		Structured Binding	Concepts
I/O Streams	Lambda Functions		`std::string_view`	Ranges Library
	constexpr		Parallel Algortihms of the STL	
	Multi-threading and Memory Model		File System Library	
	Regular Expressions		`std::any`, `std::optional`, and `std::variant`	
	Smart Pointers			
	Hash Tables			
	`std::array`			
ISO/IEC 14882:1998	ISO/IEC 14882:2011	ISO/IEC 14882:2014	ISO/IEC 14882:2017	ISO/IEC 14882:2020

Fixes on C++98: C++03: ISO/IEC 14882:2003, 2003

Latest Version as of Sep-21: C++20: ISO/IEC 14882:2020, 2020

C++ Dialects: Checking for a dialect

- **We check the language version (dialect) of C++ being used by GCC in compilation using the following code**

```
// File Check C++ Version.cpp
#include <iostream>
int main() {
    if (__cplusplus == 201703L) std::cout << "C++17\n";
    else if (__cplusplus == 201402L) std::cout << "C++14\n";
    else if (__cplusplus == 201103L) std::cout << "C++11\n";
    else if (__cplusplus == 199711L) std::cout << "C++98\n";
    else std::cout << "Unrecognized version of C++\n";
    return 0;
}
```

- **We can ask GCC to use a specific dialect by using `-std` flag and check with the above code for four cases**

```
$ g++ -std=gnu++98 "Check C++ Version.cpp"
C++98

$ g++ -std=c++11 "Check C++ Version.cpp"
C++11

$ g++ -std=c++14 "Check C++ Version.cpp"
C++14

$ g++ "Check C++ Version.cpp"
C++14
```

Default for this `g++` is `C++14`

Tutorial T02
Partha Pratim Das
Tutorial Recap
Objectives & Outline
Build Pipeline
Compilers
gcc and g++
Build with GCC
C/C++ Dialects
C Dialects
C++ Dialects
Standard Library
C Std. Lib.
C++ Std. Lib.
std
Header Conventions
Tutorial Summary

Standard Library

What is Standard Library?

Tutorial T02
Partha Pratim Das
Tutorial Recap
Objectives & Outline
Build Pipeline
Compilers
gcc and g++
Build with GCC
C/C++ Dialects
C Dialects
C++ Dialects
Standard Library
C Std. Lib.
C++ Std. Lib.
std
Header Conventions
Tutorial Summary

- A *standard library in programming* is the library *made available across implementations of a language*
- These libraries are usually described in *language specifications (C/C++)*; however, they may also be determined (in part or whole) *by informal practices of a language's community (Python)*
- A language's standard library is *often treated as part of the language by its users*, although the *designers may have treated it as a separate entity*
- Many language specifications define a *core set that must be made available in all implementations*, in addition to *other portions which may be optionally implemented*
- The line between a *language and its libraries* therefore *differs from language to language*
- Bjarne Stroustrup, designer of C++, writes:

> *What ought to be in the standard C++ library? One ideal is for a programmer to be able to find every interesting, significant, and reasonably general class, function, template, etc., in a library. However, the question here is not, "What ought to be in some library?" but "What ought to be in the standard library?" The answer "Everything!" is a reasonable first approximation to an answer to the former question but not the latter. A standard library is something every implementer must supply so that every programmer can rely on it.*

- This suggests a *relatively small standard library*, containing only the constructs that *"every programmer" might reasonably require when building a large collection of software*
- **This is the philosophy that is used in the C and C++ standard libraries**

Source: Standard library, Wiki Accessed 13-Sep-21

C Standard Library: Common Library Components

Component	Data Types, Manifest Constants, Macros, Functions, ...
stdio.h	Formatted and un-formatted file input and output including functions • printf, scanf, fprintf, fscanf, sprintf, sscanf, feof, etc.
stdlib.h	Memory allocation, process control, conversions, pseudo-random numbers, searching, sorting • malloc, free, exit, abort, atoi, strtold, rand, bsearch, qsort, etc.
string.h	Manipulation of C strings and arrays • strcat, strcpy, strcmp, strlen, strtok, memcpy, memmove, etc.
math.h	Common mathematical operations and transformations • cos, sin, tan, acos, asin, atan, exp, log, pow, sqrt, etc.
errno.h	Macros for reporting and retrieving error conditions through error codes stored in a static memory location called errno • EDOM (parameter outside a function's domain – sqrt(-1)), • ERANGE (result outside a function's range), or • EILSEQ (an illegal byte sequence), etc.

A header file typically contains manifest constants, macros, necessary struct / union types, typedef's, function prototype, etc.

