

Introduction to C++, Compile, Linker, and Runtime Errors

ELEC 376 (week 1)

About ELEC 376

Instructor: Mariam Guizani, PhD

TAs:

- FATHOLLAZAEH, Pouya
- OUF, Mohamed (Ashraf)
- TARAZ, Tooraj

- Contact information:

- Available on OnQ including your TAs emails

The university communicates with students via **Queen's email**. Please check your email regularly to ensure you do not miss important information related to your course.

Communication with instructor or TAs:

Start your email subject with [ELEC 376 F25]. This helps us get back to you in a timely manner.

Mariam Guizani



Assistant Professor

ECE & Connected Minds Member

Phone: 613-533-6000 ext. 77297

Walter Light Hall, Room: 403

Research Interests: Empirical Software Engineering, Mining Software Repositories, Open-Source Software, EDI, Human-Computer Interaction, LLM4SE

Class Schedule

Lectures: KINGSTON RM101

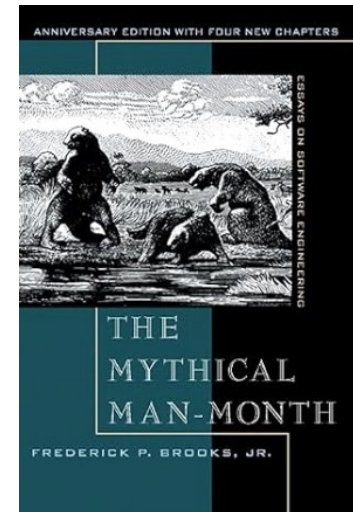
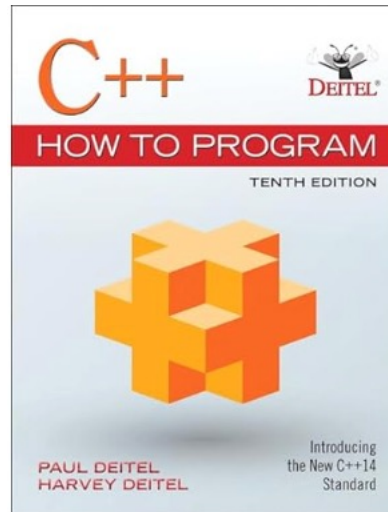
- Mondays 3h30PM - 4:30PM
- Wednesdays 2:30PM - 3:30PM
- Thursdays 4:30PM - 5:30PM

Tutorial: ELLIS RM324 (Mandatory attendance)

- Monday 10h30 - 11h30 AM

Suggested books (NOT required) but good reads

- P. Deitel and H. Deitel, *C++ How to Program*, 10th ed., Pearson, 2017.
- F. Brooks Jr. *Mythical Man-Month, The: Essays on Software Engineering*, Anniversary Edition



Topics covered

- Modern C++
- Software Development
Methods
- Project Management

On Sharing Notes

Queens's Student Accessibility Services (QSAS) Request for Note Sharing

×

This is a message from Queen's Student Accessibility Services (QSAS)

Queen's Student Accessibility Services (QSAS) is seeking individuals who are willing share their notes for this class.

By sharing your class notes, you will be supporting the creation of an accessible learning experience at Queen's University. Also, every time you share your notes, you will **be entered into a monthly draw for a \$25 Starbucks gift card (the more notes you share, the more chances to win).**

Sharing your notes is easy - when you save your notes, it's one quick step to upload those notes to the [noteQ portal](#). Simply click on the noteQ link and follow the instructions in the portal to share your class notes.

Your shared notes will only be available to students in this class who have been provided access through QSAS. Your identity will not be shared with the student requesting notes.

Please consider sharing your notes and contributing to the creation of an accessible learning experience for Queen's University students.

