



Templates and Lambda Expressions

CSE 232 – Dr. Josh Nahum

Reading:

Section 7.1, 7.2 (stopping just before Section 7.2.1), and Section 7.3 (stopping just before Section 7.3.3.2)

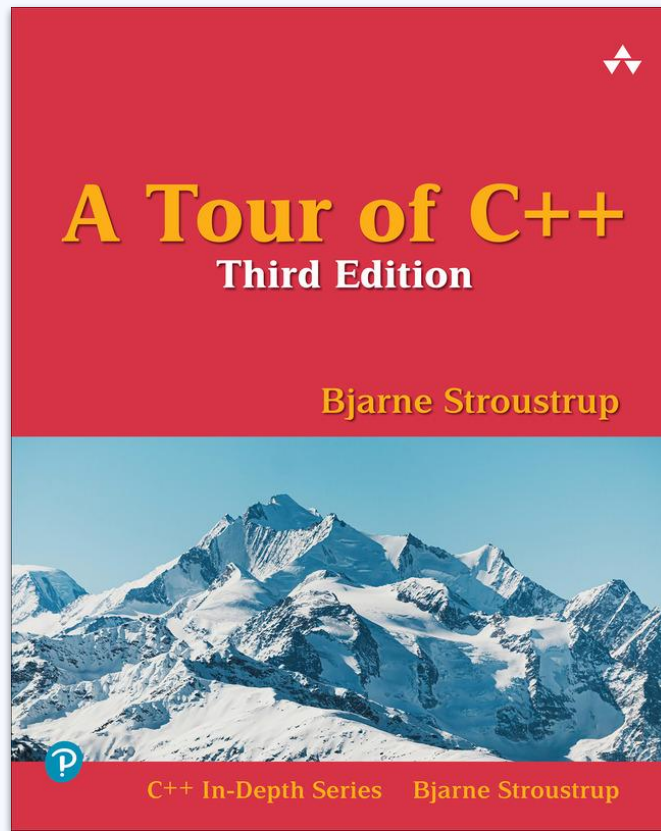




Table of contents

00

Motivation

01

Header File

02

Lambda Expressions

03

Untaught C++





00

Motivation



Why Use Templates?



Generic Code

Write code that can work on multiple types, instead of overloading the same function with almost identical copies.



Future Proofing

Parameterized types and function templates can work with types that you didn't anticipate when you created them.



01

Header File



Template Notes

01 — Recipe — Templates can't be compiled, they are instructions (recipes) for how to build classes/functions.

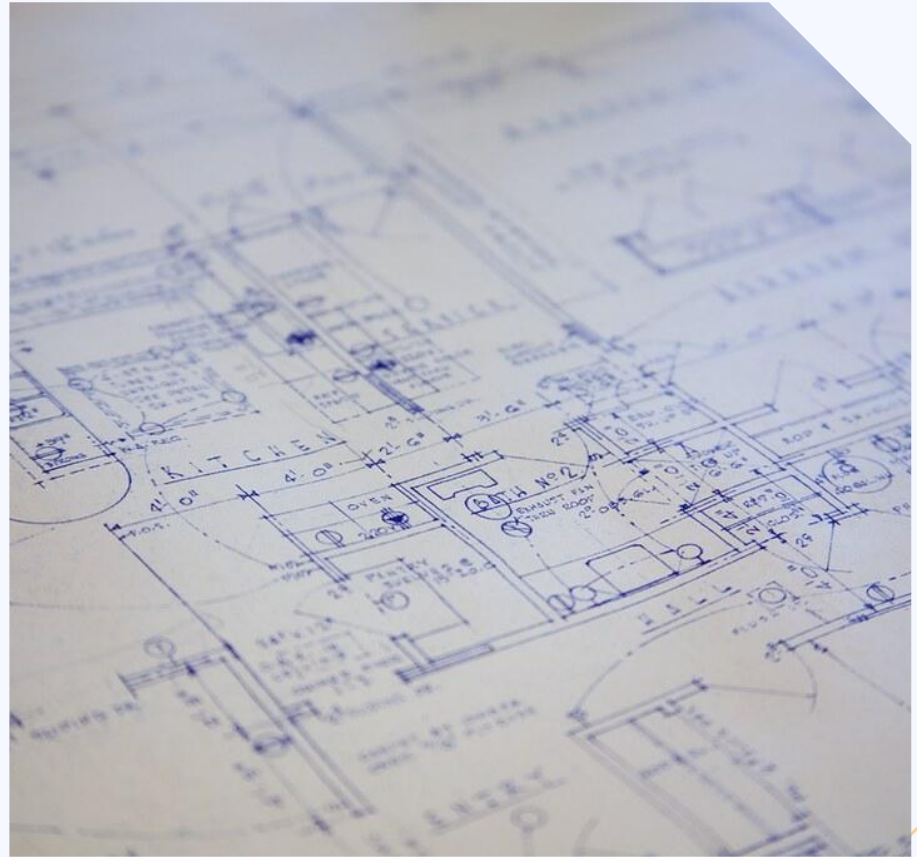
02 — Usage — In order to use a template (e.g. `Vector<char> vc(200);` or `sum(vi, 0);`), the template must be accessible to the compilation unit.

03 — .cpp — Thus the template use be in the .cpp that uses it. Or ...

04 — .h — The template must be in a header file that is included by the cpp file.

TL;DR

Parameterized types and function templates must be in header files to be used by other files.





02

Lambda Expressions



Add First and Last

```
int SumFirstLast(vector<int> const & v) {  
    return v.at(0) + v.at(v.size() - 1);  
}
```

```
int main() {  
    vector<int> v = {1, 2, 3, 4};  
    cout << SumFirstLast(v) << endl;  
}
```

```
int main() {  
    vector<int> v = {1, 2, 3, 4};  
    cout << [](vector<int> const& v) {  
        return v.at(0) + v.at(v.size() - 1);  
    }(v) << endl;  
}
```



Lambda Expressions Versus Named Functions



Convenient

For small, single-use functions, a lambda is easier to write than a named function.




Capture List

Lambdas can access local variables by reference or by copy, named functions can't do that.



Self Documenting

A simple lambda expression is easier to read than a named function.



“The most important use case for lambda expressions is with generic algorithms, which we haven't taught (yet).”

—Dr. Nahum





03

Untaught C++



Skipped Sections

Template meta-programming is a powerful tool that we are only lightly touching on in CSE 232.

A more advanced class would discuss constraints on parameterized types (called "concepts"), value template arguments (which allow compile time creation of types with different constants), template argument deduction (which allow disambiguation of overloaded functions).

We also won't be discussing many other advanced usages of lambda expressions and compile time optimizations.

Attribution

Please ask questions via Piazza

Dr. Joshua Nahum

www.nahum.us

EB 3504



CREDITS: This presentation template was created by [Slidesgo](#), and includes icons by [Flaticon](#), and infographics & images by [Freepik](#)

© Michigan State University - CSE 232 - Introduction to Programming II