C Standard Library: math.h

Tutorial T02
Partha Pratim Das
Tutorial Recap
Objectives & Outline
Build Pipeline
Compilers
gcc and g++
Build with GCC
C/C++ Dialects
C Dialects
C++ Dialects
Standard Library
C Std. Lib.
C++ Std. Lib.
std
Header Conventions
Tutorial Summary

```
/* math.h
 * This file has no copyright assigned and is placed in the Public Domain.
 * This file is a part of the mingw-runtime package.
 * Mathematical functions.
 */
#ifndef _MATH_H_
#define _MATH_H_
#ifndef __STRICT_ANSI__  // conditional exclusions for ANSI
// ...
#define M_PI 3.14159265358979323846  // manifest constant for pi
// ...
struct _complex {  // struct of _complex type
    double        x;        /* Real part */
    double        y;        /* Imaginary part */
};
_CRTIMP double __cdecl _cabs (struct _complex);  // cabs(.) function header
// ...
#endif /* __STRICT_ANSI__ */
// ...
_CRTIMP double __cdecl sqrt (double);  // sqrt(.) function header
// ...
#define isfinite(x) ((fpclassify(x) & FP_NAN) == 0)  // macro isfinite(.) to check if a number is finite
// ...
#endif /* _MATH_H_ */
```


Source: C math.h library functions Accessed 13-Sep-21

C++ Standard Library: Common Library Components

Component	Data Types, Manifest Constants, Macros, Functions, Classes, ...
iostream	Stream input and output for standard I/O • cout, cin, endl, ..., etc.
string	Manipulation of string objects • Relational operators, IO operators, Iterators, etc.
memory	High-level memory management • Pointers: unique_ptr, shared_ptr, weak_ptr, auto_ptr, & allocator etc.
exception	Generic Error Handling • exception, bad_exception, unexpected_handler, terminate_handler, etc.
stdexcept	Standard Error Handling • logic_error, invalid_argument, domain_error, length_error, out_of_range, runtime_error, range_error, overflow_error, underflow_error, etc.
Adopted from C Standard Library	
cmath	Common mathematical operations and transformations • cos, sin, tan, acos, asin, atan, exp, log, pow, sqrt, etc.
cstdlib	Memory alloc., process control, conversions, pseudo-rand nos., searching, sorting • malloc, free, exit, abort, atoi, strtold, rand, bsearch, qsort, etc.

namespace std for C++ Standard Library

C Standard Library	C++ Standard Library
• All names are global • stdout, stdin, printf, scanf	• All names are within std namespace • std::cout, std::cin • Use using namespace std; to get rid of writing std:: for every standard library name

W/o using

```
#include <iostream>


int main() {

    std::cout << "Hello World in C++"
              << std::endl;

    return 0;
}
```

W/ using

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

int main() {

    cout << "Hello World in C++"
         << endl;

    return 0;
}
```

Standard Library: C/C++ Header Conventions

Tutorial T02
Partha Pratim Das
Tutorial Recap
Objectives & Outline
Build Pipeline
Compilers
gcc and g++
Build with GCC
C/C++ Dialects
C Dialects
C++ Dialects
Standard Library
C Std. Lib.
C++ Std. Lib.
std
Header Conventions
Tutorial Summary

	C Header	C++ Header
C Program	Use `.h`. Example: `#include <stdio.h>` *Names in global namespace*	Not applicable
C++ Program	Prefix `c`, no `.h`. Example: `#include <cstdio>` *Names in `std` namespace*	No `.h`. Example: `#include <iostream>`

- A C std. library header is used in C++ with prefix '`c`' and without the `.h`. These are in `std` namespace:

```
#include <cmath>  // In C it is <math.h>
...
std::sqrt(5.0);   // Use with std::
```

It is possible that a C++ program include a C header as in C. Like:

```
#include <math.h> // Not in std namespace
...
sqrt(5.0);        // Use without std::
```

This, however, is not preferred

- **Using `.h` with C++ header files, like `iostream.h`, is disastrous. These are deprecated. It is dangerous, yet true, that some compilers do not error out on such use. Exercise caution.**

Tutorial Summary

Tutorial T02
Partha Pratim Das
Tutorial Recap
Objectives & Outline
Build Pipeline
Compilers
gcc and g++
Build with GCC
C/C++ Dialects
C Dialects
C++ Dialects
Standard Library
C Std. Lib.
C++ Std. Lib.
std
Header Conventions
Tutorial Summary

- Understood the overall build process for a C/C++ project with specific reference to the build pipeline of GCC
- Understood the management of C/C++ dialects and C/C++ Standard Libraries