Topics covered

- **Modern C++**
- Software Development Methods
- Project Management
- Program structure
- Compiler errors/ warning and linker errors
- Variables and constants
- Statements and Operators
- Arrays and vectors
- Program flow controls
- Characters and Strings
- Streams and File I/O
- Functions
- Pointers and references
- Classes and Object-Oriented Programming
- Constructors and destructors
- Inheritance
- Polymorphism
- Operator Overloading
- Handling exceptions
- The Standard Template Library (STL)

Topics covered

- Modern C++
- **Software Development Methods**
- **Project Management**
- Software development methodologies
- Agile Programming Techniques
- UML Diagrams
- Functional and non-functional requirements
- User Interface (UI) Prototypes low-fidelity and high-fidelity prototypes
- Software project management tools and best practices (e.g., "The Mythical Man Month")

Deliverables

- Three Quizzes (10% each)
- Group software development project (40%) - Individual and group grading
 - Weekly meetings with your assigned TA (**mandatory** attendance to the tutorial)
 - RAD, SDD documents
 - Three sprint project updates
 - Presentation and demo
- Midterm (30%)
- A late deliverable submission will result in a 10% deduction of the grade per late day.

On teaming

Analyzing the Communication Patterns of Different Teammate Types in a Software Engineering Course Project

Yanye Luther Colorado State University Fort Collins, Colorado, United States yanye.luther@colostate.edu	Lindsey Nielsen Colorado State University Fort Collins, Colorado, United States lindsey.rebecca.nielsen@colostate.edu	Logan Cadman Colorado State University Fort Collins, Colorado, United States allen.cadman@colostate.edu
Marcia Moraes Colorado State University Fort Collins, Colorado, United States marcia.moraes@colostate.edu	Sudipto Ghosh Colorado State University Fort Collins, Colorado, United States sudipto.ghosh@colostate.edu	Bianca Trinkenreich Colorado State University Fort Collins, Colorado, United States bianca.trinkenreich@colostate.edu

Abstract

Effective communication is vital for the success of professional software engineering (SE) teams. As SE courses teach essential industry skills like teamwork and collaboration, ensuring effective communication becomes important in student projects. However, poor engagement from team members can lead to conflicts, uneven workloads, and diminished learning experiences. Teammate types such as Couch Potatoes, who contribute minimally, and Hitchhikers, who rely on others while taking credit, exacerbate these issues. In contrast, Lone Wolves work independently, potentially isolating themselves, while Good Teammates actively collaborate and contribute fairly, driving team success. In this study, we aimed to investigate the communication patterns of teammate types such as Couch Potato, Hitchhiker, Lone Wolf, or Good Teammate during a SE testing course. We applied Ordered Network Analysis (ONA) to the conversational data of the teams to examine the distinct communication patterns of students whose contributions were either perceived positively (e.g., Good Teammates) or negatively (e.g., Couch Potato, Hitchhiker, Lone Wolf) by their peers. The findings reveal distinct communication behaviors across teammate

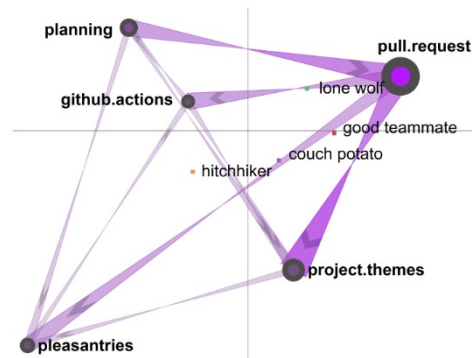
ACM Reference Format:

Yanye Luther, Lindsey Nielsen, Logan Cadman, Marcia Moraes, Sudipto Ghosh, and Bianca Trinkenreich. 2025. Analyzing the Communication Patterns of Different Teammate Types in a Software Engineering Course Project. In *33rd ACM International Conference on the Foundations of Software Engineering (FSE Companion '25)*, June 23–28, 2025, Trondheim, Norway. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3696638.3727232>

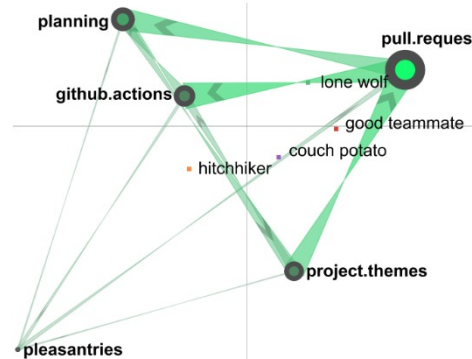
1 Introduction

Software engineering is fundamentally a socio-technical discipline [32] in which effective team communication plays a critical role in the success of software projects, contributing not only to academic achievement but also to professional development in the software engineering field [10, 24]. In educational settings and the software industry, collaboration and communication are fundamental to achieving successful outcomes in group-based course projects. The skills developed through team work during academic training directly translate into the collaborative nature of software development in real-world industry environments [24].

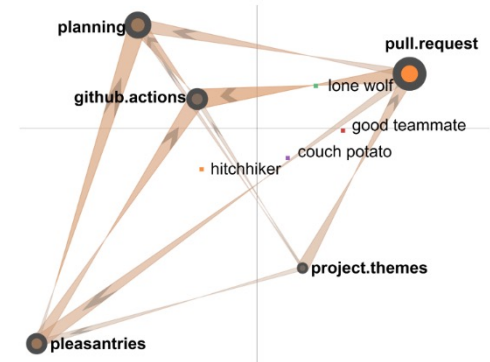
Working in teams provides students with the opportunity to



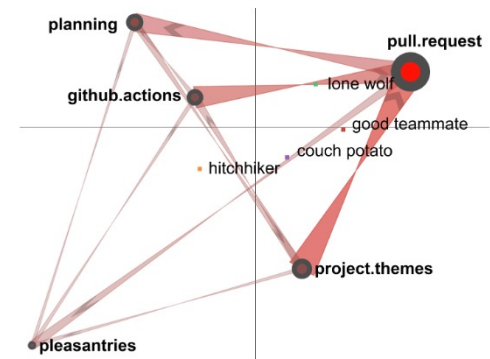
(a) Couch Potato



(c) Lone Wolf



(b) Hitchhiker



(d) Good Teammate

[1] Luther, Yanye, et al. "Analyzing the Communication Patterns of Different Teammate Types in a Software Engineering Course Project." *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*. 2025.



TODO: Programing and teaming experience background survey

- **TODO:** Programming and teamwork survey due on Thursday, Sep 4
(11:59pm)
 - On onQ: week 1 (activities and assessments)
 - You must fill out the survey to be added to a team
- **Note:** start thinking about ideas to share with your team on Monday

Why C++?

Popular

- A lot of code is written in C++ (popularity indexes)
- GitHub, StackOverflow
- Ranked 2nd in the TIOBE index as of 2025.
 - TIOBE (<https://www.tiobe.com/tiobe-index/>)

Aug 2025	Aug 2024	Change	Programming Language		Ratings	Change
1	1			Python	26.14%	+8.10%
2	2			C++	9.18%	-0.86%

Why C++?

Relevant:

- Windows, MacOS, many of the Adobe products, game engines
- Type of technology: Machine Learning, VR, Networking and Telecom...
- Many companies: Amazon, Microsoft, Apple ,...

Why C++?

Powerful

- Fast, flexible, scalable and portable

Good career opportunities

- Good for your CV 😊

A bit of history

1979

- Bjarne Stroustrup (work for his Ph.D. thesis)
- Called it “C with classes”

1983

- Renaming to C++

1989

- Commercial release

A bit of history

1998

- C++ 98 standard

2003

- C++ 03 standard

2011

- C++ 11 standard

2014

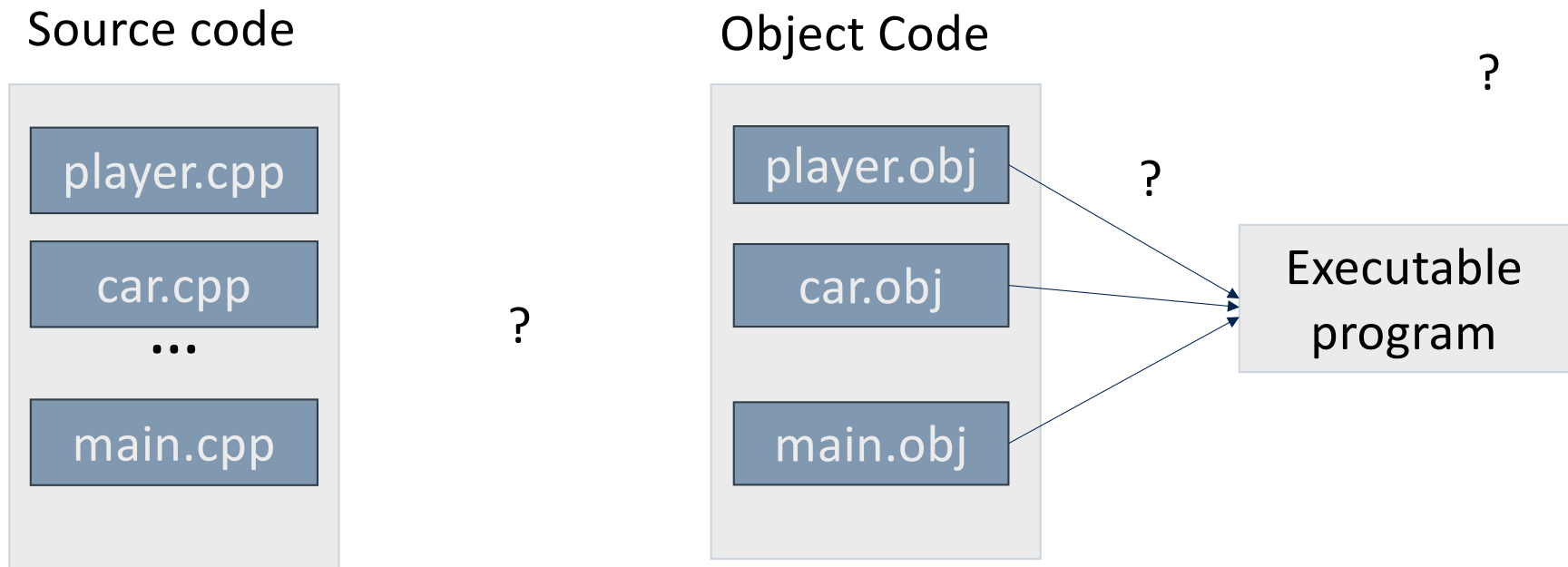
- C++ 14 standard

2017

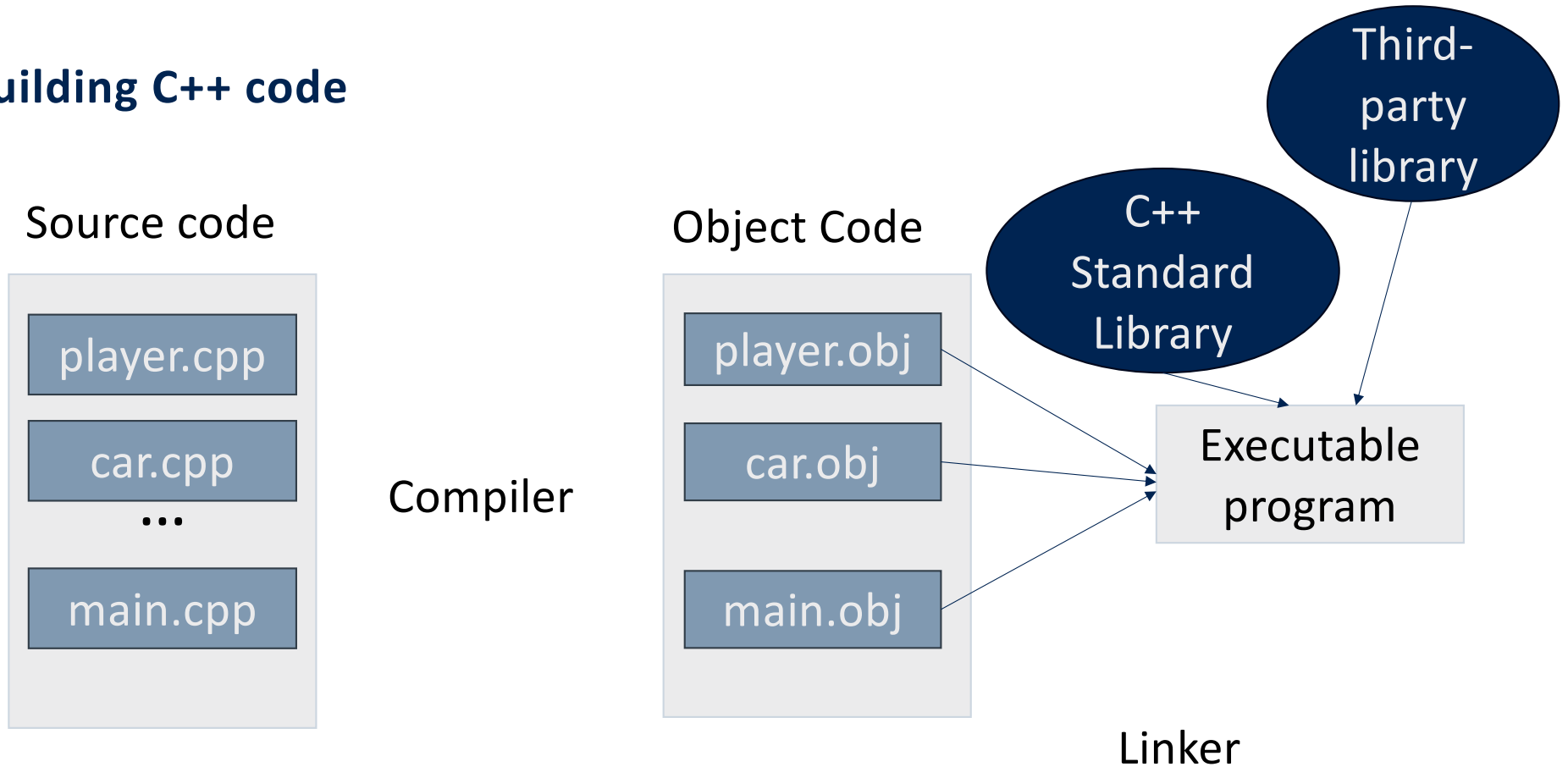
- C++ 17 standard
- ...

More on the history: <https://cplusplus.com/info/history/>

Building C++ code



Building C++ code



Editors/ IDEs

- Editor
- Compiler
- Linker
- Debugger (most IDE do)

Examples:

- CodeLite (debugger problem with M2 Macs)
- **VS Code**
- Clion
- Code::Blocks

VS Code

- Download the tutorial corresponding to your OS
 - VsCode_win_tutorial
 - VsCode_mac_tutorial

C++ errors

What type of error is this?

```
int main()  
    cout << "Hello world << endl;
```

missing closing quote C/C++(8)
[View Problem](#) [Quick Fix...](#)

What type of error is this?

```
    return 0  
    expected a ';' C/C++(65)  
View Problem Quick
```

C++ errors

Compiler Error

What kind of compiler errors?

```
int main()  
    cout << "Hello world << endl;
```

missing closing quote C/C++(8)

[View Problem](#) [Quick Fix...](#)

```
    return 0  
}
```

expected a ';' C/C++(65)

[View Problem](#) [Quick](#)

C++ errors

What type of error is this?

```
Class
no operator "-" matches these operands C/C++(349)
main.cpp(45, 22): operand types are: BankAccount - int

}
main.cpp(45, 22): candidate function template "std::__1::operator-
(const std::__1::fpos<_StateT> &__x, const std::__1::fpos<_StateT>
&__y)" failed deduction

int
main.cpp(45, 22): candidate function template "std::__1::operator-
(const std::__1::reverse_iterator<_Iter1> &__x, const
std::__1::reverse_iterator<_Iter2> &__y)-
>decltype((__y.std::__1::reverse_iterator<_Iter2>::base() -
__x.std::__1::reverse_iterator<_Iter1>::base()))" failed deduction

main.cpp(45, 22): candidate function template "std::__1::operator-
(const std::__1::move_iterator<_Iter1> &__x, const
cout << account1 - x << endl;
```


C++ errors

Compiler error

What kind of compiler error?

```
Class
no operator "-" matches these operands C/C++(349)
main.cpp(45, 22): operand types are: BankAccount - int

main.cpp(45, 22): candidate function template "std::__1::operator-
(const std::__1::fpos<_StateT> &__x, const std::__1::fpos<_StateT>
&__y)" failed deduction
}
int
main.cpp(45, 22): candidate function template "std::__1::operator-
(const std::__1::reverse_iterator<_Iter1> &__x, const
std::__1::reverse_iterator<_Iter2> &__y)-
>decltype((__y.std::__1::reverse_iterator<_Iter2>::base() -
__x.std::__1::reverse_iterator<_Iter1>::base()))" failed deduction
main.cpp(45, 22): candidate function template "std::__1::operator-
(const std::__1::move_iterator<_Iter1> &__x, const
cout << account1 - x << endl;
```

Another example

Is this an error?

```
int score;  
cout << score << endl;
```

Another example

Is this an error?

```
int score;  
cout << score << endl;
```

```
:39:14: note: initialize the variable 'score' to silence this warning  
int score;  
    ^  
    = 0  
1 warning generated.
```

Will the code run?

C++ errors

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

extern int score;

int main () {
    cout << score << endl;
    return 0;
}
```

Any errors?

Linker errors

```
Undefined symbols for architecture arm64:  
  "_score", referenced from:  
      _main in main-bac406.o  
ld: symbol(s) not found for architecture arm64  
clang: error: linker command failed with exit code 1 (use -v to see invocation)  
  
Build finished with error(s).
```

Not a compiler error this is a linker error.

When using third party libraries.

C++ errors

Examples:

- Divide by zero
- File not found
- Out of memory

What type of errors are these?

C++ errors

What about this code?

Will the program run?

```
int main() {  
    int num1 = 10;  
    int num2 = 20;  
  
    int average = num1 + num2;  
  
    std::cout << "The average is: " << average << std::endl;  
  
    return 0;  
}
```

C++ keywords

- C++ has a lot of keywords compared to other languages
- C
- Java
- Example: int, return
- Notice **#include** is not a keyword

C++ keywords

This is a list of reserved keywords in C++. Since they are used by the language, these keywords are not available for re-definition or overloading. As an exception, they are not considered reserved in [attributes](#) (excluding attribute argument lists). (since C++11)

A - C	D - P	R - Z
<code>alignas</code> (C++11)	<code>decltype</code> (C++11)	<code>constexpr</code> (reflection TS)
<code>alignof</code> (C++11)	<code>default</code> (1)	<code>register</code> (2)
<code>and</code>	<code>delete</code> (1)	<code>reinterpret_cast</code>
<code>and_eq</code>	<code>do</code>	<code>requires</code> (C++20)
<code>asm</code>	<code>double</code>	<code>return</code>
<code>atomic_cancel</code> (TM TS)	<code>dynamic_cast</code>	<code>short</code>
<code>atomic_commit</code> (TM TS)	<code>else</code>	<code>signed</code>
<code>atomic_noexcept</code> (TM TS)	<code>enum</code> (1)	<code>sizeof</code> (1)
<code>auto</code> (1) (2) (3) (4)	<code>explicit</code>	<code>static</code>
<code>bitand</code>	<code>export</code> (1) (3)	<code>static_assert</code> (C++11)
<code>bitor</code>	<code>extern</code> (1)	<code>static_cast</code>
<code>bool</code>	<code>false</code>	<code>struct</code> (1)
<code>break</code>	<code>float</code>	<code>switch</code>
<code>case</code>	<code>for</code> (1)	<code>synchronized</code> (TM TS)
<code>catch</code>	<code>friend</code>	<code>template</code>
<code>char</code>	<code>goto</code>	<code>this</code> (4)
<code>char8_t</code> (C++20)	<code>if</code> (2) (4)	<code>thread_local</code> (C++11)
<code>char16_t</code> (C++11)	<code>inline</code> (1)	<code>throw</code>
<code>char32_t</code> (C++11)	<code>int</code>	<code>true</code>
<code>class</code> (1)	<code>long</code>	<code>try</code>
<code>compl</code>	<code>mutable</code> (1)	<code>typedef</code>
<code>concept</code> (C++20)	<code>namespace</code>	<code>typeid</code>
<code>const</code>	<code>new</code>	<code>typename</code>
<code>constexpr</code> (C++20)	<code>noexcept</code> (C++11)	<code>union</code>
<code>constexpr</code> (C++11)	<code>not</code>	<code>unsigned</code>
<code>constinit</code> (C++20)	<code>not_eq</code>	<code>using</code> (1)
<code>const_cast</code>	<code>nullptr</code> (C++11)	<code>virtual</code>
<code>continue</code>	<code>operator</code> (4)	<code>void</code>
<code>co_await</code> (C++20)	<code>or</code>	<code>volatile</code>
<code>co_return</code> (C++20)	<code>or_eq</code>	<code>wchar_t</code>
<code>co_yield</code> (C++20)	<code>private</code> (3)	<code>while</code>
	<code>protected</code>	<code>xor</code>
	<code>public</code>	<code>xor_eq</code>

- (1) — meaning changed or new meaning added in C++11.
- (2) — meaning changed or new meaning added in C++17.
- (3) — meaning changed or new meaning added in C++20.
- (4) — new meaning added in C++23.

Preprocessor directive

```
1 #include <header>  
2 #include "file"
```

- C++ preprocessor is a program that processes the code before sending it to the compiler
- Takes out comments
- Looks for **preprocessor directives** and executes them
 - Example: source file inclusion

Comments

- Readable explanation in the source code
- The compiler does not see the comments they are taken out by the preprocessor
- Two styles of comments:

- Single line comment

```
// This is a single line comment
```

- Multi-line comments

```
/* This is
```

```
a multi-line comment */
```

Comments do's and don't

- Self-documenting code
- Clear variable naming
- Avoiding obvious comments
- Commenting complex code
- Consistent comment style
- Quality comments and code
- Updating comments with code changes
- Avoid using comments as version control use git instead

Best practices

- Have a consistent naming style
 - Camel case or underscore (aka snake case)
 - `studentCount` or `student_count`
- Explicit names not too long and not too short (avoid abbreviations)
- Initialize variable before using them (to avoid unexpected results)
- Variables should be declared near where they are used in the code.

Example: Google have their own published C++ style guide

Variable Names

The names of variables (including function parameters) and data members are `snake_case` (all lowercase, with underscores between words). Data members of classes (but not structs) additionally have trailing underscores. For instance: `a_local_variable`, `a_struct_data_member`, `a_class_data_member_`.

Common Variable names

For example:

```
std::string table_name; // OK - snake_case.
```

```
std::string tableName; // Bad - mixed case.
```

Class Data Members

Data members of classes, both static and non-static, are named like ordinary nonmember variables, but with a trailing underscore.

```
class TableInfo {  
    ...  
private:  
    std::string table_name_; // OK - underscore at end.  
    static Pool<TableInfo>* pool_; // OK.  
};
```

<https://google.github.io/styleguide/cppguide.html>

Google C++ style guide

<https://google.github.io/styleguide/cppguide.html>

C++ Version	
Header Files	Self-contained Headers The #define Guard Include What You Use Forward Declarations Inline Functions Names and Order of Includes
Scoping	Namespaces Internal Linkage Nonmember, Static Member, and Global Functions Local Variables Static and Global Variables thread_local Variables
Classes	Doing Work in Constructors Implicit Conversions Copyable and Movable Types Structs vs. Classes Structs vs. Pairs and Tuples Inheritance Operator Overloading Access Control Declaration Order
Functions	Inputs and Outputs Write Short Functions Function Overloading Default Arguments Trailing Return Type Syntax
Google-Specific Magic	Ownership and Smart Pointers cpplint
Other C++ Features	Rvalue References Friends Exceptions noexcept Run-Time Type Information (RTTI) Casting Streams Preincrement and Predecrement Use of const Use of constexpr, constexpr, and constexpr Integer Types 64-bit Portability Preprocessor Macros 0 and nullptr/NULL sizeof Type Deduction (including auto) Class Template Argument Deduction Designated Initializers Lambda Expressions Template Metaprogramming Concepts and Constraints Boost Other C++ Features Nonstandard Extensions Aliases Switch Statements
Inclusive Language	
Naming	General Naming Rules File Names Type Names Variable Names Constant Names Function Names Namespace Names Enumerator Names Macro Names Exceptions to Naming Rules
Comments	Comment Style File Comments Struct and Class Comments Function Comments Variable Comments Implementation Comments Punctuation, Spelling, and Grammar TODO Comments
Formatting	Line Length Non-ASCII Characters Spaces vs. Tabs Function Declarations and Definitions Lambda Expressions Floating-point Literals Function Calls Braced Initializer List Format Looping and branching statements Pointer and Reference Expressions Boolean Expressions Return Values Variable and Array Initialization Preprocessor Directives Class Format Constructor Initializer Lists Namespace Formatting Horizontal Whitespace Vertical Whitespace
Exceptions to the Rules	Existing Non-conformant Code Windows Code

The main() function

- Each program has exactly one main function
- It is where the program starts the execution
- Return 0 indicates successful execution

```
int main() {  
    //code  
    return 0;  
}
```

```
int main(int argc, char* argv[]) {  
    // code  
    return 0;  
}
```

main expects two arguments:

Argument count and the actual arguments

Basic I/O cin and cout

- cout
 - Standard output stream
 - Console
- cin
 - Standard input stream
 - Keyboard
- <<
 - Insertion operator
 - output streams
- >>
 - extraction operator
 - Input streams

Namespaces

- Their role is to avoid **naming conflicts**

`std::cout`

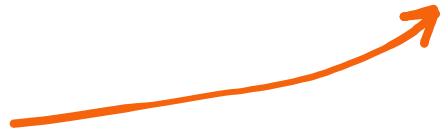
- Third party libraries will have their namespaces
- `::` is the scope resolution operator
- we can use

`using namespace std;`

`using std::cout;`

Namespaces

- Same name is possible as long as it is in a different namespace.
- To place your class definitions in a namespace, simply name it:
- Now the class animal can be defined in a different namespace (e.g., std) and there will be no conflict.



```
namespace alpha {  
    class Animal {  
        ...  
    };  
}
```

Namespaces

- Particularly useful when you are building a library of classes.
- Your namespace's name must be unique.
- You can use an alias to make this easier:
 - `namespace alpha = Alpha_Software_Company;`
- Now the user of your library, just needs to type:
 - `alpha::someThing();`

Namespaces best practice

- The only namespace that you should identify with the **using** keyword is `std`.
- All other namespaces should be named in the statement, like **`alpha::`**
- Then you won't conflict with anything in the `std` namespace.

Nested namespaces

- You can declare a namespace inside another namespace:

```
namespace A {  
    namespace B {  
        namespace C {  
            // ...  
        }  
    }  
}
```

- Or you could just do:

```
namespace A::B::C {  
    //...  
}
```

Variable declaration and initialization

- Examples:

```
double num; // not initialized
```

```
int studentCount = 74; // C-style initialization
```

Variable declaration and initialization

- Examples:

```
int studentCount(74); // constructor style C++
```

```
bool flag(true); // constructor style
```

```
int studentCount{74}; // list initialization syntax
```

(C++ 11)

Variable declaration and initialization

- This is what we are used to

```
int studentCount = 74;
```

- For a C++ “style” declaration and initialization, use:

```
int studentCount{74};
```




ELEC 376 – SOFTWARE DEVELOPMENT METHODOLOGY

Course Syllabus – Fall 2025

This is your course syllabus. Please download the file and keep it for future reference.

LAND ACKNOWLEDGEMENT

Queen's University is situated on traditional Anishinaabe and Haudenosaunee Territory.
See: <http://www.queensu.ca/encyclopedia/t/traditional-territories>